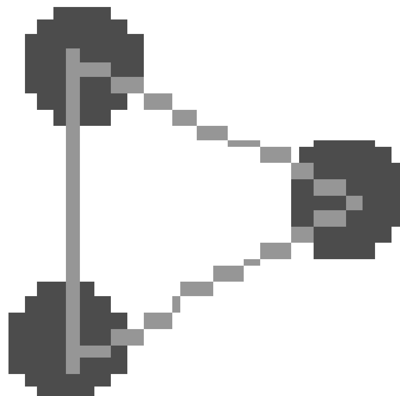
Algorithmische Graphentheorie im Unterricht unter Verwendung objektorientierter Datenstrukturen

Beim Fachbereich Mathematik der Bergischen Universität/Gesamthochschule Wuppertal eingereichte Dissertation zur Erlangung des Grades

Dr. paed.

von

Rainer Eidmann



Algorithmische Graphentheorie im Unterricht unter Verwendung objektorientierter Datenstrukturen

A Einleitung 2

B Didaktik und Methodik 11

I Didaktische und methodische Analyse der Behandlung von Graphen und Graphentheorie im Unterricht 11

II Didaktische und methodische Analyse der Verwendung von objektorientierten Datenstrukturen 44

III Konkrete methodisch-didaktische Bemerkungen zur Durchführung von Unterrichtsreihen zum Thema algorithmische Graphentheorie 62

C Skizzen von Unterrichtsreihen 149

I Knoten,Kanten,Bäume,Wege und Komponenten 152

II Suchverfahren in Graphen,Baum-und Pfadalgorithmen,Kreise,Gerüste 163

III Alle Pfade sowie minimaler Pfad zwischen zwei Knoten 193

IV Eulerbeziehung,ebene Graphen und Färbbarkeit 205

V Euler- und Hamiltonlinien 222

VI Graph als Netzplan 244

VII Graph als endlicher Automat 258

VIII Graphen als Relationen 271

IX Maximaler Netzfluss 283

X Maximales Matching 297

XI Gleichungssystem 316

XII Markovketten/Reduktion von Graphen 328

XIII Kosten und Transportprobleme,optimales Matching 380

XIV Das Chinesische Briefträgerproblem 413

D Zusammenfassung 421

E Literaturverzeichnis 425

F Anhang 429

I Beschreibung der Datenstruktur 430

II Die Bedienung des Programms Knotengraph 454

III Eigenschaften der verwendeten Programmiersprache Delphi 485

IV Unterrichtspläne zu ausgewählten Unterrichtsstunden 511

V Ergebnisse und Antworten von Fragen an Schülern zu den Unterrichtsreihen,Beispiel einer Facharbeit,UML-Diagramme,Update 542

VI Quelltextlisting der verschiedenen Programme,Lösungen der Klausuraufgaben,Beschreibung der Objektmethoden im PDF-Format auf CD

A Einleitung:

Die Graphentheorie wurde begründet durch den Mathematiker Euler (1707 bis 1782),der 1736 das berühmte Königsberger Brückenproblem löste.Dabei wurde ein Rundweg gesucht,der über die sieben Brücken des Flusses Pregel unter Einbeziehung von zwei Inseln so führen sollte, dass jede Brücke nur einmal benutzt wird.

Euler ordnete jedem der durch den Flusslauf vorgegebenen voneinander getrennten (Land-)Gebieten einen Buchstaben (A,B,C...) zu und faßte die Brükken als Übergänge zwischen den Gebieten mittels der Bezeichnungen AB,BC usw. auf.Obwohl sich Euler dabei noch nicht der heute üblichen Graphen-Terminologie bediente,benutzte er damit doch indirekt die Grundstruktur eines Graphen,der aus einer Menge von Knoten besteht,die durch gegenseitige Relationen d.h. Beziehungen -nämlich den Kanten (hier Brücken genannt)- miteinander in Wechselwirkung stehen.

Er verallgemeinerte die Aufgabenstellung und zeigte,dass bei einem allgemeinen „Gebiets-Brücken-Problem" ein Rundweg (Eulerlinie),der jede Brücke genau einmal enthält,nur dann möglich ist,wenn in jedes Gebiet eine gerade Zahl von Zugangsbrücken führt,als auch,dass ein Nicht-Rundweg,der über jede Brücke genau einmal führt,genau dann zwischen zwei Gebieten konstruiert werden kann,wenn diese beiden Gebiete die einzigen sind,die eine ungerade Anzahl von Zugangsbrücken besitzen.Damit war auch bewiesen,dass es beim Königsberger Brückenproblem keine Lösung geben kann,weil jedes der (mehr als zwei) Gebiete eine ungerade Zahl von Zugangsbrücken hatte.

Ungefähr einhundert Jahre später 1847 entdeckte der Physiker Kirchhoff bei der Untersuchung der Ströme in elektrischen Netwerken,dass eine allgemeine Lösung dadurch möglich ist,dass das Netzwerk als Graph dargestellt wird, und ein den Graphen aufspannender Baum gesucht wird.

Einen anderen Zugang zur Graphentheorie wählte Caley 1857,indem er die chemische Struktur von Isomeren gesättigter Kohlenwasserstoffe mit Hilfe von Bäumen ermittelte.

Der Mathematiker Hamilton warf 1859 die Frage auf,ob ein Rundweg auf einem regulären Dodekaeder mit 20 Knoten (Ecken) und 30 Kanten möglich ist,so dass jede Ecke genau einmal besucht wird.Er formulierte die Aufgabe als Spiel,das sich am besten als Graph darstellen läßt,in dem die Knoten (Ekken) Städte und die Kanten Wege zwischen diesen Städten bedeuten.Die Lösung des Problems sind die Hamilton-Kreise.Sucht man den Hamilton-Kreis mit der kleinsten Wegstreckenlänge,führt dies zum Problem des Traveling-Salesman.

Ebenfalls seit der Mitte des 19. Jahrhundert ist das Vierfarbenproblem auf Graphen bekannt,bei dem es darum geht,die Knoten eines ebenen Graphen mit maximal vier Farben so zu färben,dass jeweils durch Kanten verbundene Knoten verschiedene Farben erhalten.Die Fragestellung ist äquivalent zum Problem,ob es immer möglich ist,bei der Färbung der Länder einer Karte nur mit vier Farben auszukommen.

Ein Beweis dieses Vierfarbenproblems schien lange Zeit unmöglich,bis nach vielen vergeblichen Ansätzen erst vor ca. 20 Jahren ein Beweis unter Einsatz eines Computerprogramms zum ersten Mal gelang.

So wurde die Graphentheorie an Hand von verschiedenen Problemen und Aufgabenstellung von mehreren Personen begründet.Erst durch das Buch von König „Theorie der endlichen und unendlichen Graphen" 1936 wurden die bis dahin vorliegenden Ergebnisse einheitlich zusammengefaßt.

Seitdem hat sich die Graphentheorie im großen Umfang weiterentwickelt und hat immer neue Anwendungsgebiete erschlossen.

Graphen werden eingesetzt zur Lösung von Problemen des Operations Research und der linearen Optimierung,z.B. von Minmalkosten-, Maximalfluss- oder

Transportproblemen in Netzwerken,zur Ermittlung eines kostengünstigen,kürzesten Verlaufs für eine Versorgungsleitung vom Versorger zum Verbraucher,zur gezielten,optimalen Auswahl von Zuordnungen (Funktionen auch Matching genannt),die unter bestimmten,vorgegebenen Gesichtspunkten z.B. der Zuordnung von Maschinen zu verschiedenen auf ihnen auszuführenden Tätigkeiten die bestmögliche Effizienz ermöglichen,zur Darstellung von Relationen zwischen den Elementen einer Menge und Ermittlung ihrer Rangfolge,z.B. der Darstellung von psychologischen,zwischenmenschlichen Beziehung in einer Gruppe von Personen,zur Bestimmung aller und der kürzesten Straßenverbindung zwischen zwei Orten in einer Karte,umgangssprachlich auch Routing-Problem genannt,zur Lösung von kombinatorischen Problemen,zur Modellierung von Vorgängen,die mit den Mitteln der Wahrscheinlichkeitstheorie zu berechnen sind,z.B. mittels Pfadbäumen und Markovketten,in der Physik zur Veranschaulichung von Elementarteilchenprozessen mit Hilfe von Feyman-Diagrammen,in der Informatik zur Veranschaulichung von grundlegenden Datenstrukturen wie z.B. Listen und Bäume bzw. zur Veranschaulichung von Modellen der theoretischen Informatik wie z.B. Automaten.

Mit dieser Auflistung sind die Anwendungsgebiete der Graphentheorie noch lange nicht alle erfaßt.Also scheint die Graphentheorie insbesondere für mathematische aber auch außermathematische Fachwissenschaften grundlegende Verfahrensweisen bereitzustellen,um die dort auftretenden Problemstellungen angemessen lösen zu können.

Demgegenüber verwundert es,dass die Graphentheorie im Lehrplan bzw. in den Richtlinien und in der Unterrichtspraxis des Mathematikunterrichts der Schulen der Sekundarstufe I und II praktisch - bis auf ganz wenige Ausnahmen z.B. der Darstellung von Wahrscheinlichkeitspfaden in der Stochastik - nicht vorkommt.

Die Stoffauswahl wird über große Teile statt dessen von den Gebieten Algebra,Analysis und Geometrie bestimmt,die weitgehend ohne die Benutzung von Graphen auskommen.

Die neuen Richtlinien für die Sekundarstufe II Mathematik NRW,die im Jahre 1999 in Kraft treten, enthalten beispielsweise das Wort Graphentheorie überhaupt nicht,und das Wort Graph kommt in ihnen nur in der Bedeutung als Funktionsgraph vor.(Graphen werden nur,wenn auch nicht in den Richtlinien explizit erwähnt,im Gebiet Wahrscheinlichkeitsrechnung zur Modellierung von Wahrscheinlichkeitsprozessen benutzt.)

Im Informatikunterricht dagegen,werden Graphen schon seit längerem und öfter zur Veranschaulichung und zur Lösung von komplexeren Problemen,vor allen Dingen aber auch als Mittel der Darstellung von Datenstrukturen eingesetzt.

Liegt das Fehlen von graphentheoretischen Unterrichtsinhalten im Mathematikunterricht vielleicht darin begründet,dass die Verfahren der Graphentheorie sehr oft Algorithmen sind,die Operationen mit den Kanten und Knoten eines Graphen erfordern,die meistens nicht nur rechnerischer,numerischer Art sind,sondern das Auswählen, Umordnen, Nummerieren, Neubezeichnen,Hinzufügen oder Löschen von Knoten und Kanten unter bestimmten Gesichtspunkten bedeuten,so dass sie im normalen Unterricht ohne Einsatz von komplexeren Hilfsmitteln wie Computern nur mit den Werkzeugen Bleistift und Papier alleine mühsam durchzuführen sind?

Zwar lassen sich die meisten Algorithmen anschaulich beschreiben,ihre Durchführung erfordert jedoch insbesondere bei größeren Graphen das mühevolle Zeichnen von vielen neuen,jeweils immer wieder veränderten Graphenbildern,bis das Verfahren terminiert.

Demgegenüber sind die Algorithmen der Algebra oder Analysis reine Zahlenalgorithmen oder Termumformungsalgorithmen,die leichter mit den Hilfsmitteln Papier und Bleistift zu dokumentieren sind.Auch in der Geometrie läßt sich ein Algorithmus mit den Werkzeugen Zirkel und Lineal leichter als in der Graphentheorie,nämlich als Konstruktionsalgorithmus d.h. durch Zeichnen von

(meistens) einer einzigen Figur unter Dokumentation der erforderlichen Hilfslinien durchführen.

Die Situation hat sich mit der Bereitstellung von Computern als Werkzeug gewandelt.Das mühevolle Zeichnen immer neuer Graphen wird jetzt abgelöst durch das Schreiben eines Programms,das diesen Graphenalgorithmus ausführen kann und den Algorithmus durch zeichnerische Darstellung auf einem Computermonitor visualisiert.

Dieses könnte eine Erklärung sein,warum die Graphen längst Einzug in die Schulinformatik genommen haben,in der Rechner von der Natur des Faches her eingesetzt werden.Im Mathematikunterricht dagegen steht das Hilfsmittel Computer dagegen nicht zur Verfügung,zum einen,weil dort oft die nötigen Programmierkenntnisse als Grundbedingung fehlen,zum anderen weil die technischen Voraussetzungen in den Klassenräumen nicht vorhanden sind,und der Unterricht nur mit den Hilfsmitteln Kreide, Tafel und Taschenrechner auskommen muß.

Durch das immer weitere Vordringen der Computer in alle Bereiche unser Lebenswelt und damit auch in die Schule könnte sich die oben geschilderte Situation grundlegend ändern,und der Computer steht vielleicht demnächst als Unterrichtswerkzeug auch für den Mathematikunterricht bereit.Dann wäre der Weg frei für eine algorithmische Behandlung der Graphentheorie mit Hilfe von Programmen im Unterricht.

Sollte dann die Graphentheorie auch verstärkt in den Lehrplan aufgenommen werden?Ist es sinnvoll,dass sie dann notwendigerweise traditionelle Stoffgebiete verdrängen würde?Welche Probleme der Graphentheorie sollten im Unterricht behandelt werden?Eignet sich die Behandlung von Graphentheorie dazu,dieselben Lernziele wie mit konventionellen Unterrichtsinhalten zu erreichen?Lassen sich geeignete Klausuraufgaben stellen?

Eine Antwort auf diese Fragen kann nur durch die Diskussion allgemein didaktischen Prinzipien des Mathematikunterrichts beantwortet werden.Sinnvoll ist es dabei auch die didaktischen Prinzipien des Fachs Informatik mit einzubeziehen,da bei der algorithmischen Behandlung der Graphentheorie mit Hilfe von Computerprogrammen beide Fächer aufs engste miteinander verknüpft werden.Dies soll in den folgenden Kapiteln vorgenommen werden.

Ein anderer Aspekt ergibt sich wenn man bedenkt,dass es wünschenswert ist, dass die Algorithmen in möglichst anschaulicher Weise graphisch wie bei dem manuellen Anfertigen von aufeinanderfolgenden Zeichnungen der immer wieder veränderten Graphen einschließlich der Übergänge zwischen diesen einzelnen Phasen dargestellt werden sollten.Dieses erfordert komplexe Algorithmen, die auf graphischen Benutzeroberflächen und Betriebssystemen ablaufen müssen.Die Gefahr besteht,dass die Programmierung dieser Algorithmen sich zu sehr im Detail des Entwickelns grundlegender Strukturen zur Bereitstellung der nötigen Graphiken verliert und die Algorithmen zu komplex werden,so dass der Blick für das Wesentliche verloren geht.

Hier setzt nun das Konzept der objektorientierten Programmierung an.Durch die Bereitstellung von Methoden werden Softwaretools geschaffen,die vom späteren Benutzer geerbt und in eigenen Algorithmen als Grundbausteine verwendet werden können.Daher ist es notwendig, das Konzept der objektorientierten Programmierung einzubeziehen und ebenfalls unter didaktischen Gesichtspunkten zu erörtern.

Das dieser Arbeit zu Grunde liegende Programmsystem Knotengraph zur Entwicklung und Demonstration von Algorithmen der Graphentheorie im Unterricht,läßt sich unter drei verschiedenen methodisch-didaktischen Konzepten im Unterricht verwenden.

Erstens stellt es in Form eines Softwaretools und Unterrichtswerkzeuges Softwarebausteine einer Klassenbibliothek in Form von Objekten (der Programmiersprache Delphi) zur Programmierung von Graphenalgorithmen zur Verfügung.Durch Vererbung der vorgegebenen Objektmethoden ist es möglich,sich bei der Programmierung eines Graphenalgorithmus auf die wesentlichen mathe-

matisch-algorithmisch interessanten Aspekte zu konzentrieren und sich nicht in Einzelheiten der Eigenschaften sowie der zeichnerischen Darstellung von Graphen zu verlieren.Trotzdem kann der Ablauf des Algorithmus anschließend als zeichnerische Darstellung in einem Anwendungsfenster mit dem Komfort einer graphischen Windowsanwendung vom Benutzer verfolgt werden.

Diese Konzeption reicht so weit,dass sogar eine ganze Benutzeroberfläche,die schon alle Menüs zur graphischen Erzeugung und Editieren der Kanten und Knoten eines Graphen samt Dateihandling (Speichern und Lesen von Graphen) sowie Ausgabeverwaltung von Ergebnissen enthält,mittels visueller Vererbung (ab der Version Delphi 2.0) bereitgestellt werden kann.Der diese Oberfläche durch Vererbung benutzende Programmierer kann sofort damit beginnen,die eigentlich mathematisch interessanten Graphenalgorithmen zu implementieren und als weiteres Menü der Oberfläche hinzuzufügen.Diese Oberfläche inklusive der vorgegebenen Objekte soll in dieser Arbeit als Entwicklungsumgebung Knotengraph (EWK) bezeichnet werden.

Als zweite methodisch-didaktische Konzeption wird eine bezüglich der Darstellung von Graphen reduzierte Version des Programms Knotengraph in Form von entsprechenden Units bereitgestellt,die als zeilenorientierte Textanwendung abläuft und in dieser Arbeit als Consolenanwendung bezeichnet werden soll.Sie enthält alle grundlegenden Objektdatenstrukturen des Graphen in etwas vereinfachter Form und demonstriert die grundsätzlichen Struktur eines objektorientierten Graphentyps auf der Grundlage von Objektlisten.

Sie soll als Vorlage für ein mit Schülern durchzuführendes Programmierungsprojekt dienen,in dem gezeigt wird,wie das Programm Knotengraph bzw. dessen Entwicklungsumgebung grundsätzlich arbeitet und wie eine wiederverendbare,erweiterbare,objektorientierte Graphenstruktur prinzipiell aufzubauen ist.

Hierbei wurde auf die graphische Darstellung der Graphen verzichtet und statt dessen die Form der Textanwendung bezüglich der Ein-und Ausgabe gewählt,um den Blick auf das Wesentliche nicht zu verlieren.

Knoten werden dabei in der Reihenfolge der Knotenliste des Graphen und Kanten in der Reihenfolge der Kantenliste jeweils mittels ihrer Bezeichner dargestellt,wobei diese Informationen bei jeder Änderung des Graphen aktualisiert werden.Die Darstellung erfolgt bei Delphi Version 1.x mittels der komfortabeln Unit WINCRT und bei höheren Versionen durch die Consolenoption des Compilers.

Diese zweite Konzeption,die im folgenden Consolenanwendung Knotengraph genannt wird (CAK),eignet sich vor allen Dingen zum Einsatz im Informatikuntericht,während die erste Konzeption die mathematischen und informatischen Aspekte jeweils in gleicher Weise berücksichtigt.

Die dritte methodisch-didaktische Konzeption ist der Einsatz des **fertig codierten** (und compilierten) Programms Knotengraph als Demonstrationsprogramm für Graphenalgorithmen beispielsweise zum Einsatz im Mathematikunterricht,nachdem diese z.B. in Form von Verbalalgorithmen besprochen wurden, oder zum selbständigen Entdecken (der Prinzipien) dieser Algorithmen durch Schüler an Hand des Demomodus dieses Programms,wodurch eine Beschreibung des Ablaufs durch die Schüler selber erstellt werden kann.Dazu wird ein Demonstrationsmodus (Demomodus oder Einzelschrittmodus) für jeden der vorgegebenen Graphenalgoritmen bereitgestellt,in dem der Ablauf zeitverzögert und unter Darstellung der wesentlichen Schritte beobachtet werden kann.Diese Art der Benutzung soll in dieser Arbeit als didaktisches-methodisches Werkzeug Knotengraph (DWK) bezeichnet werden.Außerdem soll der Quellcode der Algorithmen der Untermenüs der Menüs Pfade und Anwendungen dieses (fertig codierten) Programms als Musterbeispiele für mit Schülern zu entwickelnde Graphenalgorithmen bzw. als Anregung für eigene,selbständige Entwicklungen für die Entwicklungsumgebung Knotengraph (EWK) dienen.

Bei der Erstellung der Module des Programms Knotengraph standen diese dritte Konzeption und die erste Konzeption in einem gewissen Widerspruch zueinander.

Denn auf der einen Seite sollen die Algorithmen in Form der bereitgestellten Methoden möglichst überschaubar und einfach sein und nur die wesentlichen mathematischen Schritte enthalten,um der ersten Konzeption gerecht zu werden,auf der anderen Seite sollen sie in Bezug auf die dritte Konzeption möglichst viele Anzeigemöglichkeiten zur Demonstration des Graphenalgorithmus und komfortable Bedienungselemente enthalten.Es wurde versucht jeweils beiden Aspekten gerecht zu werden,teilweise schließen sie sich jedoch aus.In den Quelltextlistings dieser Arbeit (aber nicht im Anhang) wird daher auf die Darstellung des Quelltextes für den Demomodus (und auch auf Anweisungen zum Setzen eines vom Anwender ausgelösten Algorithmusabbruchs) verzichtet.

Im Anschluß an diese Einleitung wird zunächst im Abschnitt BI und BII didaktisch und methodisch analysiert,ob und in welchem Umfang Probleme der Graphentheorie Unterrichtsinhalte des Mathematik- und Informatikunterrichts sein sollten,und inwieweit es sinnvoll ist,bei der Implementierung von Graphenalgorithmen objektorientierte Datenstrukturen einzusetzen.Danach (Kapitel B III) schließt sich die konkrete methodische Beschreibung sowie didaktische Kommentierung der Durchführung von Unterrichtsreihen zum Thema Algorithmische Graphentheorie an,die der Verfasser dieser Arbeit mehrfach mittels Einsatz des Programmsystems Knotengraphs unter Benutzung der drei weiter oben genannten Konzeptionen (EWK,DWK und CAK) durchgeführt hat.Dabei wird in Einzelheiten erläutert,wie das Programmsystem Knotengraph im Unterricht eingesetzt werden sollte,wie eine komplette Gesamtkonzeption Graphentheorie auf der Sekundarstufe II aussehen und wie konventionelle Lerninhalte mit eingebunden werden können,als auch wie eine Einführung und ein kompletter Kursus der objektorientierten Programmierung unter Einbeziehung von Graphenstrukturen und deren Algorithmen gestaltet werden kann.

Unter anderem wird hier ein didaktisch-methodischer Einstieg in die objektorientierte Programmierung mittels eines Projekts,das die graphische Darstellung eines Graphen zum Thema hat,vorgestellt,woran sich die Beschreibung eines zweites didaktisch-methodischen Einstiegsprojektes zum Thema Aufbau einer objektorientierten Listenstruktur gemäß der Konzeption des abstrakten Datentyps (ADT) anschließt,die eine Listenstruktur als Objekt bereitstellt,die für verschiedenste Anwendungen flexibel angepaßt und eingesetzt werden kann,da sie nicht auf die Verwaltung von bestimmten Daten spezialisiert ist.

Diese Listenstruktur ist dann die Grundlage für den Aufbau der objektorientierten Graphenstruktur für die Konzeption CAK im Unterricht.Um dabei nur den Blick auf das Wesentliche zu lenken,wird dabei aus methodischen Gründen auf eine graphische Darstellung des Graphen verzichtet und dafür eine textorientierte,zeilenorientierte Ein-und Ausgabe als Consolenanwendung realisiert (Konzeption CAK).

Diese Version sollte zum Einsatz kommen,wenn es darum geht,in einem Programmierungsprojekt zu zeigen,wie ein noch auf unterschiedliche nachträgliche Wünsche hin zu verändernder Graph mit Hilfe von objektorientierten Listendatentypen durch Vererbung aufgebaut werden kann.

Im Abschnitt BIII wird erläutert,wie durch einen an bestimmten Stellen durch Vorgaben methodisch gelenkten Unterrichtsgang,die grundlegende Struktur des Graphen als Programmierungsprojekt von Schülern zum großen Teil selber entwickelt und der entsprechende Quelltext erstellt werden kann.

Wird die Objekt-Listenstruktur spezialisiert auf die Strukturen Keller und Schlange und der Objekt-Graph auf die Struktur des Binärbaums,können diese im Fach Informatik bisher traditionell enthaltenden Lerninhalte ohne weiteres mit in den Unterrichtsgang einbezogen werden.

Als Anwendungen dieser Objekt-Graphenstruktur wird dann aufgezeigt,wie der Quelltext zu fünf der wichtigsten Algorithmen der Graphentheorie erstellt werden kann:Tiefer Baumdurchlauf (als Beispiel für Durchlaufstrategien), Labyrinthproblem d.h. minimaler Pfad bzw. alle Pfade zwischen zwei Knoten, Hamiltonkreise, geschlossene/offene Eulerlinien sowie Untersuchung der Färbbarkeit.

(Wobei unter den fünf Algorithmen ausgewählt werden sollte,nicht unbedingt alle realisiert werden müssen und natürlich auch jederzeit andere Graphenalgorithmen als Anwendungen als die hier genannten gewählt werden können.Weitere Beispiele hierzu finden sich in Abschnitt C.)

Die Realisation als Text-bzw. Consolenanwendung ist auf die Dauer unbefriedigend,da sie nicht die für das Betriebssystem Windows typischen Eigenschaften von Programmen in Form einer graphischen Benutzeroberfläche ausnutzt.An dieser Stelle bietet sich deshalb der Übergang zur EWK an,wobei der Umstieg hier besonders leicht fällt,da die den Schülern bekannten mittels CAK erstellten Algorithmen mit kleineren Änderungen übernommen werden können und die Datenstruktur des Graphen der EWK den Schülern aus der Konzeption der CAK (da,darauf aufbauend) prinzipiell bekannt ist.

Aber natürlich kann auch direkt mit der Konzeption der EWK begonnen werden, wobei dann bei den Schüler schon grundlegende Kenntnisse objektorientierten Programmieren vorhanden sein müssen.Wenn z.B. das oben beschriebenen Einstiegsprojekt zur Einführung in die objektorientierte Programmierung durchgeführt wurde,ist die prinzipielle Art der zeichnerischen Darstellung eines Graphen den Schülern aus diesem Projekt her schon bekannt.Dann sollte als nächstes ein Einblick in die Datenstruktur und die wichtigsten Methoden der EWK gegeben werden.Entsprechende Hinweise hierzu finden sich Kapitel B III. Außerdem wird hier schließlich ausführlich die Methodik der Konzeption DWK erläutert.

Der Abschnitt C dieser Arbeit bildet dann die detaillierte Fortsetzung der didaktisch-methodischen Erörterung dieser beiden letztgenannten Konzeptionen in Form von konkreten Unterrichtsskizzen und Vorschlägen,um ausgewählte Probleme der Graphentheorie und des Operations Research zu behandeln.Er enthält eine ausführliche didaktisch-methodische Darstellung der Lösung von verschiedenen Problemen der algorithmischen Graphentheorie und des Operations Research in der Form von Unterrichtsprojekten durch das Programm oder die Entwicklungsumgebung Knotengraph mittels der Konzeptionen DWK und EWK.Die verwendeten Algorithmen werden ausführlich erläutert,und der Programm-Quelltext kommentiert.Das mathematische Umfeld,die Voraussetzungen,Beweise und Hintergründe werden eingehend erläutert.Zu jedem Algorithmus werden jeweils ein problemorientiertes Einstiegsprojekt sowie mehrere Anwendungsbeispiele und Übungsaufgaben angegeben,deren Aufgabenstellung dann als Graphen modelliert werden können,so dass das Problem bei der Konzeption DWK durch Anwahl des dem Algorithmus entsprechenden Menüs des (fertig codierten) Programms Knotengraphs automatisch oder aber durch Erstellen eines entsprechenden Quelltextes und der dazugehörigen Graphenstrukturen durch die Schüler mit Hilfe der Entwicklungsumgebung bei der Konzeption EWK gelöst werden kann.

Folgende Probleme der Graphentheorie,für die das Programm Knotengraph Lösungsalgorithmen bereitstellt,die aber auch als Musterbeispiele für Entwicklungen mittels der Entwicklungsumgebung Knotengraph dienen sollen,wurden ausgewählt:

Pfadprobleme (Wege,Bäume,Kreise):

Bestimmung aller Pfade zu den Knoten des Graphen von einem Knoten aus,Inorderdurchlauf eines Binärbaums,Bestimmung aller Kreise des Graphen von einem Knoten aus,Bestimmung aller minimaler Pfade zu den Knoten des Graphen von einem Knoten aus, Bestimmung der Anzahl der auf Pfaden erreichbaren Zielknoten von einem Knoten aus,Tiefer Baumdurchlauf des Graphen von einem Knoten aus,Breiter Baumdurchlauf des Graphen von einem Knoten aus,Abstand von zwei Knoten des Graphen (Labyrinthproblem),Binäres Suchen in einem geordneten Binärbaum,Bestimmung aller Pfade zwischen zwei Knoten des Graphen,Bestimmung eines minimalen Gerüstes des Graphen

Allgemeine Graphenprobleme (Anwendungen):

Bestimmmung eines Zeitplans für einen Graphen,der als Netzplan aufgefaßt wird,nach der CPM-Methode,Bestimmung von Hamiltonkreisen eines Graphen mit

Lösung des Travelling- Salesman-Problems,Bestimmung der geschl. Eulerlinien eines Graphen,Untersuchung,ob ein vorgegebener Graph (die Knoten) mit einer vorgegebenen Anzahl von Farben gefärbt werden kann,Bestimmung einer offenen Eulerlinie in einem geeigneten Graph,Simulation eines endlichen Automaten,Bestimmung der Ordnungsrelation sowie der transitiven und reflexiven Hülle eines als Relation aufgefaßten Graphen,Bestimmung des maximalen Netzflusses nach dem Algorithmus von Ford-Fulkerson in einem Graphen mit Quellen- und Senkenknoten,Bestimmung eines maximalen Matchings nach der Methode Suchen eines erweiternden Wegs in einem Graphen,Anschauliches Lösen eines Gleichungssystems nach dem Algorithmus von Mason auf einem Graphen,Bestimmung der Übergangswahrscheinlichkeiten und mittleren Schrittzahlen auf einem Graphen,der als absorbierende Markovkette aufgefaßt wird,Bestimmung der Übergangswahrscheinlichkeiten und mittleren Schrittzahlen auf einem Graphen,der als stationäre Markovkette aufgefaßt wird,Reduzieren eines Signalflussgraphen bzw. Bestimmung der Lösungen einer absorbierenden Markovkette durch Reduzierung des Graphen,Lösungen des Minimalen bzw. Maximalen Kostenproblems und Ermittlung eines geeigneten Graphflusses,Lösungen des allgemeinen Transportproblems bzw. des Hitchcockproblems sowie als Verallgemeinerung Lösen von linearen Optimierungsproblemen auf Graphen,Bestimmung eines Optimalen Matchings auf einem bipartiten Graph,Lösen des Chinesischen Briefträgerproblems auf einem gerichteten Graphen.

Die Darstellung und Erörterung der genannten Probleme in den einzelnen Kapiteln erfolgt jeweils so,dass sie als Skizze bzw. Grobgerüst einer Unterrichtssequenz,die die Erläuterungen des mathematischen Hintergrunds,der notwendigen Definitionen,des Einstiegsproblems sowie von geeigneten Anwendungsaufgaben einschließt,aufzufassen ist.Manche Erörterungen der mathematischen Voraussetzungen sind dabei allerdings nur als Hintergrundinformation für den Lehrer gedacht,und auch die angegebenen Beweise müssen nicht alle im Unterricht dargestellt werden.Die Übungsaufgaben sollten jeweils noch durch weitere selbst zu erstellende Aufgaben ergänzt werden.Jeder Algorithmus wird als Verbalalgorithmus und danach erst im Quellcode angegeben.Auf Grund der Erläuterungen der Algorithmen sollte es möglich sein,die Algorithmen nachzuvollziehen und gegebenenfalls auch ähnliche neue Algorithmen selbst zu entwerfen.

Abschließend werden in einer Zusammenfassung noch einmal die didaktisch-methodischen Aspekte und Vorteile des Einsatzes des Programm Knotengraph im Unterricht gemäß den verschiedenen Konzeptionen als Unterrichtstool zur Erstellung von Graphenalgorithmen mit einer graphischen Benutzeroberfläche (EWK),als Programmierungsprojekt unter dem Aspekt der Entwicklung der Datenstruktur eines Graphen (CAK) bzw. als fertiges Werkzeug zur Lösung und Darstellung von Problemen der Graphentheorie (DWK) dargestellt.

Im Anhang wird eine Übersicht über den Aufbau und den Zusammenhang der im Programmsystem Knotengraph (EWK und DWK) verwendeten objektorientierten Datenstrukturen gegeben.Dabei wird auch erläutert,wie man für eigene Zwecke neue Graphentypen durch Vererbung von dem durch die Entwicklungsumgebung Knotengraph vorgegebenen Graphdatentyp ableitet,wodurch sich jederzeit eigene auf dem System aufbauende Anwendungen programmieren lassen.Es folgt eine Bedienungsanleitung der Benutzeroberfläche des Programms Knotengraph,an die sich eine Darstellung der Eigenschaften der Programmiersprache Delphi sowie eine Erörterung von deren Besonderheiten,soweit sie im Programm Knotengraph zum Einsatz kommen,anschließen.Detailierte Unterrichtspläne mit didaktisch-methodischen Erläuterungen zu ausgewählten Unterrichtsstunden sowie die Auswertung von Fragen an Schüler zu den Unterrichtsreihen,die von einem Schüler erstellte Facharbeit zu Objekt-Datenstrukturen von Graphen und Algorithmen zu Bäumen in Graphen als auch UML-Diagramme der Objektbeziehungen der Konzeptionen CAK,EWK und DWK beenden den Anhang.

Auf der zu dieser Arbeit gehörenden CD findet sich eine Beschreibung aller Methoden der Objekte des Programmsystems Knotengraphs sowie Quelltextlistings aller in dieser Arbeit erwähnten Programme und die Lösung von Klausuraufgaben.

Das Programm Knotengraph liegt auf der CD in mehreren Versionen vor:

1.

Knotengraph als Programm-File KProjekt.exe als Demonstrationsprogramm zur Lösung der oben dargestellten Problemen der Graphentheorie zur Verwendung z.B. im Mathematikunterricht als sofort unter Windows 95, Windows 98 oder Windows NT 4.0 ablauffähige Anwendung gemäß der Konzeption DWK.

2.

Knotengraph als zu KProjekt.exe zugehörige in die Entwicklungsumgebung von Delphi ladbare Programmquelltextversion KProjekt.dpr einschließlich der Quelltexte aller Units (Unit UList,Unit UGraph, Unit UInhgrph, Unit UKante, Unit UAusgabe, Unit UPfad, Unit UMath1, Unit UMath2, Unit UKnoten, Unit UForm) als Dokumentation der Datenstruktur,der Methoden der Objekte und Komponenten und der Quelltexte der bestehenden Anwendungen. (DWKQuell)

(Für Windows 95, Windows 98 sowie Windows NT 4.0 und jeweils Delphi 2.0 oder höhere Version,getestet auch mit 6.0.Mit Hife von Borland Kylix kann Knotengraph möglicherweise auch auf Linux oder Unix portiert werden.)

Bedeutung der Units:

UList:enthält die grundlegende objektorientierte Listenstruktur
UGraph:enthält einen Objekt-Graphen (ADT) ohne graphische Darstellungsmöglichkeiten
UInhgrph:fügt dem Graphen der vorgenannten Unit die graphischen Darstellungsmöglichkeiten hinzu.
UKante: Unit,die zur Kantenform gehört,wodurch Kantendaten in einem Eingabefenster eingegeben werden können.
UAusgabe: Unit,die zur Ausgabeform gehört,wodurch Ergebnisse in einem Fenster dargestellt werden können.
UPfad:enthält die Objektdatenstrukturen und die Methoden,um die Pfadalgorithmen (aufgerufen durch die Menüs von Pfade) zu realisieren.
UMath1:enthält die Objektdatenstrukturen und Methoden,um die Anwendungsalgorithmen 1. Teil (aufgerufen durch die Menüs von Anwendungen) zu realisieren.
UMath2:enthält die Objektdatenstrukturen und Methoden,um die Anwendungsalgorithmen 2. Teil (aufgerufen durch die Menüs von Anwendungen) zu realisieren.
UKnoten:enthält als Form Knotenform eine Oberfläche zur Verwaltung der Knoten und Kanten des Graphen (Erzeugen, Löschen, Editieren, Speichern, Laden,Eigenschaften usw.),die an Knotenformular in der Unit UForm visuell vererbt wird.
UForm:enthält Knotenformular als Hauptform,d.h. Hauptanwendungsfenster des Programms Knotengraph,das visuell alle Eigenschaften der Knotenform aus der vorigen Unit erbt,und außerdem zusätzlich die Menüs (Menü-Objekt-Ereignismethoden) zum Aufrufen der Algorithmen Pfade und Anwendungen (und der damit im Zusammenhang stehenden Methoden) enthält.

3.

Knotengraph gemäß der Konzeption EWK als reduzierte Version (mit Quelltexten) ohne die Algorithmen der oben genannten Menüs Pfade und Anwendungen und ohne die Units UPfade,UMath1,UMath2 mit einer visuell vererbten Quelltext-leeren Benutzeroberfläche Knotenformular (Unit UForm),so dass diese Version als Ausgangspunkt für von den Schülern als Programmierungsprojekte zu erstellende Graphenalgorithmen mit graphischer Oberfläche benutzt werden kann.Durch die Quelltext-leere Unit UForm kann wie bei einem neuen Projekt begonnen werden,jedoch enthält die Form Knotenformular dieser Unit schon alle Eigenschaften der Form Knotenform der Unit UKnoten zur Verwaltung eines Graphen.Je nach Auswahl des Graphenproblems sollte eine der Units UPfade,UMath1 oder UMath2 (natürlich auch mit anderem Bezeichner) mit dem Teilquelltext des zugehörigen Objektgraphen neu erstellt werden.
(Version 2 kann dabei für den Lehrer mit den Quelltexten der Units UPfad, UMath1 und UMath2 sowie UForm als Vorgabebeispiel benutzt werden.)

(Version für Windows 95, Windows 98 sowie Windows NT 4.0 und jeweils Delphi 2.0 oder höhere Version,getestet auch mit 6.0)

4.

Knotengraph als im Umfang von der Benutzeroberfläche her eingeschränkte (zeilenorientierte) Text- bzw. Consolenanwendung mit Quelltexten als Beispiel für ein Programmierungsprojekt gemäß der Konzeption CAK,das die grundlegenden Datenstrukturen zum Aufbau eines flexibel verwendbaren Objektgraphen auf der Grundlage von objektorientierten Listen demonstriert,als auch fünf der wichtigsten Algorithmen der Graphentheorie (Tiefer Baumdurchlauf, Hamiltonkreise,geschlossene/offene Eulerlinien,minimaler Pfad und alle Pfade zwischen zwei Knoten,d.h.Labyrinthproblem sowie Untersuchung der Färbbarkeit) als Anwendungen für den Unterricht bereitstellt.

(Units UListC,Unit UGraphC,Unit UInhgrphC,Unit MathC)

(Version für Windows 95, Windows 98 sowie Windows NT 4.0 oder Windows 3.1 und jeweils Delphi 1.0 oder 1.1,das mit Delphi 2.0 ausgeliefert wird.Bei Nichtverwendung der Unit WINCRT auch als Consolenanwendung unter Delphi – Versionen >=2.x realisierbar.)

Bedeutung der Units:

UListC:enthält die grundlegende objektorientierte Listenstruktur als reduzierte Version der Unit UList
UGraphC:enthält einen Objekt-Graphen (ADT) ohne textorientierte (und natürlich auch ohne graphischen) Darstellungsmöglichkeiten als reduzierte Version der Unit UGraph
UIngrphC:fügt dem Graphen der vorgenannten Unit die Möglichkeiten der textorientierten Eingabe Ausgabe sowie Darstellung hinzu.
UMathC:enthält die Objektdatenstrukturen und Methoden,um die oben genannten fünf Pfad- oder Anwendungsalgorithmen zu realisieren.

Das Hauptprogramm Crtapp.pas ist ein Pascal/Delphi-Hauptprogramm,das die Unit WINCRT benutzt.

5.

Mittels des C++-Builders von Borland/Inprise lassen sich auf der Grundlage der mit Delphi erstellten Entwicklungsumgebung EWK durch Vererbung aller dort definierten Objekte und Formulare sämtliche in Abschnitt C dieser Arbeit genannten Graphenalgorithmen (und natürlich darüber hinaus noch weitere Algorithmen) in der Programmiersprache C++ erstellen.Der C++-Builder besitzt nämlich dieselbe visuelle Komponentenbibliothek (VCL) sowie alle von TObject abgeleiteten vordefinierten Objektklassen wie Delphi.Dadurch ist es möglich,das unter Delphi erstellte Formular Knotenform (mit seinen sämtlichen Eigenschaften) visuell zu vererben und weitere darauf aufbauende Graphenalgorithmen als C++-Quelltext zu schreiben.Durch die Gleichheit der Objektklassen läßt sich der Delphi-Quellcode teilweise fast 1:1 in C++ übersetzen.

Ein Beispiel,das die Anwendung Alle Pfade (von einem Knoten aus) als C++-Programm auf der Grundlage der Delphi-Units realisiert,ist im Installationsverzeichnis CPPEWK gespeichert und wird im Anhang (PDF-File der CD) beschrieben.

Schließlich befinden sich in weiteren Verzeichnissen der CD noch die Quelltexte sowie die ausführbaren Programmfiles der weiteren in dieser Arbeit erwähnten oder besprochenen Programme,nämlich das Einstiegsprojekt und das Projekt Liste als Objekt und außerdem alle Beispielgraphen,die im Kapitel C erörtert werden,als Dateien.Alle Dateien der CD können mit Hilfe eines Setup-Programms auf die Festplatte kopiert werden.

B I Didaktische und methodische Analyse der Behandlung von Graphen und Graphentheorie im Unterricht

Während die Graphentheorie in den Lehrplänen der Mathematik der Sekundarstufen I und II kaum vorkommt,und auch in der Unterrichtspraxis Graphen nur im geringen Maße, und wenn dann fast nur als Hilfsmittel in der Wahrscheinlichkeitsrechnung zur Darstellung von Wahrscheinlichkeitspfaden benutzt werden,enthalten die Lehrpläne des Faches Informatik der meisten Bundesländer verpflichtend zumindestens die Behandlung von Spezialfällen von Graphen wie Listen und Bäume zum Thema Datenstrukturen.

In Verbindung mit der Einführung der neuen Mathematik in den Schulen in den 70-er Jahren wurden in den meisten Mathematik-Schulbüchern im Rahmen des Themas Topologie auch Inhalte der Graphentheorie berücksichtigt.Mit dem Ändern der Richtung der Mathematikdidaktik weg von der Strukturmathematik verschwanden diese Themen wieder aus den Büchern.

Die von H. Piper und G. Walther (Lit 7, S. 233) durchgeführte Untersuchung von Lehrplänen in den Jahren 1984/85 zeigt schon damals eine deutliche Reduzierung oder sogar Eliminierung von graphentheoretischen Themen aus den Lehrplänen der Bundesländer und aus den Schulbüchern der Sekundarstufe I und der Primarstufe.

Vorher waren dort oft Problemstellungen der Graphentheorie wie „Benachbarte Gebiete" und „Gebietsfärbungen", „Lagebeziehungen und Eigenschaften von Linien", „Wege in Netzen","Rundwege" bis hin zum Eulerproblem enthalten,die mit der Umstellung der Lehrpläne auch aus den Schulbüchern entfernt wurden.(Lit 7)

Diese Entwicklung - weg von Inhalten der Graphentheorie - wird von Piper und Walther als „offensichtliches Defizit" (der Entwicklung der Mathematikdidaktik) bezeichnet,weil sich gerade mit Hilfe der Graphentheorie auch Nicht-Mathematikern ohne aufwendige Vorbereitungen zeigen läßt,dass die Mathematik sich nicht in dem Nachvollziehen der vom unterrichtenden Lehrer vorgegebenen Verfahren bzw. Algorithmen erschöpft,sondern,dass es durchaus noch neue (teilweise ungelöste) Mathematikprobleme gibt,die man neu entdekken kann.

Außerdem bietet gerade die Graphentheorie die Möglichkeiten für nichtnumerische und "nicht streng sequenzierte" Unterrichtseinheiten,die geeignet sind,Phantasie und Kreativität der Schüler anzuregen,weil sie stark problemorientiert ausgerichtet sind.

Probleme der Graphentheorie sollten daher nach Piper und Walther „substanziell zur pädagogischen Konzeption eines modernen Mathematikunterrichts gehören und dürfen auch bei noch so starker zeitlicher Beanspruchung des Mathematikunterrichts durch den notwendig umfangreichen Anteil des Rechnens aus dem Curriculum nicht gestrichen werden.In ihnen dürfte in besonderer Weise das Anliegen der Reformpädagogik realisiert sein." (Lit 7 S. 240/241)

In diesem Kapitel soll die Notwendigkeit einer verstärkten Berücksichtigung (bzw. Wiedereinführung) des Themas Graphentheorie im (heutigen) Mathematikunthericht und Informatikunterricht sowie deren Bedeutung für beide Fächer diskutiert werden.Weil die Graphentheorie ein beide Fächer verbindendes Thema ist,ist es sinnvoll gleichzeitig beide Fächer in die didaktischmethodische Diskussion einzubeziehen.

Zunächst ist zu erörtern,welche Lernziele allgemein in beiden Fächern erreicht werden sollten,und welche Lernziele speziell mit dem Thema Graphen verbunden sind.

Bei den Lernzielen ist zu unterscheiden zwischen den Erziehungszielen und fachspezifischen allgemeinen Lernzielen.

Nach H.Bigalke lassen sich folgende allgemeine Erziehungsziele formulieren (Lit 11,S.39 ff.):

„1)Erziehung zur Fähigkeit,als unabhängiges Individuum mit sozialer Sensibilität in einer pluralistischen Gesellschaft zu leben.

2)Erziehung zur Kommunikationsbereitschaft und Kooperationsfähigkeit

3)Erziehung zur Mitbestimmung und Mitverantwortung in der Gesellschaft

4)Erziehung zur Innovationsbereitschaft,Kenntnis der Veränderungsprozesse der Gegenwart,Fähigkeit,selbständig zu lernen („Lernen lernen")

5)Erziehung zum Verständnis des Gleichgewichts zwischen ökologischen Systemen und der Technik

6)Erziehung zu wissenschaftlicher Intelligenz:Kenntnis und Verstehen der Prinzipien der Wissenschaft,ihrer grundlegenden Verfahren und ihrer Grenzen,Fähigkeit sich der Wissenschaft zu bedienen

7)Erziehung zu technologischer Intelligenz:Fähigkeiten,Technologien bereitzustellen und unter Kontrolle zu halten und mit der ständigen Erweiterung des Anwendungsbereichs von Technologien und ihrem Einfluß auf soziale Institutionen und Wertsysteme fertig zu werden.

8)Fähigkeiten,die vielfältigen Informationen des Medienangebots zu verarbeiten."

Nicht jedes Fach der Schule kann in gleicher Weise Beiträge zu jedem der Erziehungsziele leisten.

Die Fächer Mathematik und Informatik sind besonders dafür geeignet, Beiträge zu den Erziehungszielen 4 bis 8 zu erbringen.

Nach allgemeiner heutiger Auffassung ist es nicht möglich,aus den Erziehungszielen allgemeine und spezielle Lernziele eines Faches logisch stringent zu deduzieren (Lit 11, S. 41).Sehr wohl sollten aber allgemeine und spezielle Lernziele trotzdem unter „wechselseitiger Beeinflussung" (Lit 11, S. 41) an Hand der Erziehungsziele begründet werden.

Diese Begründung soll im folgenden einerseits durch Angabe der jeweiligen Erziehungsziele,die zum einzelnen allgemeinen mathematischen Lernziel gehören,als auch durch Angabe von speziellen diesen zugeordneten Lernzielen aus dem Gebiet der Graphentheorie durchgeführt werden.

Als allgemeine Lernziele des Mathematikunterichts,lassen sich nach H. Winter (Lit 11, ,S. 41 ff.) und F. Zech (Lit 64) die im folgenden aufgeführten Ziele nennen.

Zu jedem dieser 15 Ziele werden dann jeweils in Klammern Erziehungsziele genannt,die durch diese Lernziele angesprochen werden.

Zusätzlich wird,wie schon erwähnt,bei jedem allgemeinen Ziel außerdem ein oder mehrere dazu passende spezielle Lernziele,die zum Thema Graphentheorie im Unterricht gehören und die durch die dieser Arbeit beschriebenen Unterrichtsreihen unter Einsatz des Programms oder der Entwicklungsumgebung Knotengraph erreicht werden können,angegeben,um zu zeigen,dass Algorithmische Graphentheorie unter Einsatz objektorienterter Datenstukturen besonders geeignet ist,die Intentionen der genannten Lernziele zu verwirklichen.

(Natürlich ist diese Zusammenstellung nicht vollständig,sondern das jeweils genannte Lernziel kann oft nicht nur durch diese genannten Ziele sondern noch durch viele weitere anzustrebende Fähigkeiten bzw. durch die Wahl anderer Unterrichtsinhalte erfüllt werden.)

I)Lernziel: „Dialogfähigkeit und Dialogwillen (Sprachförderung und Kritikfähigkeit)“

(Erziehungsziele: 1,2)

Speziell Graphentheorie:Mathematische Aussagen über die Eigenschaften von Graphen zu formulieren,Lösungsalgorithmen verbal beschreiben können,die Schnittstellen von Programmodulen mit anderen Projektgruppen abstimmen zu können.

II)Lernziel:„Der Schüler muß lernen neue (mathematische) Situationen zu erzeugen und mit neuen Situationen fertig zu werden (Problemlöseverhalten).“

(Erziehungsziele: 1,4,5,6,7,8)

Speziell Graphentheorie:Neue Gesetzmäßigkeiten von Graphen zu erkennen,neue Graphen,die bestimmten Bedingungen genügen,zu erzeugen,Graphen miteinander vergleichen,für veränderte Graphen neu angepaßte Lösungsalgorithmen entwerfen.

III)Lernziel:„Der Schüler soll lernen,wie man eine inner- oder außermathematische Situation mit (mathematischen) Mitteln ordnen kann.“

(Erziehungsziele: 1,3,5,6,7,8)

Speziell Graphentheorie:Zu Problemen des Operations Research,zu Flussproblemen oder zu Problemen der Wahrscheinlichkeitsrechnung einen geeigneten Graphen erzeugen,eine Ordnungsrelation auf einem Graphen erzeugen,zu einem Transport-oder Hitchcockproblem bzw. zu einem allgemeinen Optimierungsproblem einen geeigneten Graphen entwerfen können.

IV)Lernziel: „Geistige Grundtechniken:“

(Erziehungsziele: 1,2,3,4,5,6,7,8)

a)Klassifizieren

Speziell Graphentheorie:Graphen nach ihren Eigenschaften beurteilen zu können,Objektklassenhierachien für Graphenalgorithmen in objektorientierten Programmiersprachen entwickeln (siehe nächstes Kapitel),Graphenstrukturen Bezeichner und Eigenschaften zuordnen:z.B. Liste, Baum, allgemeiner Graph, gerichtete oder ungerichtete Graphen,Flussgraph usw..

b)Ordnen

Speziell Graphentheorie:Verschiedene Pfade in Graphen nach ihrer Länge vergleichen und der Größe nach anordnen zu können,Probleme der Graphentheorie in Teilprobleme unterteilen und in eine logische Reihenfolge bringen,Datenstrukturen von Algorithmen ihrer Bedeutung nach anordnen,spezielle Graphendatenstrukturen der Reihenfolge ordnen:z.B.: Liste, Baum, allgemeiner Graph,Ordnen der Knoten gemäß einer Relationsbeziehung in Graphen,die als (antisymmetrische) Relation aufgefaßt werden kann.

c)Analogisieren

Speziell Graphentheorie:Verschiedene Graphen mit unterschiedlichem Aussehen aber mit gleicher Struktur als äquivalent erkennen,gleiche Datenstrukturen erkennen und durch Objektklassen beschreiben,einen Graphen als mathematisches Modell erkennen und beschreiben.

d)Vergleichen

Speziell Graphentheorie:Zwei oder mehrere Graphen bezüglich ihrer Knoten- oder Kantenzahlen miteinander vergleichen,Graphen von unterschiedlicher Gestalt aber mit mit gleichen Eigenschaften erkennen,gleiche Lösungsalgorithmen bzw. Lösungsverfahren für unterschiedliche Graphen entwikkeln,Graphenalgorithmen bezüglich ihrer Effizienz vergleichen.

e)Generalisieren

Speziell Graphentheorie:In einem Graphen,bei dem alle Knoten geraden Grades sind,erkennen,dass er zur Klasse der Graphen gehört,die eine Eulerlinie besitzen,erkennen können,dass ein spezieller Graph beispielsweise mit zwei Farben färbbar ist,allgemein:von den vielen Eigenschaften eines konkret vorgegebenen Graphen auf wenige viele Graphen verbindende Eigenschaften abstrahieren zu können,allgemeine Objektklassen für objektorientierte Programmiersprachen zur Lösung von Graphenalgorithmen entwerfen,die Eulerbeziehung in Bäumen auf allgemeine ebene Graphen verallgemeinern.

f)Konkretisieren

Speziell Graphentheorie: Bei einem vorgegebenen Graph die konkrete Eigenschaften des paaren Graphen erkennen,erkennen,dass ein Graph das Modell eines endlichen Automaten ist,erkennen,dass die Lösung eines speziellen Graphenproblems mit Hilfe einer bestimmten schon vorgegeben Objektdatenstruktur möglich ist,z.B. den Algorithmus „Bestimme minimalen Kostenfluss" zu einem Algorithmus zur Lösung des Transportproblems spezialisieren.

g)Formalisieren

Speziell Graphentheorie:Der Schüler soll Graphenalgorithmen als Flussdiagramm oder in einer Programmiersprache beschreiben können,Beweise zu Graphenalgorithmen bzw. Graphenproblemen sollen in mathematischer Schreibweise verstanden werden,z.B. der Beweis zur Korrektheit des Algorithmus von Djkstra,einen endlichen Automaten durch einen Graph beschreiben,Graphen als mathematische Modelle verwenden.

V)Lernziel:„Der Schüler besitzt logisches und heuristisches Denkvermögen."

(Erziehungsziele: 1,4,5,6,7,8)

Speziell Graphentheorie:Aus der Planarität eines Graphen auf die Gültigkeit der Eulerschen Formel für Knoten,Kanten und Gebiete schließen,einen Algorithmus zur Erzeugen einer globalen Ordnung auf einem Graphen mit partieller Ordnung entwickeln,eine Lösung des chinesischen Briefträgerproblems durch Probieren finden,ein maximales Matching in einem Graph suchen.

VI)Lernziel: „Der Schüler besitzt Anschauungsvermögen."

(Erziehungsziele: 5,6,7,8)

Speziell Graphentheorie:An einem Graphen spezielle mathematische Eigenschaften ablesen können,strukturgleiche Graphen erkennen können,ein reales Problem in der Form eines Graphen als Modell darstellen können:z.B. das Personalzuteilungsproblem,erkennen können,ob ein Graph plättbar ist.

VII)Lernziel:„Selbständigkeit und Selbsttätigkeit"

(Erziehungsziele: 1,2,3,4,5,6,7,8)

Speziell Graphentheorie:Ohne fremde Hilfe Graphenalgorithmen (manuell,d.h. zeichnerisch oder verbal) ausführen zu können:z.B. den Algorithmus von Busacker und Gowen,ohne fremde Hilfe,verbale Algorithmen zu Problemen der Graphentheorie zu finden,ohne fremde Hilfe Graphenalgorithmen in einer Programmiersprache codieren können,ohne fremde Hilfe Eigenschaften von Graphen beweisen können:z.B. den Zusammenhang zwischen der Anzahl der Kanten und Knoten eines Baums.

VIII)Lernziel: „Beherrschung von sog. Kulturtechniken"

(Erziehungsziele: 4,6,7,8)

Speziell Graphentheorie:Graphen zeichnerisch darstellen können,Computer bedienen können,um ein Programm zur Graphentheorie zu starten.

Spezielle mathematische allgemeine Lernziele:

IX) „Die Fähigkeit Umweltsituationen zu mathematisieren"

(Erziehungsziele: 4,5,6,7,8)

Speziell Graphentheorie:Zu einem vorgegebenen Problem,z.B. dem Problem des kürzesten Weges einen geeigneten Graphen konstruieren,Probleme des Operations Research wie z.B. das minimale Kosten-Problem,das Transportproblem,das Personalzuteilungsproblem,das allgemeine Optimierungsproblem mit Hilfe von Graphen lösen können,optimale Zeitpläne für den Ablauf der Tätigkeiten eines Projekts (z.B. von Bauvorhaben) entwerfen können.

X)Lernziel:„Die Fähigkeit Umwelterscheinungen mathematischer Art zu verstehen (und kritisch zu beurteilen)."

(Erziehungsziel: 1,4,5,6,7,8)

Speziell Graphentheorie: z.B. mit Hilfe von Markovgraphen die Gewinnchanchen von Glücksspielen beurteilen können,das Verkehrsproblem in Städten mit Hilfe von Graphen (z.B. Maximalflussproblem) analysieren.

XI)Lernziel: „Die Fähigkeit,Möglichkeiten und Grenzen der Mathematik zu sehen"

(Erziehungsziele: 5,6,7,8)

Speziell Graphentheorie:Die Lösungsmöglichkeit von Optimierungsproblemen mit Graphen erkennen,erkennen,dass das Traveling-Salesman-Problem für große Knotenzahlen immer schwerer lösbar wird,da es NP-vollständig ist,erkennen,dass nicht alle Aspekte der realen Wirklichkeit durch die Modellierung der Realität mit Hilfe von Graphen erfasst werden können.

XII)Lernziel: „Die Freude an der ästhetischen und spielerischen Seite der Mathematik"

(Erziehungsziele: 6)

Speziell Graphentheorie:Denksportaufgaben als Graphen darstellen und mit Graphenalgorithmen lösen,Spiele bzw. Spielstrategien mittels Graphen ausführen (z.B. Umfüllaufgaben),Symmetrien in der Struktur von Graphen erkennen.

Spezielle Lernziele der Informatik:

(nach Gunzenhäuser,Lit 2, S. 169)

XIII)Lernziel:Vertrautheit mit Algorithmen und ihrer Programmierung

(Erziehungsziele: 4,5,6,7,8)

Speziell Graphentheorie:Graphenalgorithmen in einer Programmiersprache entwerfen können,objektorientierte Datenstrukturen bei Graphenalgorithmen einsetzen können.

XIV)Lernziel:Einblick in Aufbau und Wirkungsweise eines Computers

(Erziehungsziele: 4,5,6,7,8)

Speziell Graphentheorie:Modell des endlichen Automaten mit Hilfe von Graphen darstellen und simulieren können,einen Computer bedienen können,um ein

Programm zur Graphentheorie ausführen zu lassen,Datenstrukturen im Hinblick auf ihre effiziente Speicherung im Computer auswählen können.

XV)Lernziel:Kenntnisse der Anwendungen und Auswirkungen der Informatik

(Erziehungsziele: 3,4,5,6,7,8)

Speziell Graphentheorie:Mittels Graphen Modelle der Wirklichkeit simulieren und über die Ergebnisse dieser Algorithmen reflektieren,z.B. Verkehrsprobleme,Probleme des kürzesten Weges oder der kürzesten Rundreise,Optimierung von Kosten,Zeitpläne für den Ablauf eines Projektes.

Die letztgenannten Lernziele stimmen im wesentlichen überein mit den detailliert ausgeführten Lernzielen der Einheitlichen Prüfungsanforderungen der Bund-Länder-Kommission für Forschungsförderung und Bildungsplanung zur „vertieften informationstechnischen Bildung" aus dem Jahre 1989:

(vgl. Lit 2,S.170)

1)Kennen von „Methoden der Informatik (a) bei der Problemanalyse und -spezifikation,(b) beim Entwurf und zur Darstellung von Lösungsverfahren,(c) bei der Codierung von Lösungsalgorithmen,(d) beim Testen und bei der Korrektur von Programmen,(e) bei der Beurteilung und Optimierung der Problemlösung,(f) bei der Dokumentation"

2)Kennen von „Funktionsprinzipen von Hard-und Softwaresystemen (a) im Bereich der Anwendersoftware,(b) im Bereich der Programmiersprachen,(c) bei Betriebssoftware,(d) bei Rechnerkonfigurationen und Rechnermodellen,(e) bei den theoretischen Grundlagen"

3)Kennen von „Anwendungen von Hard- und Softwaresystemen und deren gesellschaftlichen Auswirkungen (a) in typischen Anwendungsgebieten,(b) bei der Beurteilung der Mensch-Maschine-Schnittstelle,(c) bei der Kenntnis und Beurteilung der Grenzen und Möglichkeiten,Chancen und Risiken des Einsatzes der Informations- und Kommunikationstechniken"

Nach H.Boer,R.Krane und H.Wiggermann (Lit 9,Anwendungen der Graphentheorie im Hinblick auf die Konstruktion von Unterrichtseinheiten für den Mathematikunterricht) sollte die Auswahl der Unterrichtsinhalte für den Mathematikunterricht dagegen weniger nach allgemeinen Lernzielen als nach der Schülerrelevanz,d.h. danach,ob sie für die tägliche Erfahrung der Schüler von Bedeutung sind,solcher Themen erfolgen:

„Im Unterricht soll es um die Schülerrelevanten Probleme gehen.Und:Die Schüler sollen lernen,gerechtfertigt zu handeln.Damit ist gefordert,Unterrichtshandeln zu rechtfertigen (Mittel müssen dem Ziel angemessen sein!Man lernt,was man tut, und nur das!)Die Aufforderung an die Schüler,sich mit einem Unterrichtsgegenstand auseinanderzusetzen,muß Schülerrelativ einsehbar (kontrollierbar) begründet werden.Also: Im Mathematikunterricht sollen Problemsituationen behandelt werden.Die Forderung nach Emanzipation erfordert zusätzlich:

1.Die gestellten Probleme sollen relevante Probleme für Schüler sein.

2.Die gestellten Probleme sollen als relevant einsehbar gemacht und eingesehen werden.

3.Schüler sollen befähigt werden,die als relevant eingesehenen Probleme (damit selbstgestellten Probleme)solidarisch zu lösen.

Mathematikunterricht hat mit Problemen zu beginnen und wieder im Gegenstandsbereich des Problems zu enden.Erster Rechtfertigungsgrund ist der Aufweis,daß die Behandlung des Problems sinnvoll für die Schüler ist."

(nach Lit 9, S. 48)

Als Forderungen für die Stoffauswahl ergeben sich (Lit 9) dazu die folgenden Kriterien:

1.Die Probleme sollen an den Erfahrungshorizont der Schüler anschließen.

2.Die Aufgabenstellungen sollen gesellschaftliche Probleme sein.

3.Das Lösen der Probleme soll die Möglichkeit emanzipatorischen Handeln unterstützen.

4.Die Problemlösung soll direkte Handlungsmöglichkeiten des Schülers initiieren.

Einen Mathematikunterricht,der diese Ziele verfolgt,heißt nach Boer,Krane und Wiggermann „Problemorientierter Mathematikunterricht in emanzipatorischer Absicht (PROMEA)".

Als Beispiele für Anwendungen der Graphentheorie,die für Schüler von Relevanz sein können, und mit denen die oben genannten Ziele erreichbar sind,werden genannt:

A)Graphen als Darstellungsmittel in der täglichen Umgebung:

Beispiele:Graphen-Diagramme zur Darstellung des Bildungsweges unter Besuch von verschiedenen Schultypen Schulen und Universitäten bis zum Schul- bzw. Universitätsabschluß,Graphen-Diagramme zur Darstellung der Hierachie von verschiedenen Institutionen des Staates,Graphen-Diagramme zur Veranschaulichung von wirtschaftlichen und Kapitalverpflechtungen sowie von Massenmedien

B)Graphen als Darstellungsmittel in Wissenschaftsgebieten:

Darstellung von Gleichungssystemen durch Graphen,Darstellung von Markovketten und Spielbäumen durch Graphen,Darstellung von Beispielen aus den Naturwissenschaften wie Strukturformeln in der Chemie und Vererbungsbeziehungen in der Biologie sowie Soziogramme in Form von Graphen in den Sozialwissenschaften als auch die Darstellung von Grammatiken in den Sprachwissenschaften. (Zu ergänzen wäre hier die Darstellung von Grammatiken von Programmiersprachen in der Informatik durch Bäume.)

C)Graphen als Darstellungsmöglichkeit in Wirtschaft und Verkehr:

Verkehrstechnik und Planung,Beziehungen in der Nachrichten- und Elektrotechnik sowie Elektronik (z.B. Schaltplatinenlayout),Flussdigramme in der Informatik

D)Probleme des Operations Research

Optimierungsprobleme, Netzplantechnik, Versorgungsprobleme, Transportprobleme, Zuordnungprobleme usw.

Offenbar sind die von Boer,Krane und Wiggermann allgemeinen Zielen doch schon in den oben genannten Lernzielen,insbesondere in dem speziellen Lernziel

„Der Schüler soll lernen,wie man eine inner- oder außermathematische Situation mit (mathematischen) Mitteln ordnen kann."

oder in den allgemeinen Lernzielen

„Erziehung zu technologischer Intelligenz: Fähigkeiten, Technologien bereitzustellen und unter Kontrolle zu halten,und mit der ständigen Erweiterung des Anwendungsbereichs von Technologien und ihrem Einfluß auf soziale Institutionen und Wertsysteme fertig zu werden.

Fähigkeiten,die vielfältigen Informationen des Medienangebots zu verarbeiten.“

enthalten.

Besonders herausgestellt wird von Boer,Krane und Wiggermann nur,dass die Probleme dem unmittelbaren Erfahrungsbereich des Schülers entnommen werden sollten.

Eine große Zahl der von Boer,Krane und Wiggermann genannten Anwendungen insbesondere die in den Punkten B bis D genannten sind als konkrete Algorithmen im Programm Knotengraph implementiert bzw. es wird dargestellt,wie die Implementation objektorientiert durch Schüler erfolgen kann (vergleiche Abschnitt C).Punkt A dagegen beinhaltet mehr den Aspekt der Modellierung eines vorgegebene Problems durch einen Graphen,der in vielen Fällen mit der Erzeugung einer Ordnung verbunden ist.Dieser Gesichtspunkt spielt bei der Erzeugung eines geeigneten Graphen bei allen Anwendungen von Knotengraph eine wichtige Rolle,und speziell der Aspekt des Ordnens wird besonders in dem Kapitel des Abschnitts C Graph als Relation behandelt.

Das Programm bzw. die Entwicklungsumgebung Knotengraph kann also in besonderer Weise dazu beitragen,die eben genannten speziellen als auch die bisher diskutierten allgemeinen Lernziele,denen durch die obige Zusammenstellung im bisherigen Teil dieses Kapitels jeweils konkrete Ziele der in dieser Arbeit beschriebenen Unterrichtssequenzen zugeordnet wurden,zu erreichen.

Zur Bewertung von Lernzielen liegt seit 1972 die Taxonomie von Bloom vor,die drei Lernzielbereiche umfaßt:

A)der kognitive Bereich

B)der affektive Bereich

C)der psychomotorische Bereich

Der Bereich A) umfaßt die Lernziele,die Wissen und Denken betreffen,Bereich B) die Lernziele,die Einstellungen,Interessen und Wertschätzungen umfassen und Bereich C) handelt von den manuellen und motorischen Fähigkeiten.

Dadurch dass die Beschäftigung mit Themen der Graphentheorie zum selbstständigen Problemlösen anregt,weil viele Probleme leicht ohne Vorwissen verständlich sind,und auch viele Probleme einen Anwendungsbezug haben sowie durch die Darstellung als Graph sehr übersichtlich zu formulieren sind,lassen sich Lernziele aus dem Bereichs B) wie Wecken von allgemeinem Interesse für die Untersuchung von mathematischen Fragestellungen,Verstehen der Bedeutung der Mathematik für unsere Gesellschaft sowie Nachdenken über die Schönheit der Mathematik (besser Eleganz von mathematischen Lösungsverfahren) z.B. durch Ausnutzen von symmetrischen oder besonders prägnanten,weil aus nichtmathematischen Bereichen z.B. aus der Kunst oder von Spielen bekannten Graphenstrukturen erreichen oder aber ein Lösungsweg ist durch die Graphenstruktur unmittelbar trivial einsichtig.

Auch psychomotorische Lernziele C) können angesprochen werden,wie z.B. das Erlangen von Fingerfertigkeiten beim Zeichnen eines Graphen, oder bei der Bedienung der Tastatur eines Computers,während der Eingabe oder der Bedienung eines Programms,das einen Algorithmus der Graphentheorie beinhaltet.

Die Hauptlernziele des Mathematik- und Informatikunterrichts liegen jedoch im Bereich A).Er wird wiederum unterteilt in sechs Kategorien:

Wissen, Verstehen, Anwendung, Analyse, Synthese, Bewertung

Dabei baut jede Kategorie auf der vorigen auf, und jede nachfolgende Kategorie benötigt die Vertrautheit mit der vorigen Kategorie.

Die Vorschriften beim Einreichen von Abiturvorschlägen im Fach Mathematik in NRW sehen u.a. vor,die Aufgabenteile von Abituraufgaben gemäß ihres Schwierigkeitsgrades in diese Kategorien einzuordnen.Dabei werden die Kategorien zu drei Bereichen zusammengefaßt.

Bereich I umfaßt die Kategorien Wissen und Verstehen,Bereich II die Kategorien Anwendung und Analyse und Bereich III die Kategorien Synthese und Bewertung.Der letzte Bereich wird auch als kreatives Denken bezeichnet.Verlangt wird,dass das Verhältnis der Aufgabenteile aller Aufgaben zu den drei Bereiche bei einem Abiturvorschlag im Verhältnis von ca. 30 zu 60 zu 10 berücksichtigt wird,d.h. das Hauptgewicht liegt auf dem Bereich II:Anwendung und Analyse.

Die Klassifizierung der Lernziele,die mit dem Unterricht zum Thema Graphen verbunden sind,spricht alle sechs Kategorien (bzw. die Kategorien I bis III) der Bloomschen Einteilung an.

Dafür werden im folgenden einige Beispiele gegeben:

Wissen:

allgemein:

Wissen und Kenntnis von Begriffen, Sätzen, Algorithmen, Fachbegriffen, Definitionen, Formeln, Anweisungen einer Programmiersprache, Datenstrukturen

Graphen:

Die Eulersche Beziehung zwischen Knoten, Kanten und Gebiete nennen können, definieren können,was ein vollständiger, ein paarer (oder bipartiter),ein schlichter oder ein kreisfreier,ein gerichteter oder ein zusammenhängender Graph ist usw.

Verstehen:

allgemein:

Verständnis und eigenständige Beschreibung von Fachsymbolen,Bezeichnungen, Algorithmen,Beweisen,Datenstrukturen

Graphen:

Zu einem bipartiten,einem vollständigem,einem schlichtem oder einem kreisfreien Graphen jeweils ein Beispiel zeichnen können (oder mit dem Programm Knotengraph erzeugen),die einzelnen Schritte eines Graphenalgorithmus erläutern können,die vorgegebene Datenstruktur des Graphen erläutern können

Anwendung:

allgemein:

Gebrauch und Verwendung von Fachsymbolen,Bezeichnungen, Algorithmen, Beweisen, Datenstrukturen

Graphen:

Einen Algorithmus der Graphentheorie auf einen vorgegeben Graphen mittels immer neuer Zeichnungen der jeweiligen Veränderungen der Graphen zeichnerisch durchführen können, ein Programm zu einem vorgegebenen Algorithmus schreiben können,einen Beweis nach einem schon bekannten Muster auf ein verwandtes Graphenproblem übertragen können (Transfer)

Analyse:

allgemein:

Fähigkeit Aufgaben einer bestimmten Art lösen,Algorithmen eines bestimmten Typs anzuwenden,ein bestimmtes Beweisverfahren durchführen,Datentypen einer bestimmten Klasse definieren

Graphen:

Einen Beweis nach einem schon bekannten Muster auf ein verwandtes Graphenproblem übertragen können (Transfer),gleiche Strukturen bei Graphen feststellen zu können, einen Algorithmus an ein neues Problem anpassen können

Synthese:

allgemein:

Fähigkeit neue Beweise aus vorgegebenen Voraussetzungen zu finden,neue Algorithmen entwerfen,neue Datenstrukturen entwickeln,neue mathematische Eigenschaften entdecken

Graphentheorie:

Einen eigenen, neuen Beweis für ein Problem der Graphentheorie finden,einen neuen Algorithmus für ein vorhandenes Problem der Graphentheorie entdecken,neue Eigenschaften von Graphen ermitteln

Bewertung:

allgemein:

Lösungsverfahren,Beweise,Algorithmen und Datenstrukturen auf Richtigkeit und Effizienz prüfen und miteinander vergleichen können

Graphen:

Verschiedene Beweisverfahren der Graphentheorie auf Richtigkeit prüfen und vergleichen,verschiedene Graphenalgorithmen auf Effizient überprüfen,verschiedene Datenstrukturen für Graphen miteinander vergleichen

Welche der bisher genannten Lernziele und Lernzielkategorien lassen sich nun durch Lerninhalte der Graphentheorie besonders gut erreichen?

Nach H.Bialke (Lit 23,S. 5 ff.) läßt sich die Bedeutung der Graphentheorie für den Mathematikunterricht durch Angebote aus den folgenden vier Bereichen charakterisieren:

1)"Veranschaulichung von (eventuell) komplizierten Zusammenhängen"

Bereits gelöste Problem können mit Hilfe von Graphen so veranschaulicht werden, dass sie für den Schüler einsichtiger werden. Zusammenhänge zwischen verschiedenen Mathematikobjekten werden dargestellt und regen zu weiteren Fragestellungen an.

Beispiele sind Veranschaulichung von Beziehungen der Gruppentheorie (Gruppengraphen),Verbandstheorie (Hase-Diagramme), Relationstheorie (Pfeildiagramme), Kombinatorik (Graphen mit kombinatorischen Strukturen), Wahrscheinlichkeitsrechnung (Wahrscheinlichkeitsbäume, Markovketten), Mengenlehre (Punktmengen und ihre Beziehungen als Graphen), Darstellung von Algorithmen (Flussdiagramme),Chemie (Strukturformeln), Biologie (Stammbäume), Technik (Schaltpläne), Psychologie (Soziogramme), Sozialwissenschaften (Kommunikationsmodelle), Sprachwissenschaften (Strukturdiagramme).

Das Veranschaulichen eines Problems in Form eines Graphen bedeutet Erkennen seiner grundsätzlichen Struktur und Übertragung der Beziehungen in Kanten-Knoten-Relationen.Dies erfordert Analyse der Aufgabenstellung und Transfer (Anwendung) auf das Graphenmodell.

2)"Modelle für Projekte, die eine eigene algorithmische Lösung erfordern"

Viele mathematische Probleme lasen sich erst dadurch lösen,dass sie in Form eines Graphen dargestellt werden und ein entsprechender Graphenalgorithmus entwickelt wird.

Beispiele hierfür sind Projektplanungen der Netzplantechnik, Transportproblem bei Netzwerkflüssen, Routing- bzw. Rundreiseprobleme, Labyrinthprobleme (Verkehrsplanung), Probleme, die mit Hilfe von (endlichen) Automaten gelöst werden können (Ampelsteuerung) sowie Probleme der Unterhaltungsmathematik (Denksportaufgaben: Überfahrten usw.).

Zusätzlich zu den Lernzielkategorien, die bei 1) genannt wurden, ist hier noch die Synthese in Form der Erstellung eines geeigneten Graphenalgorithmus erforderlich.

3)"Lösungen von Problemen mit Hilfe graphentheoretischer Sätze"

Bei diesem Bereich von Problemen lassen sich die Lösungen auf schon bekannte Sätze oder Algorithmen der Graphentheorie zurückführen.

Beispiele dafür sind das Anwenden von Sätzen über planare Graphen auf das Layout von elektrischen Schaltungen,das Verwenden der Algorithmen für Euler-und Hamiltonproblem auf Rundreisen,das Benutzen von Sätzen über Anzahl und Eigenschaften von Bäumen für das Aufstellen von chemische Strukturformel,das Anwenden von Sätzen (z.B. Heiratssatz) zur Ermittlung des maximalen Matchings eines Graphen,die Ermittlung von Schnitten sowie Flüssen in Netzwerken,die Verwendung von Sätzen zum Färbungsproblem für das Erstellen von Landkarten.

Die Haupttätigkeit liegt hier in dem Übertragen des Problems in eine Graphenstruktur (Tranfer, d.h. Anwendung) und dem Erkennen, welche Sätze oder Algorithmen der Graphentheorie angewendet werden sollen.(Anwendung)

4)"Leicht verständliche Probleme mit beliebig hohem Schwierigkeitsgrad"

Unter dieser Art von Problemen sind solche Probleme zu verstehen,die sich leicht als Graphen darstellen lassen und deren Beweise oder Lösungsalgorithmen an Hand der Graphenstruktur unmittelbar einsichtig werden.

Beispiele hierfür sind Sätze über den numerischen Zusammenhang zwischen Knoten und Kanten in speziellen Graphen,Sätze über Wege und Kreise, einfache Sätze über Knoten- und Kantenfärbungen (z.B. in paaren Graphen).

Da die Beweise und Algorithmen unmittelbar evident sind,sind diese Lernziel unter der Kategorie Anwendungen einzuordnen.

Die Lernziele,die mit den vier oben genannten Bereichen verbunden sind, liegen durchweg im Bereich der Kategorien Anwendung und Analyse (Bereich II) und reichen zum geringeren Teil in den dritten Bereich des kreativen Denkens (Synthese) hinein. Es handelt sich also gerade hauptsächlich um Lernziele aus dem Bereich II, die bei der Erstellung von Abituraufgaben und Klausuraufgaben besonders gefordert werden.Da die Lernbereiche der Bloomschen Kategorie aufeinander aufbauen, sind Lernziele aus dem Bereich I zum Erreichen der Lernziel des Bereiches II automatisch enthalten.(Z.B. muß man wissen, was ein Graph ist, bevor man einen Lösungsalgorithmus damit erstellen kann.)

Es zeigt sich also,dass Aufgaben zu Problemen aus dem Bereich Graphentheorie den Anforderungen von Klausur- und Abituraufgaben vollauf genügen und dass ein Unterricht,der auf die Erreichung der damit verbundenen Lernziele gerichtet ist,geeignet ist,den auf konventionelle Lernziele zielenden Mathematikunterricht zu ersetzen.

Um die Erreichung von Lernzielen genau überprüfen zu können ist,die Operationalisierung von Lernzielen notwendig.Darunter versteht man eine präzisierte Angaben von Lernzielen, die das Endverhalten des Lernenden nach Erreichen des Lernzieles eindeutig beschreibt,sowie Voraussetzungen für die Erreichung des Lernzieles angibt.

Die Präzisieren ist so genau vorzunehmen,dass die Beschreibung konkrete Tätigkeiten angibt,die der Lernende nach Erreichen des Ziels ausführen kann. Am besten läßt sich das Erreichen der Lernziele mittels der Angabe eines konkreten Testverfahrens überprüfbar machen.

Hilfen zum Erreichen des Zieles müssen genauestens angegeben werden.Außerdem soll unter Umständen eine Zeitangabe erfolgen,wie lange es dauert, bis das Lernziel erreicht werden kann.

Im folgenden sollen einige ausgewählte Lernziele der Graphentheorie,soweit sie für Unterrichtsreihen mit dem Programm Knotengraph von Bedeutung sind, in operationalisierter Form angegeben werden.Eine vollständige Angabe aller Lernziele ist natürlich wegen des damit verbundenen Aufwands auf Grund des großen Umfangs der Möglichkeiten des Programm Knotengraphs unmöglich.

Die Operationalisierung von Lernzielen hängt eng mit dem Entwurf von Klausuraufgaben bzw. von mündlichen Prüfungsaufgaben zu den Lernzielen einer Unterrichtsreihe zum Thema Algorithmische Graphentheorie zusammen.Auch dazu geben die folgenden Beschreibungen Beispiele.

Operationalisierung von Lernzielen der Behandlung von Graphen im Unterricht in Auswahl:

Die Schüler sollen die Definition eines Graphen nennen können.

Die Schüler sollen Eigenschaften von Graphen nennen können, und jeweils für jede Eigenschaft einen zugehörigen Beispielgraphen zeichnen oder mit dem Programm Knotengraph erzeugen können,z.B. paarer Graph, vollständiger Graph, zusammenhängender Graph, Graph mit isolierten Knoten,ungerichteter Graph usw..

Die Schüler sollen von einem Startknoten eines vorgegebenen Graphen aus alle tiefen Baumpfade durch Nennen der Knoten- bzw. Kantenfolgen angeben können und alle dabei erreichten Zielknoten aufschreiben können.

Die Schüler sollen für einen vorgegebenen Graphen bestimmen können,ob er eine geschlossene Eulerlinie enthält,indem sie alle Knotengrade nennen und anschließend eine Aussage darüber treffen können,ob sich unter den Knotengraden eine ungerade Zahl befunden hat oder nicht.

Die Schüler sollen zu zwei vorgegebenen Knoten eines Graphen alle Pfade bzw. den Pfad kleinster Länge in Form einer Folge der Kanten des Pfades hinschreiben können.

Die Schüler sollen den Algorithmus von Ford-Fulkerson zur Bestimmung des maximalen Flusses auf einem vorgegebenen Graphen ausführen können,indem Sie jeweils in mehrerer Zeichnungen alle veränderten Graphen aufzeichnen.

Die Schüler sollen einen Algorithmus,der bestimmt, ob sich ein vorgegebener Graph mit vier Farben färben läßt,als Verbalalgorithmus hinschreiben können.

Die Schüler sollen den Algorithmus Suche erweiternden Weg (und färbe die Kanten um) zur Bestimmung eines maximalen Matchings nach Vorgabe als Verbalalgorithmus in der Programmiersprache Delphi einschließlich aller benötigten Methoden aufschreiben können.

Die Schüler sollen eine objektorientierte Datenstruktur für einen Graphen aufschreiben können.

Die Schüler sollen den Programmcode für den Algorithmus Bestimmung eines Minimalgerüst in der Delphi-Entwicklungsumgebung eingeben können,das Programm debuggen und zum Laufen bringen.

Die Schüler sollen das Programm Knotengraph bedienen können,d.h. einen Graphen auf der Zeichenoberfläche erzeugen und z.B. den Algorithmus „Bestimme alle Hamiltonkreise" starten können.

Die Schüler sollen von dem Programm Knotengraph angezeigte Graphen oder per Zeichnung vorgegebene Graphen interpretieren können,d.h. z.B. beim Lösen eines Gleichungssystem nach dem Algorithmus von Mason zu einem vorgegebenen Graphen das zugehörige Gleichungssystem angeben können.

Die Schüler sollen verschiedene Algorithmen miteinander vergleichen können, d.h. z.B. zu den beiden im Programm Knotengraph enthaltenden Algorithmen zur Bestimmung der Übergangswahrscheinlichkeiten bei Markovketten (Markovketten (abs) und Graph reduzieren) die wesentlichen Schritte nennen können.Zu einem vorgegebenen Graphen soll ermittelt werden,welcher Algorithmus die größere Rechengenauigkeit liefern kann.

Die Schüler sollen praktische Anwendungsbeispiele aus dem Bereich Operations Research nennen können,die mit Graphenalgorithmen lösbar sind, und zu den Beispielen einen geeigneten Graphen konstruieren können,z.B.zum Minimalkostenproblem oder zum Transportproblem.

Die Schüler sollen angeben können,dass die Suche einer Lösung beim Traveling-Salesman-Problem mit einer Knotenzahl von mehr als 20 Knoten zu unvertretbar langen Rechenzeiten führt.Als Grund dafür soll die NP-Eigenschaft genannt werden,und definiert werden können,was NP-vollständig heißt.

Im Kapitel B III „Konkrete methodisch-didaktische Bemerkungen zur Durchführung von Unterrichtsreihen zum Thema algorithmische Graphentheorie" werden konkrete Aufgaben angegeben und erläutert, die im Zusammenhang mit Unterrichtsreihen über Algorithmische Graphentheorie als Klausuraufgaben oder als Abituraufgaben gestellt werden können.

Nach der Auswahl, Begründung, Klassifikation und Operationalisierung von Lernzielen die mit Hilfe der Behandlung von Algorithmischer Graphentheorie im Unterricht erreicht werden können,sollen jetzt im zweiten Teil dieses Kapitels verschiedene Unterrichtsmethoden zum Erreichen dieser Ziele diskutiert werden.

Bezüglich der Methodik des Unterrichts lassen sich für die Fächer Mathematik und Informatik sechs besondere Unterrichtsformen bzw. Unterrichtskonzeptionen unterscheiden,für die jeweils geprüft werden soll,inwieweit sie sich zur Behandlung des Themas Graphen im Unterricht eignen.

J. Claus (Lit 11,S. 82 und S. 150 ff.) zählt dazu folgende Unterrichtskonzeptionen auf:

I)Operationalisierter Unterricht

II)Wissenschaftsorientierter Unterricht

III)Anwendungsorientierter Unterricht

IV)Problemorientierter Unterricht

V)Genetischer Unterricht nach Wagenschein und Wittenberg

VI)Projektorientierter Unterricht

Diskussion der einzelnen Unterrichtsmethoden im Hinblick auf ihre Bedeutung für die Graphentheorie im Schulunterricht:

I)Operationalisierter Unterricht:

Nach Piaget ist Denken verinnerlichtes Handeln.Die zum Handeln notwendigen Operationen sind demnach nicht nur reale ausgeführte Tätigkeiten sondern auch Umgang mit abstrakten Objekten wie Bezeichnern, Variablen, Formeln, Algorithmen usw.. Nach dem Prinzip des aktiven Lernens sollten Anweisungen und Hilfestellungen des Lehrers ein Handeln des Schülers bewirken,dass den Schüler zu intensiven Auseinandersetzung in Form von (verinnerlichten) Aktivitäten mit dem zu lernenden Objekt veranlaßt.

Der Schüler soll dabei die von dem Objekt gebildete Gesamtstruktur bzw. die Beziehung der das Objekt enthaltenden Einzelobjekte zueinander erkennen können.

Im Einzelnen soll der Schüler erkennen,

1."welche Eigenschaften und Beziehungen den Objekten durch Konstruktion aufgeprägt ist"

2."welche Operationen an ihnen und mit ihnen ausführbar sind,und wie sie miteinander wechselwirken"

3."welche Wirkungen Operationen auf Eigenschaften und Beziehungen der Objekte haben"

(nach E. Wittmann in Lit 11)

Operationen mit Graphen ausführen,heißt Algorithmen auf sie anzuwenden. Das Handeln mit Algorithmen kann entweder manuell ausgeführt werden,indem die entsprechenden Graphen in der durch den Algorithmus veränderten Formen immer wieder bis zum Lösungsgraphen neu gezeichnet werden,oder aber indem in den Graphen Zahlen eingetragen und wieder gelöscht (z.B. radiert) werden,die den Fortlauf des Algorithmus kennzeichnen.

Andererseits besteht ein verinnerlichtes Handeln darin,die Schritte des Algorithmus vorzudenken und in Form von Programmzeilen oder als größere Einheiten in Form von Programmblöcken aufzuschreiben,um dann den Algorithmus durch Mausklick auf einmal für einen vorgegebenen Graphen ablaufen lassen.

Hier wird besonders deutlich, dass die Erstellung eines Programms im Mathematik-oder Informatikunterricht von der Entwurfsphase bis zur Codierung im Grunde eine operationalisierte Unterrichtsform ist.

Das Stichwort Objekt in den obigen Kriterien 1 bis 3 führt sofort zu einer besonderen Programmierungsmethode,nämlich der Verwendung von Objekten als Kombination bzw. Kapselung von Daten und Methoden zur Erstellung von Graphenalgorithmen.

Die Beziehungen der Objekte untereinander wird durch ihre (vom Programmierer konstruierte und modellierte) Vererbungsstruktur vorgegebenen.(vgl. 1)

Die Methoden der Objekte geben an,"welche Operationen an ihnen und mit ihnen ausführbar sind und wie sie miteinander wechselwirken" (vgl. 2).

Durch Verwendung von virtuellen Methoden können Operationen auf die Eigenschaften der Objekte verschiedene Wirkungen haben (vgl. 3).

Die Methoden können aufgefaßt werden als Nachrichten,d.h. Handlungen, die die verschiedenen Objekte miteinander austauschen,worauf eine je nach Objekt verschiedene Reaktion gezeigt wird.

Somit fällt das objektorientierte Programmieren von Problemen der Graphentheorie direkt unter das Stichwort operationalisierter Unterricht.Wegen der Wichtigkeit des Einsatzes von Objekten im Programm Knotengraph wird der didaktischen Analyse der Programmierung mit Objekten im nächsten Kapitel A II ein eigener Abschnitt gewidmet.

II)Wissenschaftsorientierter Unterricht:

Der wissenschaftsorientierte Unterricht geht davon aus,den zu behandelnden Unterrichtsstoff in seinem Umfang zu beschränken (Vermeidung von Stofffülle) und sich stattdessen mit einigen allgemeinen Ideen von überragender Bedeutung eines Faches zu beschäftigen (z. B. dem Funktionsbegriff in der Mathematik) und diese nach allen Seiten hin auszuleuchten. Dabei soll der Unterrichtende eine Forschungshaltung einnehmen können.

Universelle Leitideen können nach A. Schreiber (Lit 11, S. 156) sein:

„1)Algorithmus (Rechen- oder Entscheidungsverfahren, Berechenbarkeit, Programmierung, mathematische Modellbildung)

2)Approximation (Exhaustion, numerische Näherungsverfahren, angenähertes Herstellen von Formen).

3)Funktion (Zuordnung, Abbildung, Invarianz)

4)Optimalität (Eigenschaften von Formen, Größen, Zahlen, etc. einer vorgegebenen Bedingung „bestmöglich zu genügen")

5)Charakterisierung (Kennzeichnung von Objekten durch Eigenschaften,Klassifikation von Objekten und Strukturen)"

Die letzte Eigenschaft 5) führt sofort wieder zum Objektorientierten Progammierungsansatz.Die Klassifikation der Objekte erfolgt an Hand ihrer Klasse,ihre Kennzeichnung durch die Reaktion auf verschiedene auf sie angewendete Methoden,die bestimmt wird durch den inneren Zustand (der Datenfelder) des Objekts.

An dem Beispiel erkennt man den engen Zusammenhang zwischen mathematischem Denken und objektorientiertem Entwurfstechniken.

Näheres dazu ist, wie schon oben erwähnt, im nächsten Kapitel nachzulesen.

Die Eigenschaft 1) stellt wiederum die Leitidee des Algorithmus heraus. Wie schon ausgeführt,eignet sich gerade das Gebiet Graphentheorie besonders gut zum Einsatz von Algorithmen. Während die Algorithmen der Analysis meistens reine Zahlenalgorithmen,die lediglich nach Formeln Zahlen verändert darstellen,muß bei den Graphenalgorithmen zusätzlich die einen Graphen kennzeichnende Struktur einbezogen werden.

Das macht die Graphenalgorithmen komplizierter sowie interessanter und führt dazu,dass sie mit dem Hilfsmittel eines Taschenrechners alleine oft nicht auszuführen sind.Vielmehr müssen in den meisten Fällen noch immer eine Folge von die jeweilige Graphenstruktur kennzeichnenden Zeichnungen hinzugenommen werden.

Die Lösung der Probleme wird dadurch wesentlich komplexer und erfordert die Beachtung mannigfaltiger Aspekte.Dadurch eignen sich Probleme der Graphentheorie besonders gut für eine wissenschaftstheoretische Unterrichtsmethode,da hier exemplarisch an einem Beispiel eine Vielzahl von mathematischen Problemen ausgeleuchtet werden können.

Gleichzeitig sind viele graphentheoretische Aufgabenstellungen Abstraktionen einer real existierenden Aufgabe,wie z.B. das Routing-Problem Finden des kürzesten Weges zwischen zwei Orten,das Traveling-Salesman-Problem Finden einer kürzesten Rundreise,das Färbungsproblems Finden einer kleinsten Färbung von Landkarten,das Eulerlinienproblem Finden eines Rundweg usw..

Zur Lösung ist es notwendig die wesentliche Merkmale des gestellten Problems zu entdecken und auf einem Graph zu modellieren,d.h. eine mathematische Modellbildung durchzuführen.(vgl. Eigenschaft 1)

Auch Fragen nach der Berechenbarkeit ergeben sich zwanglos bei der Beschäftigung mit Graphen, wie z.B. das schon weiter oben angesprochene Problem der praktischen Berechenbarkeit bei NP-vollständigen Problemen (z.B. Traveling-Salesman-Problem) als auch Probleme der theoretischen Berechenbarkeit,wie sie aus der Beschäftigung mit endlichen und unendlichen Automaten erwachsen,die (zumindestens teilweise) auch wieder durch Graphen dargestellt werden können.

Als weitere Anwendungen,die wieder einer Modellbildung durch Graphen bedürfen,sind Aufgaben des Operations Research zu nennen.Hierdurch wird nun die Leitidee 4) nämlich die Optimalität angesprochen.Bei Aufgaben des Operations Research sind nämlich fast immer optimale Lösungen bezüglich bestimmten Nebenbedingungen zu suchen.

Z.B. werden beim Minimalkostenproblem alle Flüsse auf Kanten mit minimalen Gesamtosten vom Quellen-zum Senkenknoten gesucht,wobei der Fluss zwischen bestimmten Grenzen liegen muß.Oder beim Transportproblem oder Hitchcockproblem werden alle (Transport-)Flüsse zwischen mehreren Quellen (Produktionsstätten) nach mehreren Senken (Verbraucher) mit minimalen oder maximalen Kosten gesucht.Das Problem des chinesischen Briefträgers sucht einen minimalen Rundweg des Briefträgers,wobei bestimmte Kanten (Straßen) auch mehrfach durchlaufen werden können.

Der Algorithmus des Transport-oder Hitchcockproblems läßt sich so verallgemeinern,dass damit auch Probleme,die sonst dem Simplex-Algorithmus vorbehalten sind,gelöst werden können.Beim Problem des optimalen Matching geht es darum eine Kostenminimale oder Kostenmaximale Zuordnung zwischen den Elementen zweier Mengen zu finden.

Mit dem letzten Beispiel wird deutlich, dass auch Leitidee 3) nämlich der Begriff der Funktion (im Sinne einer Zuordnung) durch Probleme der Graphentheorie abgedeckt werden kann.Läßt man verallgemeinerte Zuordnungen,d.h. Relationen zu,eignet sich die Graphentheorie sogar vorzüglich dazu,diese Zuordnungen zu veranschaulichen und ihre Eigenschaften wie z.B. Transitivität,Reflexivität, Symmetrie, Äquivalenz sowie Ordnungen herauszustellen.

Auch die Leitidee 2) des Näherungsverfahrens läßt sich an Hand von Graphen behandeln. Wählt man z.B. statt des im Programm Knotengraph verwendeten Algorithmus von Engel zur Berechnung von Markovketten einen einfacheren Algorithmus,der Massen von den inneren Knoten zu den Randknoten bzw. innerhalb der inneren Knoten „pumpt“ ,bis sich an der Verteilung der Massen nichts mehr ändert (vgl. die Beschreibung hierzu im Kapitel C XII),erhält man ein Näherungsverfahren,das einem Grenzwert (Grenzverteilung) zustrebt.

Somit lassen sich alle Leitideen des wissenschaftsorientierten Unterricht mit Hilfe von Unterrichtsinhalten der Graphentheorie erreichen.

III)Anwendungsorientierter Unterricht:

Anwendungsorientierter Mathematikunterricht bedeutet Lösung von Problemen aus der Erfahrungswelt und der realen Umgebung des Schülers mit Hilfe von Methoden der Mathematik.Dazu ist zunächst von der realen Situation zu abstrahieren und ein mathematisches Modell zu erzeugen,das die wesentlichen Merkmale des ursprünglichen Problems enthält.

Danach ist das Modell mittels der Anwendung mathematischer Methoden zu lösen,und die erhaltene Lösung ist wiederum bezüglich ihrer Bedeutung für die vorgegebenen reale Ausgangssituation zurück zu übersetzen.

Nach J.Claus (Lit 11, S. 164) lassen sich folgende Ziele nennen:

„1.Vertieftes Verständnis von Situationen des Alltags,die ohne mathematische Behandlung nur lückenhaft verständlich wären.

2.Einsicht in die Bedeutung der Mathematik für die Bedeutung unseres Alltagsleben.“

Nach G. Kaiser-Messmer (Lit 22) lassen sich zwei Richtungen des anwendungsbezogenen Mathematikunterrichts unterscheiden:

„a)Eine eher „pragmatisch“ orientierte Richtung,bei der „utilitaristische“ bzw. pragmatische Ziele nämlich die Befähigung der Schülerinnen und Schüler Mathematik zur Lösung praktischer Probleme anzuwenden - im Vordergrund stehen.

b)Eine mehr an humanistischen Bildungsidealen und der Wissenschaft Mathematik orientierten Richtung,bei der die Befähigung des Lernenden, zwischen Mathematik und der Realität Bezüge herzustellen,in den Mittelpunkt gestellt wird.“

Die zweite Auffassung entspricht der problemorientierten Unterrichtsmethode und wird weiter unten als nächstes Thema behandelt.Also ist hier nur die pragmatische Richtung des anwendungsorientierten Unterrichts zu erörtern.

Gerade beim anwendungsorientierten Mathematikunterricht wachsen die Fächer Mathematik und Informatik zu einer Einheit zusammen,denn oft sind die zur Lösung erforderlichen Algorithmen so kompliziert,dass sie nur mit Hilfe eines Computers gelöst werden können.

Also ist es notwendig,die mathematischen Algorithmen in einer Programmiersprache zu formulieren.Spätestens ab diesem Schritt ist auch das Fach Informatik für den weiteren Lösungsgang verantwortlich.

Es stellt sich an dieser Stelle grundsätzlich die Frage nach dem Verhältnis zwischen Mathematik und Informatik.

Hat sich einerseits die Informatik aus der Mathematik als jüngstes Kind abgespalten,ist es andererseits so,dass die Unterrichtsinhalte der Informatik nicht mehr nur mathematischer Natur sind.Z.B. ist läßt sich die Beschäftigung mit Datenstrukturen oder mit Textverarbeitungssystemen kaum als mathematisches Thema einordnen.Außerdem gibt es natürlich viele Unterrichtsinhalte der Mathematik wie z.B. nichtkonstruktive Existenzbeweise, die nicht unter dem Begriff der Informatik einzuordnen sind.

Große gemeinsame Berührungspunkte gibt es beim Thema Algorithmen.Während hier die Mathematik den Algorithmus in abstrakter Form entwickelt und betrachtet,entwickelt die Informatik konkrete Lösungsverfahren in Form von Programmen in konkreten Programmiersprachen,die auf maschinell nämlich auf einem Computer ausführbar sind.

Auf diesem Gebiet ergänzen sich beide Wissenschaften in idealer Weise.Während die Mathematik die theoretischen Grundlagen bereitstellt,liefert die Informatik die dazu nötigen praktischen Lösungsdurchführungen.

Natürlich gibt es weitere vielfältige Gemeinsamkeiten,wie z.B. bei Fragen der Berechenbarkeit oder der Automatentheorie.

Sollte nun Informatik ein eigenständiges Schulfach sein oder als Anhängsel der Mathematik organisiert werden?

Alleine schon der rasante Fortschritt der Informatik gerade auch auf nicht-mathematischen Gebieten wie z.B. bei den Kommunikationsprogrammen (Internet usw.) und ihre daraus erwachsende gesellschaftliche Bedeutung lassen es dringend ratsam erscheinen,Informatik als eigenständiges Fach zu etablieren.

Bedauerlich ist allerdings,dass der Mathematikunterricht der Schule kaum auf ein in der Informatik vermitteltes Verfahren zurückgreifen kann,weil Informatik derzeit ein nicht für alle Schüler verbindliches Fach ist und in der Sekundarstufe I meistens nur im Wahl(pflicht-)unterricht gewählt werden kann.(In der Sekundarstufe II ist das Fach sowieso nur als nicht obligatorischer Kurs zu wählen.)

Dabei wäre es eine große Bereicherung des anwendungsorientierten Mathematikunterrichts,wenn man zur Lösung der dort allgemein besprochenen Algorithmen gleich das Instrumentarium der Informatik zur Verfügung hätte,um die Aufgaben,die oft zu komplex sind,um sie per Hand lösen zu können, mittels der Erstellung eines Programms anzugehen.

Es soll deshalb hier die Forderung erhoben werden,das Erlernen der Grundlagen einer aktuellen Programmiersprache für alle Schüler zur Pflicht in einem Sekundarstufen-I-Fach Informatik (in dem natürlich auch noch andere Inhalte vermittelt werden sollten) zu machen,so dass verschiedene Fächer auf der Sekundarstufe II,insbesondere auch die Mathematik auf diese Kenntnisse zugreifen können.

Ein Kurs über Graphentheorie könnte davon stark profitieren.Die Behandlung der Graphentheorie bietet sich nämlich als Thema eines Anwendungsorientierten Mathematikunterrichts geradezu an.

Die oben genannten Ziele 1 und 2 lassen sich auf vielfältige Weise erreichen.Wie schon weiter oben erwähnt,gibt es eine Vielzahl Probleme der Graphentheorie,die ihren Ursprung in konkreten Anwendungsproblemen haben.

Genannt werden sollen hier noch einmal die Probleme des Operations Research wie z.B. das Erstellen eines Netz(zeit)planes,das minimale Kostenproblem, das Transport-und Hitchcockproblem,das Problem des chinesischen Briefträgers,das Problem der linearen Optimierung,das Problem des Maximalflusses, das Problem des kürzesten Weges zwischen zwei Knoten, das Problem des Suchen von Pfaden z.B. nach der tiefen Baumsuche,das Problem des Minimalgerüsts (Verlegen von Telefonleitungen kleinster Länge zwischen Häuser),das Euler-und das Hamiltonproblem,das Traveling-Salesman-Problem usw..

Alle diese Probleme sind aus Alltagsproblemen entstanden und bilden deren Abstraktion bzw. Modell.Dadurch kann eine Einsicht in die Bedeutung von Mathematik und Informatik für die Bewältigung unseres Alltagslebens gegeben werden.

Gleichzeitig wird an Hand dieser Probleme auf Grund ihrer realen Komplexität deutlich,dass nur der Einsatz von auf Maschinen (Computern) ausführbaren Programmen eine für praktische Zwecke geeignete Lösung bringen kann.

Der Kritik von Felix Klein (nach J.Claus (Lit 11,S. 168), dass durch die Behandlung von Anwendungen im Mathematikunterricht die eigentliche logische Bildung verkümmert,kann unter dem Aspekt der Erstellung und Codierung von Graphenalgorithmen ganz und gar nicht zugestimmt werden.Die Erstellung solcher Algorithmen ist oft nur mit viel Intuitionsvermögen durchzuführen und gelang z.B. beim bekannten Vierfarbenproblem erst nach vielen vergeblichen Anläufen.

IV)Problemorientierter Unterricht:

Der Unterschied dieser Unterrichtsform zum Anwendungsorientierten Unterricht liegt in der Tatsache,dass beim problemorientierten Unterricht auch Anwendungsaufgaben behandelt werden,die ihren Ursprung nicht in einer realen Anwendungs- und Problemsituation haben.

Im Mittelpunkt des Unterrichts steht die Lösungsmethode von geeigneten vorgegebenen mathematischen Problemen sowie Konzeptionen zur Förderung der selbstständigen Problemlösungsfähigkeiten der Schüler.Dadurch sollen die Schüler mittels Transfer die Problemlösungsfähigkeiten auch auf andere Fä-

cher oder auf Berufssituationen übertragen.Außerdem soll das allgemeine mathematische Denken geschult werden.

Welche Problemstellungen eigenen sich nun besonderes gut für einen problemorientierten Unterricht?

Nach H.Siemon (in Lit 11,S. 174)lassen sich dafür folgende Gesichtspunkte nennen:

„(1)Das Problem muß den Schülern unmittelbar einsichtig sein.

(2) Die Aufgabenstellung muß stark verallgemeinerungsfähig (oder variationsfähig),sein.

(3)Die Aufgabe soll zu vielen falschen Ansätzen verführen,deren Klärung reaktivierend auf bereits vorhandene wirkt oder dem Schüler neue Gesichtspunkte erschließt.

(4)Die Aufgabe sollte interessante Fragestellungen in jedem Schwierigkeitsgrad gestatten."

Die genannten Gesichtspunkte lassen sich mit der Behandlung von Problemen der Graphentheorie abdecken.

Die Probleme der Graphentheorie sind sehr anschaulich,nämlich in Form eines Graphen zeichnerisch darzustellen und die Aufgabenstellungen lassen sich meist auch einem mathematischen Laien sofort erklären,weil sie real existierenden Aufgabenstellungen der Erfahrungswelt entsprechen,z.B. das Problem des Traveling-Salesman,das Routingproblem „Finden des kürzesten Weges zwischen zwei Knoten (Städten)",das tiefe Baumdurchlaufproblem als Suchen eines Ausweg aus einem Labyrinth,das Vier-Farben-Problem als Färben von Landkarten,das chinesische Briefträgerproblem usw..

Die Aufgabenstellungen lassen sich oft stark variieren,z.B. führen kleine Veränderungen des tiefen Baumdurchlauf auf die verschiedenen Problemstellungen Suchen aller erreichbarer Zielknoten von einem (Start-)Knoten aus, Suchen aller Pfade von einem Startknoten aus,Suchen aller Kreise von einem Startknoten aus,Suchen aller minimaler Pfade von einem Startknoten aus,Suchen eines breiten Baumdurchlaufs von einem Startknoten aus usw.,die alle mit ähnlichen Algorithmen zu lösen sind.

Auch das Traveling-Salesman-Problem läßt sich stark variieren.Es läßt jeweils lösen unter dem Aspekt der kleinsten oder größten Reisekosten,der kleinsten oder größten Reisezeit,der kleinsten bzw. größten Entfernungen, der Existenz größter Kreise usw..

Die Lösung vieler Aufgaben der Graphentheorie scheinen zum Greifen nahe, eingeschlagene Lösungswege erweisen sich dagegen bald als Trugschluß.So führt beispielsweise der Versuch die beim Eulerproblem gefundenen Existenzkriterien für eine geschl. Eulerlinie nämlich gerade Grade bei allen Knoten bei der Übertragung auf das sehr verwandt aussehende Problem der Existenz von Hamiltonkreise - es wird lediglich die Forderung nach dem einmaligen Besuch aller Kanten durch den (einmaligen) Besuch aller Knoten ersetzt - zu einem Lösungsfehlschlag.Es gibt überhaupt kein einfaches Kriterium, um die Existenz eines Hamiltonkreises vorauszusagen.

Ausgewählte Aufgaben der Graphentheorie ermöglichen auch Fragestellungen von unterschiedlichem Schwierigkeitsgrad.Beispielsweise kann das selbstständige Finden der Eulerbeziehung zwischen Knoten, Kanten und Flächen an Hand der platonischen Körper schon von einem 10-jährigem geleistet werden, man kann die Beziehung aber auch verwenden,um Ungleichungen zwischen Knotenzahl und Kantenzahl für das schwierige Problem der Nicht-Plättbarkeit von Graphen (notwendige Bedingungen) zu gewinnen.

Aus den Ausführungen ergibt sich,dass man an Hand von Problemen der Graphentheorie problemorientiert unterrichten kann.

V)Genetischer Unterricht nach Wagenschein und Wittenberg:

Dem Problem der Stofffülle im traditionellen Unterricht stellen Wagenschein und Wittenberg das Prinzip des Elementaren und Exemplarischen entgegen. Das bedeutet, dass die charakteristische Struktur eines Stoffgebietes schon an Hand einiger weniger Beispiele erkannt und bearbeitet werden kann.

Einen Unterricht, der nach diesen Prinzipien arbeitet,wird genetischer Unterricht genannt.Wittenberg wählt dafür auch die Bezeichnung Themenkreismethode,und Wagenschein spricht von Plattformen.Sie können nach Schwierigkeitsgrad strukturiert werden und vermitteln auf jeder Stufe Einsichten in die nächsthöhere Stufe.

Wichtig für den genetischen Unterricht ist das Prinzip der Wiederentdekkung.Die Organisation des Stoffes wird bestimmt durch die innermathematischen Notwendigkeiten.Durch die Ausrichtung des Stoffes an der Fachstruktur läßt sich nach Wagenschein und Wittenberg eine große Motivation der Schüler erreichen,da sie ihr persönliches Engagement erfordert.

Der Unterricht soll so angelegt werden,dass der Schüler möglichst viele mathematische Problemlösungen selber entdecken kann.Dazu ist eine Ungesichertheit des Lehrgangs notwendig,damit der Schüler selbst den stofflichen Fortgang des Unterrichts bestimmen kann.Gefordert wird also ein offener Unterricht,der über sich hinaus wachsen kann.

Andererseits wird der Unterricht durch das Prinzip der Abrundung beherrscht,worunter verstanden wird,dass sich der Unterricht in natürlicher Weise zu Zusammenhängen ordnet,die abgeschlossene Einheiten bilden.

Ein systematischer Unterricht,der zunächst Strukturen bereitstellt,um aus Ihnen weitere Erkenntnisse an geeigneter Stelle ableiten zu können,wird abgelehnt.Stattdessen werden die benötigten Voraussetzungen immer dann erst erforscht und begründet,wenn man bei der Lösung eines Problems erkennt, dass man dieser Voraussetzungen bedarf.(Lernen in Kontexten)

Ein wichtiges Mittel ist in diesem Zusammenhang das Beweisen.Durch das Beweisen wird verdeutlicht,wie ein komplexerer Sachverhalt auf einem anderen einfacheren ruht.

Ein Unterichtsgang zum Thema Graphentheorie läßt sich gut nach den Vorstellungen des genetischen Unterrichts strukturieren.Dazu ist eine geeignetes Einstiegsproblem auszusuchen,das dann anschließend vom Schüler nach allen Seiten hin ausgeweitet werden kann.

Als zu behandelnder Themenkreis könnte z.B. der Komplex Eulerbeziehung, ebene Graphen und Färbbarkeit,der im gleichnamigen Kapitel ausführlich dargestellt ist (Kapitel C IV),gewählt werden. Hier soll der Unterrichtsgang nur kurz skizziert werden werden (siehe zu dieser Unterrichtsreihe auch den entsprechenden Unterrichtsplan im Anhang).

Auf Grund einer Untersuchung von verschiedenen ebenen und auch nicht ebenen Graphen fällt sofort die Eulerbeziehung zwischen den Anzahlen von Knoten,Kanten und Gebieten für ebene Graphen auf,besonders,wenn man eine Tabelle mit den entsprechenden Zahlen für verschiedene Graphen erstellt. Die Beziehung ist so einfach,dass sie vom Schüler leicht in einer Formel dargestellt werden kann.

Eine Sonderstellung nehmen bei dieser Untersuchung Graphen ein,die nur ein sie umschließendes Gebiet besitzen,nämlich die Bäume (die Gerüste eines umfassenden Graphen).

Dies führt zur Beweisidee der Eulerformel,nämlich einen komplexen Graphen durch Wegnehmen von Knoten so zu reduzieren, bis nur noch das Gerüst des Graphen als Baum übriggeblieben ist.

Für Bäume läßt sich die Eulerbeziehung jedoch direkt einsehen.

Durch das sich hier als gegebene Notwendigkeit darstellende Beweisprinzip der vollständigen Induktion,zu dem dieses Beispiel den Anlaß gibt,sich mit diesem Beweisverfahren überhaupt zu beschäftigen,läßt sich die Eulerbeziehung dann auch auf komplexere ebene Graphen ausweiten.

Hier zeigt der Beweis,wie ein komplexerer Sachverhalt auf einfacherem ruht (s.o.).

Jetzt kann das Thema ausgeweitet werden,um zu untersuchen,woran es liegt, dass nichtplanare Graphen die Euler-Beziehung nicht erfüllen.Solche Untersuchungen können zu Kriterien in Form von Gleichungen bzw. Ungleichungen führen,die eine notwendige (aber keine hinreichende) Bedingung für das Vorliegen der Planarität eines Graphen darstellen.

Mit Ihnen kann aber entschieden werden (wenn sie erfüllt sind),dass ein vorgegebener Graph nicht planar ist.

Die Begriffe notwendige und hinreichende Bedingung können an dieser Stelle eingeführt werden.

Eine weiter Ausweitung des Themenkreises ist jetzt möglich durch Einbringung des Erbteilungproblems,nämlich ein Land unter 5 Söhnen so zu teilen,dass jeder mit jedem eine gemeinsame Grenze hat.Hier wird man dem Schüler Zeit zum ausführlichen Probieren lassen.

Durch Reduzierung der Länder auf ihre Hauptstädte wird die Graphenstruktur des Problems sichtbar,und das Problem erweist sich als äquivalent einen entsprechenden ebenen Graphen zu erzeugen.An dieser Stelle wird eine nächsthöhere Stufe der Einsicht ereicht,nämlich daß man zu einer Karte einen äquivalenten Graphen erzeugen kann und umgekehrt.

Sollen Schüler an dieser Stelle selbst auf diese Idee kommen,was nach der Theorie des genetischen Unterrichts wünschenswert ist,wird man auch an dieser Stelle viel Zeit für den Unterrichtsgang einplanen müssen.

Durch Verteilung von Farben für die einzelnen Länder bzw. Knoten ergibt sich zwanglos die Einsicht,dass es anscheinend immer ausreicht nur vier Farben für das Färben der Karte zu benutzen (ebener Graph).

Dies führt dann unmittelbar zur Problematik des Vier-Farben-Problems.

An dieser Stelle bietet es sich schließlich an,einen Algorithmus zu entwerfen,um zu untersuchen,wieviele Farben mindestens nötig sind,um einen vorgegebenen Graphen zu färben,d.h. der zum Begriff der chromatischen Zahl.

Dies führt zur Notwendigkeit sich mit der Darstellung von Graphen und der Programmierung entsprechender Algorithmen zu beschäftigen.

Schließlich führt der Begriff der chromatischen Zahl noch zur Definition des paaren Graphen,der die Besonderheit aufweist,immer mit zwei Farben färbbar zu sein.

Als weiteres behandelndes Problem als Beispiel für genetischen Unterricht könnte auch das Thema Optimierung von Lösungen mit Hilfe von Graphen ausgewählt werden, das hier nur kurz angedeutet werden soll.(Vgl. die Kapitel C IX und C XIII über den Maximalen Fluss und über Minimale Kosten)

Als Eingangsbeispiel kann die Aufgabe (Kapitel C IX und C XIII) dienen,in einem Verkehrsverbund mit den Transportmitteln Eisenbahn, Straßenbahn und Bus den kostengünstigsten Weg zu finden.Zunächst einmal wird man sich bei dieser Aufgabe damit beschäftigen,wieviel Personen überhaupt maximal fahren können.

Dies führt dem Algorithmus von Ford-Fulkerson..

Der Algorithmus kann dann zum Algorithmus von Busacker Gowen verallgemeinert werden,wobei es hier notwendig wird,sich mit dem Problem Finden des minimalen Pfades zwischen zwei Knoten eines Graphen zu beschäftigen.

Weitere Verallgemeinerungsstufen des Themenkreises stellen dann das Transport-und Hitchcockproblem dar,die schließlich wiederum zur Beschäftigung mit dem allgemeinen Optimierungsproblem auf Graphen oder aber zur Aufgabenstellung Finden eines Optimalen Matchings anregen.Als Anwendung kann dann noch das Chinesische Briefträgerproblem gelöst werden(siehe zu dieser Unterrichtsreihe auch den entsprechenden Unterrichtsplan im Anhang).

Wie die Beispiele zeigen läßt sich also auch an Hand eines Unterrichtsgangs der Graphentheorie die genetische Unterrichtsmethode verwirklichen.

VI)Projektorientierter Unterricht:

Projektorientierter Unterricht ist ein Unterricht,der sich unmittelbar an praktischen Gegebenheiten des täglichen Lebens orientiert oder aber der Bewältigung technisch-praktischer Fragestellungen dient.Projektunterricht ist oftmals verbunden mit fächerübergreifenden Unterricht,weil ein einzelnes Fach meistens nicht zur Lösung der Probleme,die in der Lebensumgebung der Schüler auftreten,alleine ausreicht.

Dabei ist oft nicht die Entwicklung mathematischer Fähigkeiten das vorrangige Ziel,sondern der Schüler soll ein verbessertes Verständnis der Probleme seiner sozialen Umgebung erhalten.

Bei einem Projekt steht nicht der Lehrer als Organisator und Vermittler des Lernprozesses im Vordergrund,sondern der Schüler soll weitgehend selbstständig das Lernziel auswählen und den eigenen Lernprozess bestimmen.

Da diese Selbständigkeit von Schülern erfahrungsgemäß oft nicht geleistet werden kann,beschränkt sich der in seinen Anforderungen reduzierte Projektunterricht darauf,dass der Schüler zwischen verschiedenen vom Lehrer vorgegebenen Lernzielen und Lerninhalten auswählt und der Lehrer beim Lernprozess des Schülers die Rolle des Beraters bzw. Koordinators einnimmt.

Meistens erfordert ein Projekt auch die Zusammenarbeit mit Mitschülern in einer Gruppe,die alle am selben Projekt arbeiten.Dies bedeutet eigene Tätigkeiten mit anderen abstimmen und gemeinsame Ziele anzustreben,sowie Aufteilung von Aufgaben,Koordination der Aktivitäten und Zusammensetzung aller Teilprojekte.

Speziell im Fach Informatik versteht man unter der Durchführung eines Projekts das Arbeiten von verschiedenen Gruppen an einem Gesamtprogramm,das jeweils in sinnvolle Module aufgeteilt wurde,die von jeder Gruppe selbstständig erstellt werden sollen.

Dabei ist es wichtig,dass sich die Gruppen untereinander koordinieren.Das bedeutet eine genaue Abstimmung des Schnittstellen der Module untereinander sowie eine Spezifikation,was die einzelnen Module genau leisten sollen.

Jede Gruppe testet die Funktionsfähigkeit ihres fertigen Moduls gemäß diesen Vorgaben.Zum Schluss des Projekts werden die Module zu einer Einheit zusammengesetzt.Eine in ihrem Schwierigkeitsgrad nicht zu unterschätzende Phase ist dann die Fehlersuche und das Debuggen des Gesamtprogramms.

Die Informatiklehrpläne aller Bundesländer sehen die Durchführung von Softwareprojekten in der Sekundarstufe II vor.Zur Einübung dieser Unterrichtsform sollten zunächst kleinere Projekte durchgeführt werden.

Informatik ist das Fach, das sich besonders gut für die Durchführung von Projekten eignet.Weil die Erstellung eines umfangreichen Programm für eine Person praktisch unmöglich ist,bietet sich diese Unterrichtsform geradezu an.

Auch die Algorithmen,die zu graphentheoretischen Problemen gehören,sind oft so umfangreich,so dass es sich fast von alleine ergibt die projektorientierte Unterrichtsform bei der Erstellung von Programmen zum Thema Graphen einzusetzen.

Als Zeitpunkt bietet sich hier die Jahrgangsstufe 12 und 13 an (insbesondere im Hinblick auf das Thema Graphen als Behandlung von Datenstrukturen mit den Spezialfällen Bäumen und Listen,die vom Lehrplan gefordert werden).

Auch die Forderung, dass sich ein Projekt am sozialen Umfeld und der praktischen Lebenserfahrung der Schüler orientieren sollte,kann mit dem Thema Graphen erreicht werden.

Wie schon mehrfach erwähnt,entspringen die Aufgabenstellungen der Graphentheorie oft real vorkommenden Problemen,die durch eine mathematischen Modellbildung in diese abstrakte Struktur unter Beachtung der wesentlichen Aspekte der Probleme gebracht wurden.

Die Durchführung eines Projekts kann jetzt auch als Lernziel verfolgen,die soziale Struktur der primär vorhandenen realen Probleme zu untersuchen und aufzuzeigen.

Beispielsweise könnte es ein Lernziel sein,die Problematik von Verkehrsproblemen in Innenstädten zu analysieren und zu untersuchen,inwieweit Optimierungsstrategien der Graphentheorie eine Erleichterung dieses Problems mit sich bringen könnten.(Transport-und Flussprobleme)

Weitere Beispiele sind:

Gibt es eine Möglichkeit,durch Kraftstoffeinsparung die Umwelt möglichst wenig zu belasten,indem die optimale (kürzeste) Route mittels der Graphentheorie zwischen Orten bestimmt wird,und wo in der Praxis bieten sich solche Möglichkeiten an?

Ist es möglich einen Verkehrsverbundplan z.B. durch Einsatz der Netzplantechnik effizienter zu benutzen,so dass man schneller mit öffentlichen Verkehrsmitteln zum gewünschten Ziel kommen kann?

Inwieweit kann man mit der Wahrscheinlichkeitstheorie und speziell mit als Graphen dargestellten Markovketten die Gewinnwahrscheinlichkeiten eines Glücksspiels (Lotto, Roulette usw.) vorhersagen?

Kann man mit Hilfe der Graphentheorie den kürzesten Rundweg,der durch mehrere Städte führt,finden?

Die genannten Aufgabenstellungen bieten auch Anlass zu einem fächerübergreifenden Unterricht.So bietet sich eine Zusammenarbeit mit dem Fach Politik oder Sozialwissenschaften bzw. Wirtschaftswissenschaften an,um Aufgabenstellungen aus dem Bereich Operations Research von gesellschaftsrelevanten Zusammenhängen her zu analysieren.

Es zeigt sich also,dass Projektunterricht eine geeignete Unterrichtsform zur Vermittlung von Unterrichtsinhalten der Graphentheorie ist.

Nach Lenne' (Lit 38) lassen sich neben der genetischen Unterrichtsmethode von Wagenschein und Wittenberg noch zwei andere Hauptrichtungen der Mathematikdidaktik oder besser der Methodik der Mathematik unterscheiden,nämlich die traditionelle Aufgabendidaktik und die Neue Mathematik.

Die genetische Methode wurde oben schon ausführlich erörtert.Bei der Aufgabendidaktik vermittelt der Lehrer (in den meisten Fällen)an Hand von Aufgaben durch Lehrervortrag mathematische Kenntnisse,Methoden und Lösungswege.

Zur Einübung des Verfahrens werden dann Aufgaben gestellt, die von den Schülern bearbeitet und deren Lösungen anschließen vom Lehrer kontrolliert

werden. Das Stellen, Bearbeiten, Lösen und Kontrolle sind dabei die wesentlichen methodischen Elemente der Vermittlung von Wissen.Daher müssen zu allen im Unterricht behandelten Inhalten geeignete Aufgaben bereitgestellt werden.

Bei der Neuen Mathematik unterscheiden sich eine rigorose und eine gemäßigte Richtung.Während in der rigorosen Richtung der Prozess des Deduzierens aus vorgegebenen Axiomensystemen im Vordergrund steht und sich am fachlichen Aufbau der Universitätsmathematik orientiert,legt die gemäßigte Richtung großen Wert auf den Vorgang des Mathematisierung an Hand von Modellen, dem logischen Ordnen der Einzelbeziehungen der Elemente dieser Modelle und dem Erzeugen einer globalen Ordnung durch Zusammenfügen der Elemente und der Herstellung von begrifflichen Zusammenhängen.

Während sich die Strukturmathematik der rigorosen Richtung,die sich insbesondere durch die Grundlegung aller Begriffe an Hand der Mengenlehre und durch formales Ableiten aller weiterführender Erkenntnisse aus diesen Grundlagen auszeichnete,nicht im Schulunterricht durchgesetzt hat und wieder weitgehend aus den Lehrplänen und Schulbüchern verschwunden ist,weil sie sich als Sackgasse erwiesen hat,hat sich die gemäßigte Richtung bewährt,insbesondere weil sie jeweils mit all ihren Teilen in den Zielen der oben diskutierten Methoden des Operationalisierten Unterricht,des Wissenschaftsorientierten Unterricht,des Anwendungsorientierter Unterricht und des Problemorientierten Unterrichts aufgeht.

Graphentheorie könnte nach allen drei Richtungen der Lenne'schen Hauptrichtungen unterrichtet werden.

Zum genetischen Unterricht wurden schon weiter oben Beispiele genannt,und die Methoden der gemäßigten Richtung der neuen Mathematik sind ebenfalls in den obigen Erörterung der verschiedenen Unterrichtsmethoden enthalten. Insbesondere sollte bei der letztgenannten Richtung der Algorithmenbegriff im Vordergrund stehen.

Graphentheorie läßt sich auch nach der Aufgabendidaktik unterrichen,indem man an Hand von Aufgaben im Lehrervortrag Eigenschaften und Algorithmen von Graphen erläutert,und dann geeignete Graphen als Aufgaben vorgibt,an denen diese Eigenschaften verifiziert bzw. entsprechende Algorithmen ausgeführt werden sollen.

So sollte man allerdings Graphentheorie nicht unterrichten,da sich bei einem Unterricht nach den anderen Methoden eine weit größeren Gewinn an mathematischem Können erzielen läßt.Außerdem ist diese Unterrichtsmethode für die Schüler auch am langweiligsten.Natürlich läßt sich auf das Einüben des Stoffes an Hand von Aufgaben schon im Hinblick auf Klausuren nicht verzichten.Deshalb finden sich Aufgaben zur Vertiefung des Gelernten auch in Kapitel C.Falsch ist jedoch die Lösung der Problemstellung vom Lehrer fertig vorzugeben.Vielmehr sollte Graphentheorie so unterrichtet werden,dass Schüler selber Eigenschaften von Graphen entdecken,Graphenmodelle selber aufstellen und Algorithmen selber entwerfen können.

Unter diesem Aspekt sind die Unterrichtsskizzen mit geeigneten Einstiegsproblemen des Kapitels C konzipiert worden,an Hand derer Schüler die Lösungen der Probleme,eventuell durch den Lehrer leicht gelenkt,selber entwickeln können.(Die Einstiegsaufgaben sollten deshalb unter diesen Gesichtspunkten vom Lehrer vorgeführt und präsentiert werden.Vergleiche dazu auch die Ausführungen von Kapitel B III.)

Die Einstiegsprobleme gehen fast ausnahmslos von Problemen der Alltagserfahrung der Schüler aus und sind deshalb zunächst als Graph zu modellieren.Dadurch steht kein Problem isoliert da,sondern das Lernen geschieht im Anwendungszusammenhang,wobei sich oft verschiedene Problemstellungen auseinander entwickeln lassen und aufeinander aufbauen.

Bei einem Unterricht nach der rigorosen Richtung der Neuen Mathematik würde man mit der Mengenlehre beginnen und die Definition des Graphenbegriffs sowie weitere Eigenschaften streng aus diesen Grundlagen axiomatisch unter

formaler Darstellung der Graphen mittels der mengentheoretischen Bezeichnungsweise ableiten und darstellen. Dieser Weg ist jedoch für den Schulunterricht zu abstrakt und auch zu wenig motivierend.

Die bisherigen Ausführungen dieses Kapitels zeigen, dass sich Graphentheorie unter didaktischen und methodischen Gesichtspunkt als sehr flexibles Thema erweist und (beinahe) zu jeder Richtung und Methode sowie zur Erfüllung der allgemeinen und speziellen Lernziele des Mathematik-und Informatikunterichts seinen Beitrag leisten kann.(So dass man am besten sich gar nicht auf nur eine der bisher diskutierten Methoden festlegen,sondern je nach zu vermittelndem Problem immer die jeweils günstigste Methode heraussuchen sollte.)

Erstaunlich ist dann,dass es jedoch bisher im Lehrplan Mathematik (fast) keinen Platz gefunden hat.

Wie schon in der Einleitung erwähnt,ist der Grund dafür wahrscheinlich in der Tatsache zu suchen,dass sich die Problemstellungen der Graphentheorie nicht so sehr auf eine Untersuchung der statischen Eigenschaften eines Graphen beziehen,sondern zu einem großen Teil auf den dynamischen Aspekt der Algorithmischen Graphentheorie ausgerichtet sind.

Diese Algorithmen sind aber keine reinen numerischen Algorithmen,die sich nur auf die Veränderung von Zahlenwerten beziehen,sondern erfordern die Beachtung der durch den Graph vorgegebenen Beziehungstruktur von Knoten und Kanten,wodurch bei der manuellen Ausführung eines solchen Algorithmus eine große Zahl von Zeichnungen des Graphen gemäß der verschiedenen Zustände des Algorithmus erforderlich ist.Dieses ist aber,besonders bei größeren Graphen ein sehr mühsames und wenig motivierendes Verfahren.

Es wird verbessert durch den Einsatz einer durch ein Programm gesteuerten Maschine,d.h. eines Computers,der die verschiedenen Zustände als Textausgabe oder sogar graphisch auf einer Zeichenoberfläche darstellen kann.

Dazu ist der Algorithmus nicht mehr manuell auszuführen,sondern als Programm zu schreiben.Die Beschäftigung mit Algorithmen auf Graphen wird in das Gebiet der Informatik verlagert,und deshalb ist es nicht verwunderlich, dass das Thema Graphen zumindest im Bereich der Hochschulinformatik eine wichtige Rolle spielt.

In den Schulunterricht sind allerdings bislang im wesentlichen die Graphen nur in ihrer Spezialform als Liste oder als Baum eingedrungen.

Diese Arbeit soll dazu beitragen, die Behandlung des Themas Graphentheorie sowohl im Mathematikunterricht als auch im Informatikunterricht populärer zu machen.

Zur Begründung ist hervorzuheben,inwieweit die Behandlung von Unterrichtsinhalten dieses Gebietes Beiträge leisten kann,die von anderen Unterrichtsinhalten dieser Fächer nicht in diesem besonderem Maße erreicht werden können.

Winter stellt (Lit 64,S.59) eine Checkliste zur Auswahl von konkreten Unterrichtsinhalten zusammen,die sich an den allgemeinen und speziellen Lernzielen orientiert.

„Ist der Stoff

1)-dem kognitiven Niveau und den Interessen der Schüler adäquat?

2)-dem Interesse und den didaktischen Möglichkeiten des Lehrers adäquat?

3)-von allgemeinerer mathematischer (oder informatischer) Bedeutung?

4)-in das fachliche Curriculum sinnvoll einzuordnen

Ist der Stoff gerechtfertigt

5)-als „Kulturtechnik"?

6)-durch eine umweltbezogene Sachsituation?

7)-als urteilsbildene Information?

8)-als mathematische Grundlage für technisches oder naturwissenschaftliches Verständnis?

9)-durch seine kulturhistorische Bedeutung?

10)-zur Besinnung auf Möglichkeiten und Grenzen der Mathematik?

11)-durch seinen ästhetischen Wert?

-Ist der Stoff geeignet als Beitrag

12)-zum Anschauungsvermögen?

13)-zum logischen Denken?

14)-zu Kommunikations-bzw. Kooperationsfähigkeit?

15)-zur Sprachförderung? (speziell: Fachsprache und Symbolik sowie Programmiersprache)

16)-zur Kritikfähigkeit?

17)-zum Problemlöseverhalten?

18)-zur Selbstständigkeit?

19)-zur Förderung geistiger Grundtechniken (vergleichen, ordnen, abstrahieren, verallgemeinern, klassifizieren, konkretisieren, analogisieren)?

20)-zur Förderung von „Arbeitstugenden" (Sorgfalt, Genauigkeit, Gewissenhaftigkeit, Klarheit, Ordnung)?

21)-zur allgemeinen oder speziellen Berufsvorbereitung?"

Die folgende Zusammenstellung von Vorteilen orientiert sich an dieser Zusammenstellung und gibt jeweils in Klammern die Nummern der Kriterien der Checkliste an,die erfüllt werden.

Vorteile beim Einsatz von Unterrichtsinhalten aus dem Bereich Graphentheorie:

1)Probleme der Graphentheorie sind durch die Visualisierung mittels der Struktur des Graphen außerordentlich anschaulich und elementar zu formulieren.Sie können oft ohne Rückgriff auf irgendwelche mathematischen Vorkenntnisse entwickelt werden und sind damit auch Personen ohne mathematische Vorbildung sofort verständlich zu machen.

(Kriterien: 2/11/12/15/13/17/18/19)

2)Das Suchen von Lösungen von Problemen der Graphentheorie bedeutet öfters als bei anderen mathematischen Unterrichtsinhalten das Erstellen von nichtnumerischen Algorithmen,d.h. von Algorithmen, bei denen nicht die Veränderung von Zahlen bzw. das Rechnen mit Zahlen sondern das Durchlaufen oder das Ändern einer Struktur,die durch die Beziehung von Knoten und Kanten gegeben ist,im Vordergrund steht oder ausschließlich erforderlich ist (z.B. tiefer Baumdurchlauf usw.).

Die Algorithmen der Graphentheorie sind daher anschaulicher zu verstehen.

(Kriterien: 1/2/12/13/15/17/18/19)

3)

a)Durch die vielfältigen Beziehungen,mit der Knoten und Kanten eines Graphen miteinander verknüpft sind,bieten sich Unterrichtsinhalte der Graphentheorie mehr als andere zur Erforschung, zum Probieren, zum Experimentieren und zu heuristischen Lösungsversuchen,d.h. zum entdeckenden Lernen an.Der Prozess des Mathematisieren kann hier besonders gut durchgeführt werden.

b)Probleme der Graphentheorie regen zum Beweisen an,weil die Beweisschritte oft anschaulich am Graph in Form eines Algorithmus nachvollzogen werden können und daher das Beweisen in einem manuellen Experimentieren an der Graphenstruktur besteht.

R. Bodendiek nennt (Lit 7,S. 39) folgende Beispiele für Problemstellungen, die von den Schülern leicht erraten bzw. erkannt und zum Teil auch alleine bewiesen werden können:

„a)der Graph ist geschlossen (offen) einzügig,

b)der Graph ist Hamiltonsch,

c)zwei Graphen sind zueinander isomorph,

d)die Anzahl der Ecken ungeraden Eckengrades ist in jedem Graph gerade,

e)die Summe aller Eckengrade ist in jedem Graphen gleich der doppelten Kantenanzahl,

f)in jedem Baum B mit n Ecken gibt es genau n-1 Kanten,

g)jeder Weg ist graziös,

h)die platonischen Graphen sind Hamiltonsch und graziös.

(Ein zusammenhängender, schlichter, ungerichteter Graph heißt graziös, wenn es eine surjektiver Abbildung g von den Kanten des Graphen auf die Zahlen von 1 bis k gibt,wobei k die Anzahl der Knoten des Graphen ist und das Bild jeder Kante gleich dem Betrag der Differenz der Bilder seines Anfangs-und Endknotens ist mit einer Abbildung f, die die Menge der Knoten bijektiv in die Menge der Zahlen von 1 bis k abbildet.)

(vgl. zu den obigen Behauptungen auch die Beweise im Kapitel über Knoten, Kanten,Bäume Wege und Komponenten)

(Kriterien: 1/2/3/4/5/7/13/15/17/19/19/20)

4)Probleme der Graphentheorie sind oft keine mathematisch abstrakten Probleme,sondern sind Modelle von real existierenden Problemen.Dadurch wird eine größere Motivation sich mit diesen Problemen zu beschäftigen und diese Probleme zu lösen gegeben.Im Bereich des Operations Research sind die Lösungsmöglichkeiten durch die Methoden der Graphentheorie unverzichtbar.

(Kriterien: 3/4/6/8/17/19/21)

5)Viele Probleme lassen sich auf die Struktur von Graphen zurückführen, nämlich alle die Probleme,die eine zweiseitige Relationen zwischen Elementen einer Grundmenge darstellen.Daher sollte der Begriff der Graphenstruktur und der Umgang mit ihr im Unterricht geübt werden.Die Struktur eines Graphen ist so aufgefaßt auch als Vorstufe zum Funktionsbegriff (eindeutige Relation) zu sehen.Als besondere Eigenschaften einer Relation können die Begriffe reflexiv, transitiv, symmetrisch, antisymmetrisch, äquivalent so-

wie das Erzeugen einer globalen Ordnung mit Hilfe von Graphen veranschaulicht werden.Eine Funktion (mit endlicher Definitionsmenge) läßt sich dann als spezieller gerichteter paarer Graph darstellen.

(Kriterien: 3/4/7/13/17/18/19/20)

6)In der Informatik ist die Struktur von Graphen als Datenstruktur für viele Probleme unverzichtbar(Grund: vgl. Vorteil 5).

Wenn also komplexe Programme mit Hilfe von Programmen gelöst werden sollen, sollte der Graph als Hilfsmittel bereitstehen.

(Kriterien: 3/4/8/13/17)

7)Listen- und vor allen Dingen Baumalgorithmen,mit denen Probleme wie das Suchen und Sortieren sehr effizient gelöst werden können,sind Spezialfälle der entsprechenden Graphenalgorithmen.Durch das Behandeln der Graphenstruktur werden die allgemeinen Prinzipien dieser Algorithmen besser sichtbar und können universeller eingesetzt werden.

(Kriterien: 1/2/3/4/13/15/17/18/19/20)

8)Problemstellungen der Graphentheorie sind mehr als andere Inhalte geeignet,unter Vermeidung von Frontalunterricht mit Hilfe alternativer Unterrichtsformen vermittelt zu werden,z.B. projektbezogen und Schülergruppenzentriert,weil die Erstellung der benötigten Algorithmen das kooperative Arbeiten von Gruppen erfordert.

(Kriterien: 1/2/3/4/14/16/17/18/19/20)

Welche Themen aus dem umfangreichen Gebiet der Graphentheorie sollten für den Schulunterricht als Unterrichtsinhalte ausgewählt werden?

Der Umstand,dass sich bisher im Informatikunterricht der Schule im wesentlichen nur die Behandlung von Listen und Bäumen als Spezialfälle von Graphen findet,ist wahrscheinlich darin begründet,dass das bekannte Buch Algorithmen und Datenstruktur von Nikolaus Wirth (Lit 63),das sich auf die Unterrichtsinhalte des Faches Informatik der Schule wie kein anderes ausgewirkt hat,nur bis zur Komplexität dieser Datenstrukturen führt.

(Der Einfluss Wirths auf den Informatikunterricht rührt daher,dass ihm als Begründer der Sprache Pascal,die hauptsächlich im Schulunterricht eingesetzt wurde,auch besondere Kompetenz bei der Auswahl der Inhalte des Unterricht zugemessen wurde.)

Verfasser anderer Werke neueren Datum zum Thema Algorithmen und Datenstrukturen beziehen die Graphentheorie in größerem Umfang mit ein:z.B. T.Ottmann, P.Wildmayer, Algorithmen und Datenstrukturen (Lit 46).

R. Baumann (Lit 2,S. 263) fordert daher

„die Behandlung der -bisher im Informatikunterricht favorisierten - Sortieralgorithmen und der Baumstrukturen einzuschränken und dafür die kombinatorischen bzw. Suchalgorithmen in allgemeinen Graphen als Schulbeispiele für das Thema Algorithmen und (höhere) Datenstrukturen heranzuziehen. Ferner sollte man nicht den bisher üblichen Weg „Listen (Stapel,Schlangen),Bäume gehen, sondern den Sprung mitten in die kombinatorische Optimierung wagen und die benötigten programmtechnischen Hilfsmittel (beispielsweise Zeigerkonzept in Pascal) ad hoc erarbeiten ."

Baumann nennt fünf Beispiele zum Thema Graphentheorie,die unbedingt behandelt werden sollten und „die so beziehungsfähig sind,dass sich von jedem von Ihnen das gesamte Gebiet der Graphentheorie erschließen läßt."

Die Beispiele sind:

1)Das Eulerproblem

Als praktische Anwendungen werden dabei das Königsberger Brückenproblem und das Parkuhrenproblem (Rundweg für eine Politesse,die an Bürgersteigen mit Parkuhren genau einmal vorbeigehen möchte) genannt.

2)Das Hamiltonproblem

Als praktische Anwendung wird die Springertour auf einem Schachbrett genannt. (Der Springer soll auf jedem Feld des eventuell in der Zeilen-und Spaltenzahl verkleinerten Schachbretts genau einmal zu stehen kommen.)

3)Das Labyrinthproblem

Als praktisches Anwendung wird das Suchen eines Weges vom Eingang eines Labyrinths zu seinem Ausgang genannt. Das Problem läßt sich durch einen TiefenBaumdurchlauf oder durch einen BreitenBaumdurchlauf lösen und lässt sich zur Aufgabe den minimalen Pfad bzw. alle Pfade zwischen zwei Knoten (Ein- und Ausgang) zu finden erweitern (s.u. vgl. 5).

4)Das Traveling-Salesman-Problem

Als praktische Anwendung wird u.a. die Rundreise eines Vertreters durch verschiedene Städte genannt.(Kreis mit besonderen Eigenschaften)

5)Das Problem des kürzesten Weges

Als Anwendungsbeispiel wird die Fahrt eines Frachtschiffes auf dem kürzesten Weg zwischen zwei Häfen genannt.
(Dies bedeutet das Suchen eines minimaler Weg zwischen zwei Knoten.Eine Erweiterung dieser Aufgabenstellung wäre das Suchen aller Pfade zwischen zwei Knoten.)

Alle fünf genannten Algorithmen sind in dem Programm Knotengraph enthalten.

Bielig-Schulz ergänzt die Beispiele Baumanns (Lit 5,S. 38 -42) um ein weiteres Anwendungsproblem auf Graphen:

6)Ein minimal spannender Baum

Als Anwendungsbeispiel wird das Wiederherstellen eines zusammenhängenden Weges in einem Wald genannt.

Als Lösung wird der Algorithmus von Prim vorgeschlagen.

Das Programm Knotengraph enthält (gleichwertig) eine Lösung dieses Problems nach dem Algorithmus von Kruskal.

Falls keine Zeit in einer Unterrichtsreihe zur Verfügung steht,ein Programm dieses Umfang vollständig von den Basismethoden aus beginnend zu erstellen, "sollte man eine solche Umgebung (d.h. einen Modul,der Grundalgorithmen der Graphentheorie bereitstellt) zur Verfügung stellen."(Baumann,Lit 5, S. 42)

Genau dies geschieht in dieser Arbeit mittels der Entwicklungsumgebung von Knotengraph.

Welche konkreten Unterichtsinhalte sollten in mit dem Programm Knotengraph (Konzeption DWK) bzw. mit der Entwicklungsumgebung Knotengraph (Konzeption EWK) durchzuführenden Unterichtsreihen enthalten sein?

I)Zum einen sind dies sicherlich die von Baumann und Bielig-Schulz genannten grundlegenden sechs Graphenalgorithmen (und eventuelle Erweiterungen),von denen aus sich das gesamte Gebiet der Graphentheorie erschließt. Diese Algorithmen lassen sich allgemein unter dem Begriff Suchen von Pfaden und Wegen in Graphen sowie Euler- und Hamiltonlinien zusammenfassen.

In diesem Themenkomplex sind auch die bisher im Informatikunterricht behandelten speziellen Baum-und Listenalgorithmen enthalten bzw. sollten hier eingebunden werden.Dieser Themenbereich ist deshalb auch vom Standpunkt der Informatik aus das wichtigste zu behandelnde Gebiet der Graphentheorie.

Die Skizzen zu Unterrichtsreihen zu diesem I. Komplex sind in den Kapiteln C I Kanten, Bäume, Wege und Komponenten,C II Suchverfahren in Graphen, Baum-und Pfadalgorithmen, Gerüste,C III Alle Pfade sowie minimaler Pfad zwischen zwei Knoten sowie C V Euler-und Hamiltonlinien dargestellt(aber auch im Kapitel B III).

II)Legt man großen Wert auf eine anwendungsorientierte, problemorientierte und projektorientierte Unterrichtsmethode,die auch den Erfahrungsbereich der Schüler besonders einbezieht,sollte eine Auswahl von Problemen aus dem Gebiet Operations Research behandelt werden.Diese Aufgabenstellungen haben überdies den Vorteil,dass sie das wichtige Problem des Optimierens von Lösungen,d.h. die Lösung von (diskreten) Extremwertaufgaben behandeln.

Als wichtige Anwendungen bieten sich hier das Aufstellen eines Netzzeitplans,das minimale Kostenproblem,das maximale Flussproblem,das optimale Zuordnungsproblem,das optimale Transportproblem bzw. das Hitchcockproblem, allgemeine Optimierungsprobleme,wie sie auch mit dem Simplexalgorithmus gelöst werden können und als spezielle Anwendung das chinesische Briefträgerproblem an.

Die Kapitel des Abschnitts C Graph als Netzplan (C VI),Maximaler Netzfluss (C IX), Kosten und Transportprobleme und optimales Matching (C XIII), Das Chinesische Briefträgerproblem (C XIV) beschreiben Skizzen von entsprechenden Unterrichtsreihen.

III)Will man gemäß einem wissenschaftsorientierten Unterricht die Behandlung von Strukturen der Mathematik oder Informatik,wie z.B. den Relations- und Funktionsbegriff bzw. die strukturierte Lösung von Gleichungssystemen oder aber Strukturen der theoretischen Informatik in den Vordergrund stellen,wird man auf die Behandlung von Graphen als Relationen,auf die Erzeugung eines maximalen Matchings in einem bipartiten Graphen (Zuordnungsproblem) und auf das Lösen von Gleichungssystemen mit Hilfe des Algorithmus von Mason sowie als Informatikanwendung auf die Simulation eines endlichen Automaten mit Hilfe eines Graphen Wert legen.

Die Kapitel des Abschnitts C nämlich Graphen als Relationen (C VIII), Maximales Matching(C X),Gleichungssystem (C XI),Graph als endlicher Automat (C VII) enthalten die Skizzen entsprechender Unterrichtsreihen.

IV)Wie schon oben erwähnt läßt sich die Themenreihe Eulerbeziehung,ebene Graphen und Färbbarkeit gut nach der genetischen Unterrichtsmethode behandeln.Es ist aber natürlich auch eine Problemorientierte oder Wissenschaftsorientierte Vorgehensweise möglich,wenn nicht soviel Unterrichtszeit zur Verfügung steht,wie sie bei der Methode nach Wagenschein und Wittenberg erforderlich ist.

Das Kapitel C IV Eulerbeziehung, ebene Graphen und Färbbarkeit skizziert eine entsprechende (genetische oder Problem- und Wissenschaftsorientierte) Unterrichtsreihe.

V)Eines der wenigen Unterrichtsgebiete,bei dem Graphen momentan standardmäßig als mathematisches Hilfsmittel zur Darstellung eingesetzt werden,ist das Gebiet der Stochastik.Deshalb ist es notwendig,auch dieses Gebiet angemessen zu berücksichtigen.

Die Darstellung und Auswertung von Wahrscheinlichkeits(pfad)bäumen fällt unter das schon oben genannte Thema Pfade und Wege in Graphen.Kompliziertere Probleme führen auf Markovketten mit endlich vielen Wahrscheinlichkeitszuständen.Dieser Themenkomplex läßt sich natürlich auch gut unter dem Thema anwendungsorientierter Unterricht einordnen,das hier

jedoch nicht zum Gebiet des Operations Research zählt.Deshalb befindet er sich in einem eigenen Kapitel.

Die Skizze einer zugehörigen Unterrichtsreihe ist im Kapitel C XII Markovketten/Reduktion von Graphen enthalten.

Vom Standpunkt der Informatik aus,bieten alle zuvor genannten Unterrichtsinhalte gute Möglichkeiten komplexe (z.B. objektorientierte) Datenstrukturen und den Umgang mit ihnen zu behandeln und ermöglichen somit gute Ansatzmöglichkeiten für einen Unterricht nach der operativen Unterrichtsmethode.Die Themen eignen sich insbesondere auch dazu die für den Informatikunterricht spezifischen Vorgehensweisen wie projektorientierten Unterricht und die Zusammenarbeit von Schülern bei der Erstellung eines größeren Programmierungsprojekts einzuüben.

Das Programm Knotengraph enthält,wie schon in der Einleitung dargestellt, folgende Anwendungen (Unterrichtsinhalte):

Pfadprobleme (Wege,Bäume,Kreise):

Bestimmung aller Pfade zu den Knoten des Graphen von einem Knoten aus,Bestimmung aller Kreie des Graphen von einem Knoten aus,Bestimmung aller minimaler Pfade zu den Knoten des Graphen von einem Knoten aus,Bestimmung der Anzahl der auf Pfaden erreichbaren Zielknoten von einem Knoten aus,tiefer Baumdurchlauf des Graphen von einem Knoten aus,Inorder-Durchlauf in einem Binärbaumgraphen,breiter Baumdurchlauf des Graphen von einem Knoten aus,(minimaler) Abstand von zwei Knoten des Graphen,binäres Suchen in einem geordneten Baumgraphen,Bestimmung aller Pfade zwischen zwei Knoten des Graphen,Bestimmung eines minimalen Gerüstes des Graphen

Allgemeine Graphenprobleme (Anwendungen):

Bestimmmung eines Zeitplans für einen Graphen,der als Netzplan aufgefaßt wird, nach der CPM-Methode,Bestimmung von Hamiltonkreisen eines Graphen mit Lösung des Travelling-Salesman-Problems,Bestimmung der Eulerlinie (geschl.) eines Graphen,Untersuchung,ob ein vorgegebener Graph (die Knoten) mit einer vorgegebenen Anzahl von Farben gefärbt werden kann,Bestimmung eines Eulerlinie (offen) in einem geeigneten Graph,Simulation eines endlichen Automaten,Bestimmung der Ordnungsrelation sowie der transitiven und reflexiven Hülle eines als Relation aufgefaßten Graphen,Bestimmung des maximalen Netzflusses nach dem Algorithmus von Ford-Fulkerson in einem Graphen mit Quellen- und Senkenknoten,Bestimmung eines maximalen Matchings nach der Methode Suchen eines erweiternden Wegs in einem Graphen,anschauliches Lösen eines Gleichungssystems nach dem Algorithmus von Mason auf einem Graphen,Bestimmung der Übergangswahrscheinlichkeiten und mittleren Schrittzahlen auf einem Graphen,der als absorbierende Markovkette aufgefaßt wird, Bestimmung der Übergangswahrscheinlichkeiten und mittleren Schrittzahlen auf einem Graphen,der als stationäre Markovkette aufgefaßt wird,Reduzieren eines Signalflussgraphen bzw. Bestimmung der Lösungen einer absorbierenden Markovkette durch Reduzierung des Graphen,Lösungen des minimalen bzw. maximalen Kostenproblems und Ermittlung eines geeigneten Graphflusses,Lösungen des allgemeinen Transportproblems bzw. des Hitchcockproblems sowie als Verallgemeinerung Lösen von linearen Optimierungsproblemen auf Graphen,Bestimmung eines optimalen Matchings auf einem bipartiten Graph,Lösen des Chinesischen Briefträgerproblems auf einem gerichteten Graphen.

Die genannten Themen decken auch die Auswahl der in Hochschullehrbücher behandelten graphentheoretischen Probleme gut ab und sollen mit dieser Arbeit dem Schulunterricht zugänglich gemacht werden.

Natürlich können nicht alle genannten Probleme auch Themen einer Schul-Unterrichtsreihe sein,sondern man sollte sich auf einen der oben genannten fünf Themenkomplexe beschränken und unter den dort angesprochenen Inhalten noch eine Auswahl treffen.

Einordnung in den Lehrplan,Wechselwirkung mit konventionellen Unterrichtsinhalten:

Die Forderung Inhalte der Graphentheorie in den Unterrichtsplan Mathematik einzubeziehen,soll natürlich nicht bedeuten,auf die bisherigen konventionellen Stoffe völlig zu verzichten.Graphentheorie rundet die bisher vermittelten Inhalte in hervorragender Weise ab,indem neben dem Betrachten des Grenzwertbegriffs in der Analysis und dem Beherrschen des Unendlichen,neben der Algebraisierung geometrischer Sachverhalte in der linearen Algebra und analytischen Geometrie und neben der Betrachtung von Wahrscheinlichkeitsbeziehungen in der Stochastik ein weiteres Gebiet treten sollte,das die Methoden der diskreten Mathematik auf endlichen Mengen behandelt.

Notwendig ist dazu neben der Präzisierung des Begriffs des Algorithmus die Einführung nötiger Werkzeuge in Form von geeigneten Programmiersprachen in den Unterricht.

H.U. Hoppe und W.J. Luther fordern daher (Lit 18,S. 8-14) die Behandlung folgender Unterrichtsinhalte in der Mathematik:

„
Aussagenlogik und Elemente der Prädikatenlogik (Hornklauseln und Resolutionskalkül)

Boolesche Algebren, endliche Automaten, endliche Körpern, Halbordnungen und Verbände

intuitiver Algorithmusbegriff und elementare mathematische Algorithmen

Kombinatorik, diskrete Strukturen und Graphen,diskrete Geometrie,
Optimierungsverfahren und diskrete Mengen"

Insbesondere für die Behandlung der Themen diskrete Strukturen und Graphen, Optimierungsverfahren und diskrete Mengen,intuitiver Algorithmusbegriff und elementare mathematische Algorithmen sowie endliche Automaten im Mathematikunterricht und Informatikunterricht soll mit dieser Arbeit geworben werden.

Als Konsequenz ergibt sich daraus,dass schon im Mathematikunterricht oder in einem Pflichtfach Informatik der Sekundarstufe I die Algoritmik und die Grundzüge einer Programmiersprache gelehrt werden sollten,um die Behandlung des Themas diskrete Mathematik,insbesondere das Thema Graphen auf der Sekundarstufe II vorzubereiten.

Die so erworbenen Kenntnisse könnten dann natürlich auch anderen Teilgebieten der Mathematik und selbstverständlich auch andern Fächern (z.B. Physik) nutzbringend zur Verfügung stehen.

Eine anderer Aspekt ergibt aus den neuen Beschlüssen der Konferenz der Kultusminister zur Änderung der gymnasialen Oberstufe.In NRW besagt der darauf bezogene im Amtsblatt (Lit 1) veröffentliche entsprechende Erlaß,dass beispielsweise für zwei Kurse des Faches Mathematik in den Jahrgangsstufen 12 und 13 zwei Kurse eines anderen Faches ersatzweise in die Abiturqualifikation eingebracht werden können,sofern in den Kursen dieses Faches mit mathematischen Unterichtsinhalten vergleichbare und gleichwertige Themen behandelt wurden.

(§11,(7):„Sofern die in Grundkursen der Fächer Deutsch, Mathematik, Fremdsprachen zu vermittelnden grundlegenden Kompetenzen in Grundkursen anderer Fächer curricular abgesichert und systematisch ausgewiesen sind,können für Schülergruppen im Rahmen der Profilbildung einer Schule mit vorheriger Genehmigung der oberen Schulaufsichtsbehörde je Fach höchstens zwei,insgesamt höchstens bis zu vier solcher Kurse auf die Beleg- und Einbringungsverpflichtung angerechnet werden (Substitution).Bei Abiturfächern ist keine Substitution möglich)"

Hier bietet sich Informatik als Ersatzfach an.Ein Unterricht in Informatik müsste dann auch mathematische Unterrichtsinhalte abdecken können.Besonders geeignet wären hier die Wahl von Unterrichtsreihen zum Thema Algorithmische Grahentheorie,weil Problemstellungen dieses Gebietes einerseits dem Ursprung nach mathematischer Art und andererseits unter dem Gesichtspunkt Datenstrukturen (Graphen mit den Spezialfällen Listen und Bäumen, Darstellung von Strukturen z.B. algebraischen Termen in Form von Bäumen) schon lange Bestandteile des Faches Informatik sind und deshalb ein Mathematik und Informatik verbindendes Gebiet darstellen.Auf diese Weise ist die Forderung der Vergleichbarkeit und Gleichwertigkeit der Unterrichtsinhalte besonders gut zu erfüllen.

Nach den neuen Richtlinien NRW (Lit 67,S. 25) können auch eine sogenannte „2+2“ bzw. „5+2“-Kopplung von Informatik entweder mit einem Grundkurs Mathematik (2 Wochenstunden Informatik und 2 Wochenstunden Mathematik) bzw. mit einem Leistungskurs Mathematik (2 Wochenstunden Informatik mit 5 Wochenstunden Mathematik) von Gymnasien oder Gesamtschulen als Kombikurse im Rahmen des Konzepts „Zusammenarbeit mit einer Naturwissenschaft“ als Kursfolge in den Jahrgangsstufen 12 und 13 eingerichtet werden.

Für solche Kombikurse gilt erst recht,das im vorletzten Abschnitt Gesagte,dass sich nämlich das Thema Graphentheorie und speziell die in dieser Arbeit vorgestellten Unterrichtsreihen als Unterichtsinhalte,die beiden Fächern entnommen sind,besonders wirkungsvoll einsetzen lassen.Inbesondere in der „5+2“-Kopplung kann,weil genug Zeit zur Verfügung steht und das Lernniveau im Leistungskurs Mathematik wissenschaftspropaedeutisch vertieft werden kann,eine Unterrichtssequenz gestaltet werden,die zunächst - als den Lerninhalten des Faches Informatik entnommen - die Konzeption CAK verfolgt und die Struktur eines Graphen objektorientiert aufbaut,um danach als gemeinsame Inhalte des Informatik- und Mathematikunterrichts Graphenalgorithmen mit Hilfe der graphischen Entwicklungsumgebung von Knotengraph objektorientiert zu entwickeln (Konzeption EWK),wobei der mathematische Anteil in der theoretischen Begründung der Algorithmen z.B. durch den Beweis entsprechender Sätze und in der Entwicklung von graphentheoretischen Modellen besteht,um dann als Abschluß das Programm Knotengraph als fertiges Werkzeug gemäß der Konzeption DWK einzusetzen,um die bis dahin gefundenen Ergebnisse unter rein mathematischen Aspekten mit Hilfe des Demo-Modus und durch Auswahl geeigneter Unterrichtsreihen des Kapitels C (besonders durch Themen des Operations Research und der Wahrscheinlichkeitsrechnung aus den Kapiteln C XII bis C XIV) weiter auszuweiten,und sie so von einem höheren Blickwinkel aus betrachten zu können.

Aber auch in in Kursen,die nur der Informatik oder nur der Mathematik zugeordnet sind,was der Normalfall sein dürfte,können die in dieser Arbeit vorgestellten Aufgaben,Probleme und Algorithmen gleichermaßen behandelt werden,wenn man jeweils das Schwergewicht auf eine der drei genannten Konzeptionen (CAK,EWK,DWK) dem jeweiligen Fachgebiet angemessen legt und eventuell durch einige Aspekte der anderen Konzeptionen geeignet ergänzt.

Um Graphenalgorithmen mit Hilfe einer Programmiersprache darstellen zu können, benötigt man,wie schon eben erwähnt, geeignete Datenstrukturen. Bei umfangreichen Problemstellungen ist es dabei nach dem weiter oben gesagten (Bielig-Schulz) sinnvoll,Schülern schon geeignete Softwaresteine, die Grundoperationen zur Darstellung und Erzeugung von Graphen enthalten,zur Verfügung zu stellen.

Wünschenswert ist dabei, dass der Benutzer dieser Bausteine trotzdem alle Freiheiten behält,die von ihm zu erstellenden Algorithmen seinen Wünschen und der jeweiligen Aufgabenstellung anzupassen.

Um diese Forderungen zu erfüllen, ist der Einsatz objektorientierter Datenstrukturen nötig. Der didaktischen und methodischen Analyse des Einsatzes dieser Datenstrukturen ist das nächste Kapitel gewidmet.

B II Didaktische und methodische Analyse der Verwendung von objektorientierten Datenstrukturen

„Algorithmen und Datenstrukturen" heißt der bekannte Titel eines Standardwerkes zu diesen beiden Themen von dem Begründer der Programmiersprache Pascal Nikolaus Wirth.Der Titel verweist schon darauf,dass diese beiden Bestandteile eines Programms eng zusammengehören.Algorithmen manipulieren Daten und ein Algorithmus,der ohne Daten arbeitet,bewirkt nichts.Andererseits werden die Daten eines Programms (nur) durch Algorithmen verändert.

Es liegt also nahe,diese beiden Komponenten eines Programms zu einer Einheit,dem Objekt zusammenzufassen.

Die Verschmelzung hat den Vorteil,dass die Gestaltung des Objekts so gewählt werden kann,dass nur noch die zum Objekt gehörenden Algorithmen die Daten des Objekts verändern können (bzw. sollen),so dass dadurch eine größere Übersichtlichkeit,Gliederung und Transparenz innerhalb des Programms geschaffen werden kann,wodurch die Fehleranfälligkeit reduziert wird,weil der interne Ablauf einzelner Programmteile voneinander unabhängig wird.

Die Implementierung der zum Objekt gehörenden Algorithmen heißen Methoden. Die zum Objekt gehörenden Speicherstrukturen für die Daten heißen Datenfelder oder Attribute.Nur die Methoden eines Objekts sollten auf die eigenen Datenfelder zugreifen können.

Dieses Prinzip nennt man Kapselung des Objekts.Die Aktivität eines Objekts besteht in der Ausführung seiner zulässigen Methoden.

Von jedem Objekt können beliebig viele Instanzen (Exemplare) gebildet werden.Alle Instanzen eines Objekts gehören zur gleichen Objektklasse.Der Aufruf einer Methode kann als das Senden einer Nachricht an das Objekt aufgefaßt werden.Das Objekt bzw. die Realisierung des Objekts in Form seiner Instanz reagiert darauf mit der Ausführung der angesprochenen Methode.Dadurch und nur dadurch sollte sein innerer Zustand,der durch die Werte seiner Datenfelder bestimmt wird,geändert werden.Dafür gibt es in vielen objektorientierten Sprachen auch Schlüsselwörter wie private oder public,die den Zugriff auf die Datenfelder und auch die Benutzung von Methoden) regeln.

Von einer Objektklasse können durch Vererbung Nachfolgerklassen (Subtyping) gebildet werden.Die Nachfolgerklassen erben das Verhalten,d.h. die Datenfelder und die Methoden des Vorgängers.Zusätzlich können sie neue Verhaltensmuster in Form von zusätzlichen Datenfeldern und Methoden enthalten.Ein Objektklasse kann durch mehrfache Anwendung der Vererbung immer weitere Nachfolger erzeugen.

Bei manchen Programmiersprachen ist es auch erlaubt,dass ein Objekt die Eigenschaften von mehreren Vorgängerobjekten durch Vererbung erhält.(Mehrfach-Vererbung) So gibt es in Java beispielsweise zwei Möglichkeiten,Nachfolgerobjekte zu definieren,nämlich durch eine Klassen-oder Schnittstellendeklaration (mittels extends bzw. implements).Während bei der Klassendeklaration,bei der alle Methoden und Attribute auch auf die Subklassen vererbt werden,nur Einfachvererbung möglich ist,erlaubt es die Schnittstellendeklaration (Interface) mehrere Vorgängerobjekte zu umfassen.Die Schnittstellendeklaration bedeutet einen abstrakten Objekttyp,ohne dass eine Implementation von Methoden vorhanden ist,zu definieren,wobei diese Implementation erst in den von dieser Superklasse abgeleiteten Nachfolgeobjekten vorgenommen wird.Auch Delphi gestattet die Deklaration von abstrakten Objektklassen,aber ohne die Möglichkeit der Mehrfachvererbung. (nachträgliche Bemerkung:erst ab Delphi 5.0 gibt es auch Interfaces.)In der Sprache C++ ist Mehrfachvererbung sogar mittels der Klassendeklaration möglich.Mehrfachvererbung in dieser Form bringt allerdings auch den Nachteil erhöhter Fehlerquellen infolge von Unübersichtlichkeit der Objektbeziehungen mit sich.

Wenn ein und dieselbe Nachricht an verschiedene Objekte (bzw. an die Instanzen von verschiedenen Objekten) gesendet wird,können die Objekte unterschiedlich reagieren.Dieses Verhalten bezeichnet man als Polymorphie.

Genauer gesagt gibt es verschiedene Arten der Polymorphie.Das Überladen von Methoden mit gleichem Namen beim Subtyping,bei dem Nachfolgerobjekte anders auf den Aufruf der Methode als Vorgängerobjekte reagieren,bezeichnet man als Ad-hoc-Polymorphie.Als Parametrische Polymorphie wird das Umwandeln eines Objekts in einen neuen Typ durch Type-Casting bezeichnet,das in den manchen Sprachen wie Delphi nur bei Objekttypen derselben Vererbungshierachie funktioniert.Subtype-Polymorphie bedeutet das Einfügen von Objekten unterschiedlichen Typs mittels eines gemeinsamen Behälter in eine Speicherstruktur,z.B. eine Objektliste.Dieses Verfahren wird beim Programmsystem Knotengraph beispielsweise zum Aufbau der Graphenstruktur benötigt,wobei in den Knoten-und Kantenlisten Objekte verschieden Typs,der von einem Anwender der Entwicklungsumgebung nachträglich noch festgelegt werden kann,gespeichert werden.Der Behälter ist dabei der Objekttyp TObject.Leider unterstützt Delphi diese Form der Polymorphie nicht besonders gut,weil es nämlich keine Schnittstellendeklaration (Interface) wie in Java gibt.(nachträgliche Bemerkung:erst ab Delphi 5.0 gibt es auch Interfaces.)Auch ist die Möglichkeit der Parametrisierung von Klassen unflexibler als in der Sprache C++,wodurch mittels der Definition eines Typparameters (sozusagen als Platzhalter),noch nachträglich die Instanzierung mit unterschiedlichen Objekttypen möglich ist.(Dadurch gestaltete sich die Programmierung der Objekte des Programmsystems Knotengraph schwieriger.Klassen als Parameter kommen im Programm Knotengraph bei der Erzeugung von Instanzen der Objektklassen TInhaltsgraph,TInhaltsknoten und TInhaltskante zum Einsatz,damit sich die in TInhaltsgraph enthaltenden Methoden auch auf Graphen,Knoten und Kanten von deren Nachfolgerklassen anwenden lassen.)

Verallgemeinert kann das Objektmodell nach Baumann (Lit 2,S. 279) folgendermaßen beschrieben werden:

„Objekte sind eigenständige Einheiten, die einen Zustand besitzen.
Der Zustand wird durch Attributwerte bestimmt und kann unmittelbar nur durch das Objekt selbst,mittelbar über Nachrichten ermittelt oder verändert werden.
Objekte werden von anderen Objekten durch Nachrichten zur Ausführung von Zustandsänderungen veranlasst.
Objekte sind Exemplare einer Klasse,die gemeinsame Attribute und Aktionen zusammenfaßt.
Die Datenstruktur und die Wirkungsweise der Methoden sind gekapselt.
Klassen stehen in Beziehung zueinander,es gibt Teilklassen und Oberklassen.
Eine Teilklasse unterscheidet sich von ihrer Oberklasse dadurch,dass sie mehr Attribute aufweist,also eine Spezialisierung beschreibt.
Die Teilklasse erbt Attribute und Methoden der Oberklasse.Sie kann die geerbten Methoden modifizieren und neue definieren.“

Durch das Prinzip der Vererbung sowie der Kapselung von Methoden und Daten wird die Erzeugung wiederverwendbarer Softwarebausteine wesentlich gegenüber bisher verwendeten Konzepten vereinfacht:

Die Wiederverwendbarkeit von Software,ohne Objekte zu benutzen,erfolgt durch Zusammenfassung von Prozeduren und Funktionen in Softwaremodulen bzw. Klassenbibliotheken.Der Ausgangswert der Funktionen bzw. die Wirkungsweise der Prozeduren hängt dabei ausschließlich von den an sie in Form von Parametern übergebenden Werten ab und kann deshalb als funktional charakterisiert werden.

Die Prozeduren und Funktionen, die in einer Klassenbibliothek zum Zwecke der Wiederverwendbarkeit zusammengefasst werden,müssen daher abstrakt funktional programmiert werden und dürfen nicht auf die speziellen Eigenschaften einer bestimmten Programmierungsumgebung zugeschnitten sein.

Anders ist es bei der Verwendung von Objekten in einer Klassenbibliothek.Das Verhalten der Objekte hängt außer von den an sie von außen übergebenden Parametern auch noch von ihrem inneren Zustand,vorgegeben durch ihre

Zustandsvariablen (der Werte der Datenfelder) ab.Ihre Reaktion kann deshalb nicht nur funktional beschrieben werden,sondern ist außerdem abhängig von der Beeinflussung durch vor der aktuellen Anforderung stattgefundenen Aktionen.

Das Modell das einem solchen Verhalten in der theoretischen oder technischen Informatik entspricht,ist der Mealey-Automat,dessen Ausgangsverhalten durch den inneren Zustand und durch die aktuellen Werte der Eingangsvariablen bestimmt wird.

Durch diese erweiterten Eigenschaften von Objekten kann sich ein Objekt flexibler als bei reiner funktionaler Programmierung verhalten.

Objekte können sich der jeweils vorhanden Umgebung anpassen bzw. schaffen sich durch Speichern entsprechender Informationen eine eigene Umgebungsinformation.Durch das Prinzip der Vererbung können den Objekten zusätzliche neue funktionale Eigenschaften gegeben werden,durch das Prinzip der Polymorphie kann die den funktionalen Zusammenhang darstellende Funktion neu definiert und auf die jeweils speziell geforderte Programmsituation zugeschnitten werden.

Die Vorteile liegen darin,dass Software nicht mehr von Grund auf neu programmiert werden muß,sondern dass sich die Entwicklungszeiten von Programmen durch die Benutzung wiederverwendbarer Bausteine verkürzen.

Die Struktur von Objekten in Programmiersprachen kann in Analogie gesetzt werden zum allgemeinen Objektbegriff der Ontologie.

Die Grundstruktur der Ontologie ist das Ding.Es ist entweder elementar oder setzt sich aus anderen Dingen zusammen.Jedes Ding besitzt Eigenschaften. Nur solche Eigenschaften werden von uns wahrgenomen,denen wir zuvor mindestens ein Attribut zugeordnet haben.Die Eigenschaften zusammengesetzter Dinge werden auf die einzelnen Komponenten vererbt.Dinge sind jeweils einzigartig.Wenn zwei Dinge als identisch erscheinen,dann deshalb weil ihnen nicht in allen ihren Eigenschaften Attribute zugeordnet worden sind.Dinge mit ähnlichen Eigenschaften können zu Klassen zusammengefasst werden.

Nach Bunge und Wand lassen sich folgende Analogien zwischen den Dingen der Ontologie und den Objekten von Programmiersprachen herstellen (Lit 49,S. 153 und 155):

Dinge:

„Die Welt besteht aus Dingen.

Form und Gestalt sind Eigenschaften der Dinge.

Dinge stehen in Welchselwirkung mit anderen Dingen.

Jedes Ding ändert sich.

Nichts entsteht aus dem Nichts und kein Ding wird zu Nichts.

Jedes Ding richtet sich nach den Gesetzen."

Objekte:

„-Die Welt besteht aus Objekten.

-Objekte beschreiben konkrete Dinge im Gegensatz zu Typen oder Klassen.

-Objekte sind bekannt (oder beobachtbar) durch ihre Eigenschaften.Eigenschaften sind entweder Attribute oder Gesetze.Gesetze schränken die Kombinationen der Attributwerte ein.Die Darstellung einer Eigenschaft ist entweder eine Funktion (Attributzuweisung) oder ein Prädikat,das anzeigt,ob eine Zustandsänderung eingehalten wird.

Die Menge der Eigenschaften,die ein Objekt beschreiben,ist abhängig von der Sichtweise und der Modellierungsabsicht.Diese Menge heißt funktionales Schema.Die Werte eines Attributs zu einem bestimmten Zeitpunkt werden durch eine Zustandsvariable erfaßt.Die Menge aller Zustandsvariablen definieren den Zustand eines Objekts.Gesetze bestimmen die erlaubten Zustände eines Objekts

-Objekte sind elementar oder zusammengesetzt.

Die Eigenschaften eines zusammengesetzten Objekts sind entweder vererbbar auf diesen Komponenten,oder sie haften allein der Zusammensetzung an."

Der Aufbau aller heutiger objektorientierter Programmiersprachen orientiert sich stark an dem Aufbau des Prototypen der Objekt-Sprache nämlich Smalltalk (Entwicklung 1972 bis 1980),deren Aufbau wiederum seine Wurzeln in der Sprache Simula hat,von der die erste Version im Jahre 1967 entstand.

Die Konzeption von Smalltalk läßt sich folgendermaßen beschreiben:

Objekte sind Softwareeinheiten,denen ein äußeres Verhalten und eine innere Struktur zugeordnet werden können.Durch die innere Datenstruktur können die Objekte nacheinander verschiedene Zustände einnehmen.Diese Zustandsänderungen werden durch das Empfangen von Nachrichten von anderen Objekte ausgelöst werden.

Umgekehrt kann deshalb jedes Objekt auch Nachrichten versenden und dadurch bei anderen Objekten Aktionen auslösen.Das theoretische Modell dabei besagt,dass jedes Objekt unabhängig von den anderen agiert,und die Aktionen der dabei Objekte parallel ablaufen.Bei den meisten Programmsystemen (Einprozessorsystemen) kann jeweils in der Praxis immer nur ein Objekt momentan aktiv sein,d.h. die Aktivitäten laufen als Sequenz nacheinander auf den verschiedenen Objekten ab.

Die Struktur eines Objekt ist daher zweigeteilt.Nach außen hin sind nur die an andere Objekte ausgesendeten Nachrichten (z.B. als Reaktion auf empfangene Nachrichten) sichtbar.Das Innere des Objekts bleibt für die Außenwelt der anderen Objekte unzugänglich,d.h. ein Objekt ist eine nach außen hin gekapselte Struktur (Datenkapselung).Die innere Struktur wird durch Datenfelder dargestellt,die verschiedene Werte annehmen können und die nur über die Nachrichtenschnittstelle (Interface) des Objekts mittels Methoden aber nicht direkt geändert werden können (bzw. sollten).

Damit ein Objekt auf eine Nachricht reagieren kann,muß es erkennen,dass die Nachricht für genau dieses Objekt bestimmt ist und welche Art von Aktion ausgelöst werden soll.Dazu enthält jede Nachricht eine eindeutige Information,welche Methode eines Objekts gemeint ist und übergibt eventuell noch weitere Parameter,die die Ausführung dieser Methode steuern sollen.

In Delphi,der zur Erstellung des Programms und der Entwicklungsumgebung Knotengraph verwendeten Objektsprache,besteht eine Nachricht in der Zusammenfügung eines eindeutigen Objektnamen (abgetrennt durch einen Punkt) mit der eines eindeutigen Methodennamens, vergleichbar in der Syntax mit einer Procedure.(Zusätzlich können auch noch Parameter übergeben werden).

Wenn ein Objekt nicht der Empfänger der Nachricht ist,reagiert es auf das Senden einer Nachricht nicht oder mit dem Erzeugen einer Fehlermeldung,es sei denn,es besteht die Möglichkeit die Nachricht z.B. an das Vorgängerobjekt (Superklasse) durch Vererbung weiterzuleiten(,das eventuell darauf reagieren kann).

Ein Softwaresystem kann man bezüglich der Objekte in zwei Bereiche einteilen:den Bereich der statischen Objekte und den Bereich der dynamischen Objekte.

Der Bereich der statischen Objekte ist die Menge der Objekte,die schon vom System vorgegeben werden und durch Vererbung ihrer Eigenschaften für die Erzeugung der dynamischen Objekte bereitsteht.

In den meisten objektorientierten Programmiersprachen gibt es z.B. eine Reihe von in der Sprache vordefinierten Objektklassen,z.B. in Delphi der Grundtyp TObject und viele der zur Erzeugung einer Benutzeroberfläche unter Windows vorgegebenen (Komponenten-)Typen (Klassenbibliothek).

Der dynamische Bereich wird vom Anwender des Systems gestaltet und umfaßt Objekte,die vom Anwender erzeugt und nach Gebrauch wieder gelöscht und aus dem Speicher entfernt werden.

Die Eigenschaften eines Objekts,d.h. seine innere Datenstruktur und seine nach außen sichtbaren Methoden werden durch die Objektklasse bzw. durch den Objekttyp eines Objekts beschrieben.In Delphi z.B. wird der Typ bzw. Klasse durch eine Type-Anweisung festgelegt,wobei das reservierte Wort class zu benutzen ist:

```
type  TObjekt=class
              .
              .
              .
              end;
```

Die freie Bereich ... enthält dann die Datenfelder und die Methoden.

(In Java wird ebenfalls das reservierte Wort class,allerdings mit etwas anderer Syntax benutzt.)

Die Objektklassen entsprechen dabei den abstrakten Datentypen (ADT) der theoretischen Informatik,deren Verhalten durch die axiomatische Definition der gegenseitigen Beziehungen ihrer Methoden festgelegt wird.

Von einer Objektklasse lassen sich beliebig viele Objekte mit denselben Eigenschaften,auch als sogenannte Instanzen des Klassentyps bezeichnet, durch Aufruf eines sogenannten Constructors erzeugen.

(In Delphi benutzt man dazu den Constructor create:z.B. in dem obigen Beispiel Objekt:= TObjekt.Create,in Java ist die Bezeichnung New für diesen Operator zu verwenden.Die Objekte müsssen,um den von Ihnen belegten Speicherplatz wieder freizugeben,wenn sie nicht mehr benötigt werden,wieder aus dem Speicher gelöscht werden.Dies erfolgt in Delphi mittels der Methode Free durch den Programmierer,in Java geschieht dies automatisch.)

Die Anzahl der Objektinstanzen eines Objekts bilden die extensionale Definition der Klasse.Die Anzahl der Eigenschaften,die zur Definition eines Objekts notwendig ist,d.h. die Datenstrukturen und Methoden bezeichnet man als die intensionalen Eigenschaften der Objektklasse.

Die Objektklassen kann man nun wiederum auch wieder als Objekte auffassen. Zu ihren Methoden gehören insbesondere die Standardmethoden zum Erzeugen und Löschen von Objekten dieser Klasse (in Delphi der Constructor Create und der Destructors Free).

In Smalltalk kann eine Klasse dabei auch wiederum Datenfelder in Form der Klassenvariablen enthalten.Auf diese Variablen kann dann von allen Instanzen dieser Klasse zugegriffen werden.Außerdem enthält Smalltalk im Gegensatz zu z.B. Delphi und C++ das Prinzip der Metaklassen.

Diese Metaklassen dienen zur Erzeugen der Objektklassen,wobei eine Metaklasse nur eine einzige Klasse als Instanz haben darf.Durch das Vorhandensein der Metaklassen wird das Konzept der Vererbung vereinheitlicht,indem die Erzeugung von Klassen und Instanzen nach einem ähnlichen Schema abläuft.In Delphi gibt es die Möglichkeit eine abstrakte Objektklasse durch die Anweisung class of zu definieren,z.B. für das obige Beispiel:

TObjectclass = class of TObject

Zwischen den Klassen bzw. den Objekttypen kann,wie schon mehrfach erwähnt, eine Vererbungsbeziehung definiert werden.Das erbende Objekt (Objekttyp) erhält auf diese Weise alle (intensionalen) Eigenschaften des Vorgängerobjekts.Auf diese Weise lassen sich durch mehrfache Anwendung der Einfachvererbung baumartige Objekthierachien aufbauen,die sich alle von einem Ausgangsobjekt an der Wurzel ableiten.Bei Mehrfachvererbung treten hierbei Graphenstrukturen auf.

Die Beziehung von Vorgängerobjekt (Superklasse) zum vererbten Objekt (Subklasse) läßt sich analog einer Client-Server-Beziehung beschreiben.

Das Server-Objekt (Auftraggeber) fordert die Ausführung einer Aufgabe und delegiert (schickt) sie zur Ausführung an das Client-Objekt (Auftragnehmer oder Empfänger),das die Aufgabe löst.Das den Auftrag (die Nachricht) versendende Objekt (d.h. der Server) kennt den Namen des Objekts,an die es die Aufgabe deligiert.Umgekehrt ist der Auftraggeber (Server) vom Client aus gesehen unbekannt.

Während der Zeit, in der der Auftrag an den Client delegiert wurde,wartet der Server auf die Abarbeitung der Aufgabe bis die Aufgabe gelöst wurde.

Wird eine Nachricht an die Instanz eines Objekts in einer Subklasse gesendet,wird zunächst dort nach einer entsprechenden Antwortaktion (Methode) auf die Botschaft gesucht.Wenn eine solche Methode dort nicht gefunden wird,wird die Nachricht an die nächste Superklasse weitergegeben,bis dort eine Methode gefunden wird oder der Prozess bei der obersten Superklasse angelangt ist.Wenn auch dort keine Methode vorhanden ist, bleibt die Nachricht unbeantwortet,oder es erfolgt eine Fehlermeldung.

Es ist,wie schon oben erwähnt,möglich die Methode einer Superklasse in einer Subklasse neu zu definieren,so dass dort eine andere Implementation vorliegt.Diese Verhalten ist die (Ad Hoc-) Polymorphie (s.o.).Wenn erst zur Laufzeit des Programms festgelegt wird,welche Aktion jeweils von einem jeweiligen Objekt beim Beantworten einer Nachricht ausgeführt wird,bezeichnet dies als dynamische Bindung,wenn die Festlegung schon zum Zeitpunkt der Compilierung vorgenommen wird,nennt man die Bindung statisch (statisches Überladen von Methoden).Für die Realisierung der dynamischen Bindung wird extra eine besondere interne Datenstruktur,nämlich die virtuelle Methodentabelle für Objekte angelegt.Methoden dieses Typs bezeichnet man deshalb auch als virtuelle Methoden (Deklaration als virtual bei der Ursprungsmethode und override bei den Methoden der Nachfolgerobjekte des Subtypings in Delphi).

Die gegenseitigen Beziehungen der Objekte in einem objektorientierten Programmsystem lassen sich durch drei Beziehungen kennzeichnen:

1)Die „ist"-Beziehung.
2)Die „hat"-Beziehung
3)Die „kennt"-Beziehung

Die „ist"-Beziehung ist die Vererbungsbeziehung zwischen Objektklassen.Jedes Nachfolgerobjekt „ist" auch der Objekttyp des Vorgängerobjektes.Z.B. „sind" alle Objekte des Programms Knotengraphs auch vom Typ „TObject",welches die oberste Superklasse ist.

Wenn ein Objekt wiederum von einem anderen Objekt aus durch einen Constructor-Aufruf erzeugt wird,spricht man von der „hat"- oder „besitzt"-Beziehung.Beispielsweise „besitzt" oder „hat" der Typ TGraph im Programm Knotengraph eine Knoten- und eine Kantenliste vom Typ TKnotenliste bzw. TKantenliste,in der die Knoten bzw. die Kanten des Graphen gespeichert werden.Beide Typen sind Objektklassen, die sich wiederum von TListe durch eine „ist"-Beziehung ableiten.

Der Destruktor eines Objektes,das eine Objektklassen „besitzt",muß diese Objekte erst aus dem Speicher entfernen,bevor das Objekt selbst aus dem Speicher entfernt wird.

Auf Objekte,die ein anderes Objekt „hat",kann nur durch bzw. über dieses Objekt zugegriffen werden. Beispielsweise lassen sich Knoten und Kantenliste im Programm Knotengraph nur mittels Graph.Knotenliste bzw. Graph.Kantenliste ansprechen,wenn Graph ein Objekt vom Typ TGraph ist.

Die „kennt"-Beziehung ist die unabhängigste Beziehung zwischen Objekten. Sie liegt dann vor,wenn in der Methode eines ersten Objekts eine Objektmethode eines zweiten (unabhängigen) Objekts aufgerufen wird,oder anders ausgedrückt wenn das erste Objekt an das zweite Objekt eine Nachricht sendet, die von diesem Objekt in Form einer bestimmten Aktion beantwortet wird.

Eine Möglichkeit der programmtechnischen Realisierung der „kennt"-Beziehung in Delphi ist,dass das eine Objekt Parameter einer Objektmethode des anderen Objektes ist (,wenn man nicht mit globalen Variablen arbeiten will).

Die Bemerkungen der vorigen Abschnitte charakterisieren die Programmierung mit objektorientierten Datenstrukturen.Warum aber sollte dieses Prinzip beim Programmieren im Schulunterricht eingesetzt werden.Welche Vorteile bietet es?

Seit Bestehen des Faches Informatik an der Schule sind die Betriebssysteme, die zum Ausführen von Programmen notwendig sind,immer komplexer geworden. Während es früher z.B. unter dem System DOS genügte,ein Programm mit zeilenorientierter Textausgabe oder Texteingabe zu erstellen,wird unter den graphischen Betriebsystemen wie z.B. Windows eine graphisch orientierte Ein-und Ausgabe in Form der Windowsspezifischen graphischen Steuerelementen wie Labels, Edit-Text-Fenster, Rollbalken usw. erwartet.Eine solche graphisch orientierte Benutzeroberfläche erleichtert einem Anwender die Bedienung eines Programms nicht unerheblich.

Dagegen ist die elementare (d.h. imperative) Programmierung solcher graphischen Elemente sehr aufwendig und erfordert genauste Detailkenntnisse der entsprechenden Betriebssysteme.

Hier erweist sich die Bereitstellung von Objekten als sehr nützlich,die dem Programmierer die mühevolle Detailprogrammierung abnehmen und schon gleich Methoden zum Erzeugen der entsprechenden graphischen Benutzerelemente zur Verfügung stellen.Wenn diese noch wie in Delphi visuell lediglich durch Anklicken mit der Maus bereitgestellt werden,wird das Erstellen von Quellcode nochmals wesentlich vereinfacht.

Ein (scheinbarer) Nachteil dieses Verfahrens ist jedoch,dass die eigentliche Funktionsweise eines solchen Objekts für den Programmierer (Schüler) unsichtbar bleibt und sozusagen eine Black-Box darstellt.

Das Objekt stellt sich also für den Benutzer dar als eine Struktur,von der bekannt ist, wie sie auf bestimmte Operationen reagiert.In welcher Weise diese Reaktionen geschehen ist unbekannt.

Dieses gerade beschriebene Modell ist jedoch das Modell des abstrakten Datentyps.

Unter einem abstrakten Datentyp versteht man einen Datentyp,der durch die Festlegung aller auf Objekte dieses Typs anwendbaren Operationen spezifiziert wird,wobei der Benutzer nicht zu wissen braucht,wie das Objekt im Speicher repräsentiert wird und wie die Algorithmen realisiert werden.Möglicherweise sind verschiedene gleichartige Implementation möglich.

Als ein Lernziel des Informatikunterricht wird allgemein amgesehen,von realen Datenstrukturen zu abstrahieren und die wesentlichen Eigenschaften einer Datenstruktur axiomatisch beschreiben zu können.Das zugehörende allgemeine Lernziel ist,dass der Schulunterricht dazu befähigen sollte,die we-

sentliche Aspekte eines Sachverhalts von den unwesentlichen zu trennen und diese Eigenschaften abstrakt formulieren zu können.

So erweist sich der oben angesprochene scheinbare Mangel als Vorteil.Der Schüler lernt nämlich durch das objektorientierte Programmieren den Umgang mit abstrakten Datentypen,die lediglich durch die axiomatische wechselseitige Wirkungsweise ihrer Methoden definiert werden.

Durch die Verwendung von Abstrakten Datentypen (ADT) ergeben sich (Lit 42, S.302) die folgenden neuen methodischen Aspekte:

„1.Die Bereitstellung eines ADT ermöglicht es,den bearbeiteten Problemen angepaßte Lernumgebungen zu wählen.Auch komplexe Probleme sind „einfach“ zu lösen.

2.Die Erweiterung der Möglichkeiten der benutzten Programmiersprache durch ADTs macht den Wechsel des Systems mit den dabei erforderlichen Umstellungen oft überflüssig.

3.Die Aufspaltung eines Problems in erforderliche Datenstrukturen und Anwendungen derselben verkleinert den Umfang der Detailprobleme und ermöglicht arbeitsteiligen Unterricht.

4.Unterschiedliche Implementierungen der ADTs ermöglichen starke Binnendifferenzierungen.

5.Die Beschränkung auf die bereitgestellten Operationen trennt die Beschreibung der Problemlösung weitgehend von den speziellen Möglichkeiten der benutzten Programmiersprache.Syntaktische Einzelheiten müssen natürlich beachtet werden,sind aber bei der in diesem Buch zu Grunde liegenden Sicht auf den Informatikunterricht nebensächlich und werden bei Bedarf in der Umgebung erlernt,in der gearbeitet werden soll.“

Im Programm Knotengraph (DWK) sowie in dessen Entwicklungsumgebung EWK (als auch in der textorientierten Version CAK) wird aufbauend auf dem abstrakten (allgemeinen) Datentyp einer Objekt-Liste der Datentyp eines Graphen als Aggregation der drei Objektklassen TGraph,TKnoten und TKante zunächst als reine Knoten-Kanten-Relation mit den dazugehörigen Verwaltungsoperationen wie Knoten oder Kante hinzufügen bzw. löschen usw. ohne spezielle Eigenschaften wie z.B. der graphischen Darstellung dieser Objekte also (im wesentlichen) als ADT in der Unit UGraph implementiert.Wenn auch keine axiomatische Definition des Graphen als abstrakter Datentyp vorgenommen wird,werden doch die gegenseitigen Beziehungen der Methoden der drei Objekte TGraph,TKnoten und TKante durch die Auswahl und das Zusammenwirken der in dieser Unit vorhandenen Methoden und Datenfelder verdeutlicht,zunächst ohne dabei konkrete Anwendungsalgorithmen zu berücksichtigen,die spezielle Graphenimplementationen erfordern würden.

Erst in der Unit UInhgrph werden der abstrakten Graphimplementation dann durch Vererbung Inhalte wie z.B. Bezeichner bzw. Werte von Knoten und Kanten,X-und Y-Koordinaten der Mittelpunkte der Knoten auf der Benutzeroberfläche,die Farbe mit der ein Knoten oder eine Kante gefärbt wird usw. also Eigenschaften der graphischen Darstellung hinzugefügt.

Von einem Benutzer der objektorientierten Entwicklungsumgebung können dann jederzeit noch weitere beliebige Eigenschaften der genannten Art hinzugefügt werden,so dass aus der abstrakten Implementation eines Graphen ein konkreter,an das jeweils gestellte Problem angepaßter Graphtyp durch das Vererbungsprinzip entsteht.

Die oben beschriebenen Vorteile der objektorientierten Programmierung lassen sich umschreiben mit dem Stichwörtern Erstellung von Softwaretools und Wiederverwendung von Softwarebausteinen.

Durch den großen Umfang heute üblicher Programmsysteme ist es für Schüler nur noch selten möglich,ein Programm mit angemessenem Leistungsumfang ins-

besondere unter einem graphisch-orientiertem Betriebssystem von Grund auf neu zu programmieren.

Hier setzen die Softwarebausteine ein,nämlich vorgegebene fertige Algorithmen,die vom Schüler zur Erstellung des Gesamtprogramms benutzt werden können.

Ein besonderen Vorteil bieten solche Softwarebausteine,wenn sie nicht in konventioneller (imperativer) Form vorliegen, sondern als Objekte erstellt wurden.

Dann ist nämlich nicht nur möglich,die Softwarebausteine so wie vorgegeben einzusetzen,sondern sie können durch Vererbung vom Schüler an die in seinem Programmierungsprojekt notwendigen Erfordernisse individuell angepasst werden.

Dieses Prinzip wird im Programm Knotengraph verwendet.Der vorgegebene Graph kann durch Vererbung individuell,z.B. durch Definieren neuer Methoden sowie durch Hinzufügen neuer Datenfelder zu Knoten und Kanten jederzeit neu angepasst werden.

Trotzdem lassen sich alle vorgegebenen Methoden,insbesondere die Methoden zum Erzeugen und Editieren von Kanten und Knoten,Methoden zur Dateiverwaltung oder aber auf Grund der visuellen Vererbung des gesamten Anwendungsfenster der Entwicklungsumgebung die gesamte Benutzeroberfläche wie vorgegeben einsetzen und braucht nicht neu programmiert zu werden.Quellcodeerstellung ist nur für die vom Schüler neu zu erstellenden Anwendungsalgorithmen der Algorithmischen Graphentheorie nötig,die die vorhandenen Objekte benutzen und eventuell durch weitere Vererbung auf neue Graphtypen erweitern.

Die Entwicklungsumgebung Knotengraph ist also als wiederverwendbares objektorientiert-programmiertes Softwaretool für den Einsatz zur Lösung von Problemen der Graphentheorie konzipiert.

Die Entwicklung leistungsfähiger und damit größerer Programme bringt es mit sich,dass ein solches Programm von einem Programmierer alleine nicht mehr erstellt werden kann.Daher müssen mehrere Programmierer an Teilaufgaben arbeiten.Das Programm muß also in sinnvolle Untereinheiten so genannte Module zerlegt werden,und jeder Programmierer (oder eine Teilgruppe von Programmierern) bearbeitet dann einen solchen Modul alleine.Als wichtigste Aufgabe entsteht dadurch die Notwendigkeit,Absprachen über das Zusammenfügen dieser Module zum Gesamtprogramm,den sogenannten Schnittstellen zwischen den Modulen zu treffen.

Das Bearbeiten eines solchen Gesamtprogramms durch mehrere Gruppen heißt Programierungsprojekt.Die Durchführung von solchen Programmierungsprojekten ist im Informatikunterricht der Sekundarstufe II vom Lehrplan her zwingend vorgeschrieben.

Das Fach Informatik war eines der ersten Fächer,in dem das Arbeiten von Schülern an Projekten obligatorisch vorgesehen war.Projektorientierter Unterricht wird heute allgemein als wichtige methodische Unterrichtsform angesehen,und es wird allgemein gewünscht,dass ein solcher Unterricht, wegen der damit verbundenen einzuübenden Sozialformen des „Aufeinander--Angewiesens“ und des Zusammenarbeitens von verschiedenen Personen (Schülern) an einem Thema in allen Fächern stattfinden sollte.

Im Fach Informatik ergibt er sich zwanglos aus der Aufgabenstellung der anzugehenden Probleme.

Ein Modul,der Teil eines Programmierungsprojekts ist,enthält eine Exportspezifikation(-Liste),in der die Namen (von Methoden oder Prozeduren) aufgeführt sind,die auch anderen Modulen zur Verfügung gestellt werden sollten (Definition der Schnittstelle).

Die Struktur der den Zustand des Moduls repräsentierenden Daten ist außerhalb des Moduls unbekannt.Die Daten werden ausschließlich über die exportierende Schnittstelle verändert.

Wie nicht unschwer zu erkennen ist,bedeutet dieses Modell gerade wieder das Modell des abstrakten Datentyps,der wie schon weiter oben erwähnt,durch objektorientierte Datenstrukturen am besten realisiert wird.

Ein wesentliches Kriterium für Qualität eines Softwarebausteins ist Flexibilität,d.h. die vielfältige Verwendbarkeit für verschiedenartige Zwecke. Daraus ergibt sich,dass sich ein Modul also am besten durch die Implementierung von einem oder mehreren Objekten realisieren läßt,die flexibel auf Anforderungen ihrer Umgebung reagieren können.

So führt das Konzept der Projektorientierten Programmierung fast zwangsläufig zum Konzept der Objektorientierten Programmierung.

Eine Softwarebibliothek ist dann eine Menge von Objektklassen,und ein Softwarebaustein wird durch eine oder mehrere dieser Objektklassen dargestellt.

Umgekehrt beeinflusst die objektorientierte Programmierung die projektorientierte Entwicklung von Programmen:

Probleme bei der Erstellung größerer,umfangreicherer Programme sind nämlich die Komplexität dieser Systeme zu beherrschen,dem System eine Struktur zu geben,die Schnittstellen verläßlich zu gestalten und die Software wartbar zu halten.Außerdem sollten gute Softwaremodule wiederverwendbar sein.

Alle diese Prinzipien werden durch die Verwendung objektorientierter Datenstrukturen unterstützt.Die Strukturierung des Systems wird vorgeben durch die Vererbungsbeziehung des Objekte.Die Schnittstellenkontrolle entspricht der Datenkapselung der Objekte,wobei nur die vom Objekt vorgegebenen Methoden Zugriff auf die eigenen Daten(-felder) haben dürfen.Durch ihre Strukturierung und ihre Kapselung wird die Software wartbar gehalten:

Im Fehlerfall ist dann eventuell nur der Austausch der Implementation einer oder mehrerer Objektmethoden nötig.Die vorgegebene Beziehung der Objekte und ihre gegenseitige Wechselwirkung kann erhalten bleiben.Die Wiederverwendbarkeit der Software wird hingegen durch die Prinzipen der Vererbung und der Polymorphie gewährleistet.

Die bisherigen (imperativen) Methoden zur Beherrschung der Komplexität wie strukturierte Programmierung,schrittweise Verfeinerung von Problemen (Top to Down-Methode) und Modularisierung sind mit der objektorientierten Programmierung um eine weiteres wichtiges Werkzeug verbessert worden.

Das Prinzip der strukturierten (imperativen) Programmierung reduziert die Möglichkeiten einer Sprache auf die vier Elemente Programmsequenz,Verzweigung, Schleife und Procedure.Die schrittweise Verfeinerung beginnt mit einer Teilung des gestellten Problems in Proceduren,die dann ebenfalls nach und nach in weitere Proceduren durch deren Aufruf zergliedert werden,bis man auf der Ebene der Systemproceduren (Elemente der Programmiersprache) angekommen ist.

Das Prinzip der Modularisierung zerlegt ein größeres Softwareprojekt in überschaubare sinnvolle Teilbereiche,in die Module, die wiederum aus einer Menge von Proceduren bestehen.

Der objektorientierte Ansatz dagegen strukturiert die Daten eines Programms nach der Art und Weise,wie diese Daten bei verschiedenen algorithmischen Zugriffen (Methoden) verwendet werden und erzeugt so eine Klassifikation nach Datentypen nämlich den Objektttypen.

Durch die mit Hilfe der objektorientierten Programmierung möglichen Gestaltung auch größerer Programmsysteme,erhält der Schüler ein realistischeres Bild der Informatik.Möglich wird dadurch ein anwendungsorientierter Unter-

richt,z.B. im Hinblick auf die Lösung von Problemen der Algorithmischen Graphentheorie.

Probleme von realistischer Komplexität können so in der Schule behandelt werden.

Es können nämlich jetzt auch solche Anwendungen behandelt werden und in den Schulunterricht mit einbezogen werden,die bisher wegen ihres Umfangs nicht von Schülern in der Zeit,die dem Unterricht zur Verfügung steht,bewältigt werden konnten.Nur so kann der Forderung, anwendungsbezogene Inhalte und damit die gesellschaftlichen Aspekte des Einsatzes von Computern (siehe das vorige Kapitel B I Didaktische und methodische Analyse der Behandlung von Graphen und Graphentheorie im Unterricht) in den Unterricht einzubinden,Rechnung getragen werden.

Z.B. erlaubt es das Programm Knotengraph bzw. dessen Entwicklungsumgebung, Probleme des Operations Research auf der Grundlage des vorgegeben Objektsystems graphisch anschaulich zu lösen.So kann z.B. der gesellschaftliche Aspekt von Optimierungsstrategien,z.B. bei der Schaffung von Energiesparenden Transportwegen diskutiert werden.

Ein Programm wird nicht mehr vom Kleinem zum Großen fortschreitend aufgebaut,sondern die wechselseitige Beziehung der verschiedenen globalen Programmteile steht im Mittelpunkt des Interesses.Dadurch können neben den technischen und algorithmischen Aspekten auch anwendungsbezogene Aspekte in den Unterricht mit einbezogen werden.

Beim objektorientierten Entwurf eines komplexen Software-oder Anwendersystems wird von den Details zunächst abstrahiert,vielmehr interessiert die Grobstruktur,die Architektur des Softwaresystems.

Damit zeigt sich,dass der Einsatz der objektorientierten Programmierung ein neues Verfahren,d.h. ein Umbruch ist,der es gestattet die Entwicklung eines großen Softwareprojektes einheitlich und übersichtlich zu gestalten.

Einen solchen Umbruch bzw. den Wechsel zu einer neuen (Programmierungs-) Methode bezeichnet man als Paradigama.In vielen heutigen Sprachen ist nicht nur eine dieser Methoden realisiert,sondern sie bestehen aus einer Mischung dieser Konzepte,wobei auf die Realisierung eines dieser Konzepte das Hauptgewicht gelegt wird.

Es gab in der Vergangenheit verschiedene Paradigmenwechsel in der Informatik:

Die in der Informatik zeitlich erste Methode, Programme zu erstellen,auf die natürlich auch heute noch nicht verzichtet werden kann, besteht in der Aneinanderreihung von Einzelbefehlen oder Anweisungen einer Programmiersprache,die sequentiell (unter Einbeziehung von Sprüngen) abgearbeitet wurden.Diese Methode bezeichnet man als imperatives Paradigma.Diese Möglichkeit ist der Maschinensprache eines Rechners am meistens verwandt.

Als abschreckendes Beispiel dieser Methode soll hier nur die unkontrollierte Verwendung der Goto-Anweisung (z.B. in Basic) erwähnt werden,um im Programmcode willkürlich herumzuspringen.Allerdings war die (kontrollierte) Verwendung der Goto-Anweisung unbedingt nötig,um (anfangs) Schleifen programmieren zu können.

Ein anderes Konzept ist die Programmierung mittels Funktionen.Die Anweisungen eines Programms bestehen darin,dass Funktionen wiederum als Parameter (Variablen) anderer Funktionen verschachtelt werden können.Eine solche Art zu Programmieren heißt funktionales Paradigma und wurde zum ersten Mal in der Programmiersprache Lisp realisiert.

In Sprachen wie Prolog wird die Spezifikation eines Problems in einer logischen Sprache beschrieben,und das entsprechende Programm liefert daraufhin die Lösung.Im Idealfall könnte man die Forderung aufstellen, dass die Beschreibung sogar in der Umgangssprache möglich sein sollte,wenn sie nur

hinreichend präzise gestaltet wird.Eine Wechsel zu Sprachen mit diesen Eigenschaften wird deklaratives Pardigma genannt (z.B. in Prolog realisiert).

Der Übergang zu einer Sprache mit objektorientierten Datenstrukturen heißt objektorientiertes Paradigma.Die Beschreibung der Eigenschaften und didaktischen Vorteile dieser Sprachen sind Inhalt dieses Kapitels.

In dem Buch „Das Objekt-Paradigama in der Informatik“ von Klaus Quibeldey-Cirkel (Lit 49) zeigt der Verfasser,dass die Verwendung objektorientierter Strukturen die Kommunikation und Kooperation der an einem Projekt beteiligten Partner fördert.Der gestalterische Aspekt des Softwareentwurfes wird durch den Objektorientierten Ansatz gefördert.

Betrachtet man den Einsatz von Objekten im Informatikunterricht unter diesen Aspekten,wird deutlich,dass mit ihrer Hilfe völlig neue didaktische Ziele erreicht werden sollen und können,als das Erlernen eines einfachen Algorithmenentwurfs und die Kenntnisse über die Funktionalität von Datenstrukturen,die bisher im Mittelpunkt des Unterrichts stand.

Das abstrakte didaktische Ziel ist vielmehr die Beschäftigung mit Informatik als einer interdisziplinären Wissenschaft des Entwerfens und Modellierens von komplexen Programmen.

Somit wird Informatik zur Wissenschaft des Entwurfs und der Gestaltung von Informationssystemen.

(vgl. Lit 3)

Normalerweise besteht die Erstellung eines komplexen Programmsystems in der Ausführung folgender Schritte:

1)Beschreibung des Problems

2)Analyse des Problems

3)Erstellung eines theoretischen Modells des Lösungsalgorithmus

4)Verfeinerung des Lösungsmodells unter dem Gesichtspunkt der Implementation in der vorgegebenen Programmiersprache

5)Codierung in der Programmiersprache

Beim objektorientierten Programmieren sind die Entwurfsschritte 2) bis 5) nicht unabhängig voneinander,sondern müssen von Beginn an unter dem Aspekt der aufeinander aufbauenden Objektstruktur und der günstigsten Auswahl der Objekttypen,die von der Allgemeinheit zum Speziellen in der Objekthierachie voranschreiten sollte,gesehen werden,wobei eventuell auch noch die Objektlinien verschiedener Objekte nicht nur durch Vererbung sondern auch durch Verschachtelung miteinander verzahnt sind.Unter dem letzteren ist zu verstehen,dass z.B. ein Objekt wiederum als Datentyp eines anderen Objektes auftreten kann („hat-Beziehung).

(Z.B. ist sind Programm Knotengraph die Objekte Knoten-und Kantenliste Datenfelder des Graphenobjekts.Die Knoten-und Kantenlisten sind wiederum Listen von Objekten der Typen Knoten und Kante.Von den Knoten und Kanten leiten sich durch Vererbung dann die vom späteren Softwaretool-Benutzer einzusetzenden erweiterten Knoten und Kanten sowie vom Graphen ein erweiterter Graphtyp als jeweilige Objekte ab.)

Nach Klaus Quibeldey-Cirkel (Lit 49, S. 110) läßt sich der Entwurfsprozeß einer objektorientierten Anwendung in zwei Phasen gliedern:

„
1.Explorative Phase:

„Finde Klassen,verteile Aufgaben,finde Kooperationen!“

(a)Identifiziere die Kassen durch die grammatikalische Analyse der Anforderungsbeschreibung.

(b)Identifiziere die Pflichten durch eine Klassenkritik und durch die Analyse der informationstragenden Verben in der Anforderung.

(c)Identifiziere die Kooperationen durch eine Analyse der Kommunikationspfade zwischen den Klassen,die für eine Dienstleistung erforderlich sind.(Kooperationsnetze beschreiben den Informationsfluss auf den hierarchischen Klassenbeziehungen.)

2.Verfeinerung des Entwurfs:

Ordne die Klassen, die Aufgaben,die Kooperationen!

(a)Stelle die Klassenhierachien auf.(Venn-Diagramme beschreiben die Überschneidungen der verteilten Aufgaben.)Identifiziere die abstrakten und die konkreten Klassen.

(b)Identifiziere die Teilsysteme.(Teilsysteme sind Mengen von Klassen,die eng zusammenarbeiten,um die Verträge mit externen Auftraggebern zu erfüllen.)

(c)Schreibe die Protokolle für die Kooperationen zwischen den Klassen und den Teilsystemen."

Also muß das Design einer solchen Objektstruktur vor Beginn jeder Implementierung wohlüberlegt sein.Die Erstellung eines solchen Designs wurde unter dem oben genannten Stichwort Modellieren des Programm- bzw. Informationssystems zusammengefasst.

Baumann schreibt dazu ((Lit 2, S. 281):

„Der Erwerb von Kompetenz in Analyse und Entwurf von Informatiksystemen ist ein langandauernder Prozess.Es bedarf der Erfahrung,die relevanten Objekte und ihre Beziehungen zu erkennen: Wenn die Schüler zu spät mit dem Modellieren beginnen,können sie diese Erfahrung nicht mehr aufbauen."

Objektorientiertes Denken verlagert die Schwerpunkte des Unterrichts von der Programmierkompetenz hin zu einer Modellierungs- und Evaluationskompetenz"

Die Grenzen zwischen Analyse und Design eines gestellten Problems verschwinden.

Analyse meint die wesentlichen Merkmale einer verbal gestellten Aufgabe herauszufiltern,und sie von unwesentlichen zu trennen,d.h. festzulegen,was sollen die Kern-Eigenschaften des zu entwerfenden Systems sein.

Design bedeutet das Erfinden von Datenstrukturen und Methoden,um die Lösung der gestellten Aufgabe erfassen und gestalten zu können,sowie die Gliederung in Teilprobleme und Modelle,die Definition von Schnittstellen usw..

Bei dem objektorientierten Systementwurf werden die verbal formulierten Anforderungen des Problems von vorneherein durch Objekte unseres Denkens, d.h. durch Begriffe gegliedert und als Lösungseinheiten dargestellt und durch diesen frühzeitigen Abstraktionsprozeß schon so strukturiert,dass sich das Design des Lösungsprozesses aus ihnen unmittelbar ergibt.

Die Klassen-und Objekthierachien des Objektorientierten Systementwurfs entstehen also automatisch als Struktur unserer Denkpsychologie.

Zur Festlegung eines Begriffs gehört die Angabe aller Merkmale (in Delphi durch Propertys realisiert),durch die man entscheiden kann,ob ein bestimmtes Objekt zu diesem Begriff gehört oder nicht.Die Menge aller Objekte,die zu einem bestimmten Begriff gehört,d.h. auf die die Merkmale des Begriffs zutreffen,ist die Klasse des Objekts.

Zur Festlegung eines Begriffs d.h. einer Klasse ergeben sich bei dem dafür notwendigen Abstraktionsprozeß automatisch Ober-und Unterbegriffe,d.h. Ober-und Unterklassen,und dadurch entsteht von selbst die Vererbungsbeziehung von Objektklassen.Die am weitesten oben in dieser Hierachie stehenden Klassen sind dann die abstrakten Datentypen.Ein Schreiten von den obersten Objektklassen zu den weiter unten befindlichen bedeutet Spezialisieren,der umgekehrte Weg heißt Generalisieren.

Verschiedene Begriffe oder Objekte mit ähnlichen Eigenschaften oder ähnlicher Zielrichtung werden zu einem Objektverbund zusammengefaßt.Dadurch entsteht schon bei der Analyse des Problems von alleine eine Programmstrukturierung.

Objektorientierte Strukturen entsprechen also der Struktur unseres Denkens. Nach H.Lompscher (in Lit 49, S. 134) lassen sich die geistigen Fähigkeiten auf wenige Operationen zurückführen,die allerdings meistens nicht isoliert sondern im Verbund auftreten:

„
1.Zergliedern eines Sachverhalts in seine Teile

2.Erfassen der Eigenschaften eines Sachverhalts

3.Vergleichen von Sachverhalten hinsichtlich der Unterschiede und Gemeinsamkeiten

4.Ordnen einer Reihe von Sachverhalten hinsichtlich eines oder mehrerer Merkmale

5.Abstrahieren als Erfassen der in einem bestimmten Kontext wesentlichen Merkmale eines Sachverhalts und Vernachlässigen der unwesentlichen Merkmale

6.Verallgemeinern als Erfassen der einer Reihe von Sachverhalten gemeinsamen und wesentlichen Eigenschaften

7.Klassifizieren als Einordnen eines Sachverhalts in eine Klasse

8.Konkretisieren als Übergang vom Allgemeinen zum Besonderen
"

Die Operationen haben jeweils ihre direkte Entsprechung in den Merkmalen der objektorientierten Programmierung:

Zu 1,2,3:Objekte und ihre Eigenschaften

Zu 4:Zusammenfassen von Objekten zu einem Objektverbund

Zu 5,6:Generalisieren bis zum abstrakten Datentyp

Zu 7:Konzept der Klasse, Vererbungsbeziehung

Zu 8:Spezialisierung von Objekten

Objektorientierter Entwurf bedeutet Klassifizieren,d.h. geeignete Objektklassen finden,Kategorisieren d.h. Objekte diesen Klassen zuordnen,Strukturieren, d.h. die Zusammenhänge zwischen Objektklassen erkennen sowie Systematisieren, d.h. die Objektklassen in einer Vererbungshierachie zu ordnen.

Nach der kognitiven Psychologie läßt sich die Struktur unseres Gehirns als semantisches Netz mit verknüpften Gedächtnisinhalten in Form eines Graphen darstellen.Semantische Netze sind Abbildungen unserer Wissen-oder Gedächtnisstrukturen.Den Knoten entsprechen dabei die Gedächtnisinhalte und den Kanten die Verknüpfungen der Gedächtnisinhalte als Relationen.

Dabei lassen sich nach Dörner (in Lit 49,S. 132) drei Arten von Verknüpfungen unterscheiden:

a)die Konkret - Abstrakt - Relation

b)die Ganzes-Teil-Relation

c)die Raum-Zeit-(Ablauf)Relation

Der Denkprozess kann dann als interpretierender Prozess aufgefaßt werden, bei dem gewisse Knoten (Gedächtnisinhalte) verfügbar gehalten werden und untersucht wird,ob andere Knoten (andere Gedächtnisinhalte) im Umfeld dieser Knoten liegen.

Diese Struktur des Denkens als semantisches Netz entspricht beim objektorientierten Entwurf den strukturellen Beziehungen bei der Problemanalyse sowie dem Design und schließlich den Relationen der Programmobjekte zueinander in der zu erstellenden Anwendung.

So kann der Vorgang des Modellierens als zunehmende Konkretisierung von der Ebene der Gedächtnisinhalte zur Ebene der Programmiersprache aufgefaßt werden,wobei die Anfangsstrukturen erhalten bleiben.Dass diesem Prozess eine Graphenstruktur zu Grunde liegt,betont noch einmal die Wichtigkeit der Beschäftigung mit Graphen,wie sie durch diese Arbeit hervorgehoben werden soll.

Auch die menschliche Sprache zeigt (wahrscheinlich auf Grund der Art der oben erörterten Denkvorgänge) Parallelen zur objektorientierten Struktur.

Vererbungsbeziehung: z.B. Vogel als Oberbegriff (Klasse) von Möwe

Vögel können fliegen.Die Möwe ist eine Vogel.Also kann eine Möwe fliegen.

Überdefinieren:z.B. eine goldene Brücke bauen.Der ursprüngliche Begriff von bauen wird verändert.

Polymorphie:Jetzt ist Zeit zur Muße.Verschiedene Leute werden unter dem Begriff Muße jeweils Verschiedenes verstehen.

Auf diese Weise ist der Vorgang des Modellierens mit Hilfe objektorientierter Datenstrukturen viel stärker an die Strukturen unseres Denkens und Sprechens angelehnt als frühere Programmentwurfstechniken und daher viel leichter von Schülern durchzuführen.

Dabei sollten die von der realen Welt vorgegebenen Probleme direkt als Modell in die Strukturen der Objekte übertragen werden,wobei die gegenseitigen Beziehungen der realen Gegenstände sich homomorph in den gegenseitigen Relationen und Abhängigkeiten der Objekte wiederspiegeln.

Der objektorientierte Ansatz verändert die Bedeutung bisher verwendeter Begriffe des Softwareengenierings,und es entsteht eine neue Methode des Softwareentwurfs.

Eine neuer Methodenaspekt ist das gerade beschriebene einphasige Modellieren des vorgegebenen Problems mittels Objektstrukturen.

Daraus ergeben sich weitere Veränderungen:

Der Begriff des (fertigen) Programms muß erweitert werden,und bedeutet jetzt ein Softwaresystem,das einerseits aus schon bereitgestellten Softwaremodulen und anderseits aus noch vom späteren Anwender zu erstellenden Softwarebausteinen (Objekten,die sich von den fertigen durch Vererbung ableiten oder neu erstellte Objekte) besteht.

Der Anteil der vorgegebenen Softwarebauteile ist meistens sehr viel größer als der Anteil der noch zu erstellenden Einheiten.Die vom Anwender neu erstellten Programmteile sind dabei nur im Zusammenhang mit den schon vorgegebenen Modulen lauffähig.

Bei der Aufgabenbeschreibung des Programms sind nur solche Teile zu beschreiben,die durch den späteren Benutzer der vorgegebenen Software neu zu entwickeln sind.Die Ausführung eines Programms bezieht sich nicht nur auf die vom späteren Benutzer neu entwickelten Softwaremodule sondern auf das Gesamtsystem.

Bei der Analyse und der objektorientierten Modellierung des Problems muß das Gesamtproblem in Teilprobleme zerlegt werden,die dann ihrerseits wieder zu verfeinern sind.Diese Verfeinerung bezieht sich jedoch zunächst nur auf die neu zu erstellenden Programmteile,die sich mittels der schon vorhandenen Programmteile lösen bzw. von Ihnen ableiten lassen,es sei denn es besteht die Notwendigkeit noch völlig neue Programmmodule zu entwerfen.

Welche Kriterien sollten nun einem guten Softwareentwurf zugrunde gelegt werden?

Nach Blaauw (in Lit 49,S. 199) lassen sich folgende Kriterien nennen:

„
-Vollständigkeit
Wenn ein Konzept aufgenommen wird,dann vollständig

-Allgemeinheit
Wenn ein Konzept aufgenommen wird,dann in seiner allgemeinsten Form

-Offenheit: Entwurfsfreiraum lassen

-Orthogonalität
Wenn ein Problem aufgenommen wird,dann losgelöst von anderen

-Klarheit
Wenn ein Problem aufgenommen wird,dann einsichtig und transparent

-Sicherheit:gegen natürliche und willkürliche Störungen

-Wirtschaftlichkeit:gutes Verhältnis zwischen Preis und Leistung

-Effizienz:gute Ausnützung der Mittel

-Umweltschonung:Vermeidung von Abfällen aller Art

Objektorientierte Entwurfstechniken decken die genannten Kriterien ab:

Vollständigkeit und Allgemeinheit:

Die Definition einer grundlegenden (evtl. abstrakten) Klasse,von der sich andere Klassen ableiten,wird man so gestalten,dass sie den allgemeinsten Fall abdeckt und vollständig alle dazu nötigen Methoden bereitstellt.

Offenheit:

Die Vererbungsbeziehung und die Polymorphie lassen genügend Spielraum für ein offenes,verschiedenartiges Verhalten von abgeleiteten Klassen.

Orthogonalität:

Durch das Prinzip der Datenkapselung werden Beeinflussungen der verschiedenen Objekte ausgeschlossen..

Klarheit:

Die Struktur der obersten Klassen einer Klassenbibliothek entspricht oft der Struktur eines abstrakten Datentyps und ist deshalb einfach, einsichtig und klar.

Sicherheit:

Durch die Strukturierung des Objekttypen infolge von Vererbungsbeziehungen ist eine sicherer Programmierungsansatz möglich.

Wirtschaftlichkeit, Effizienz und „Umweltschonung“
“
Durch die Wiederverwendbarkeit von Softwarebausteinen infolge objektorientierter Vererbung sind Wirtschaftlichkeit und Effizienz gegeben.Programmmodule können immer wieder verwendet werden.(Abfallvermeidung)

Die bisherigen Erörterungen zeigen,dass sich das Prinzip der Objektorientierten Programmierung in ganz besonderer Weise für den Einsatz im Unterricht eignet.

Im folgenden sollen die didaktisch-methodischen Vorteile in Kurzform noch einmal zusammengestellt werden:

1)Der objektorientierte Ansatz erleichtert und ermöglicht das Wiederverwenden von Software.Softwaremodule können dem Schüler für größere Programmierungsprojekte bereitgestellt werden.Nur so ist es im Schulunterricht möglich,größere und umfangreichere Programmprojekte wie z.B. die Darstellung von Graphen und von Graphenalgorithmen anzugehen und Elemente einer graphischen Benutzeroberfläche auf der Grundlage eines graphischen Betriebssystems in den erstellten Programmen zu verwenden.

2)Der objektorientierte Ansatz ist viel stärker als frühere Entwurfsmethoden an unserem Denken und unserer Sprache orientiert.Die Entwurfsmethode besteht deshalb nur aus einer Phase.Die grundlegenden Objekte und ihre Beziehungen entwickeln sich schon bei der Analyse des Problems als Denkstrukturen und Begriffe,die dann schrittweise konkretisiert werden und brauchen nicht in einer zweiten Entwurfsphase als Datenstrukturen und Operationen (künstlich) neu entworfen zu werden.

(Prinzip des Modellierens 1. Teil)

3)Der objektorientierte Ansatz bedeutet die Möglichkeit einer intuitiven Softwareentwicklung,in der die reale Welt in Abschnitte,die Realitätsausschnitte aufgeteilt wird, die dann durch die Nachrichten austauschenden Objekte codiert und simuliert werden.Anstatt das Gesamtsystem durch eine völlig neu zu entwerfende Aufrufstruktur verschachtelter Unterprogramme bzw. Module darstellen zu müssen,kann man intuitiv die vorgegebenen Strukturen der Realität als Objektbeziehungen darstellen.

(Prinzip des Modellierens 2. Teil)

4)Der objektorientierte Entwurf ist anschaulich,strukturiert,einfach und übersichtlich und entspricht den Kriterien für einen guten Softwareentwurf. Der Schüler wird zu strukturierten Program-und Datentechniken gezwungen.

Objektorientiertes Programmieren bedeutet Klassifizieren von Objektklassen, Zusammenfassen durch Kapseln sowie Lokalisieren von Realitätsauschnitten, die durch gleiche Objekte beschrieben werden können,Identifizieren eines Objektes,das verschiedene Zustände annehmen kann und Abstrahieren von Realitätseigenschaften eines Weltausschnitts hin zur Objektstruktur.

Durch fortlaufende weitere Abstraktion beim Durchlaufen einer Objekthierachie von den Subklassen-Objekten zu den Objekten der obersten Superklasse ergibt sich wie von selbst der Begriff des abstrakten Datentyps,der von allen realen Implementationen mit konkreten Datenfelder absieht und in dem nur noch die gegenseitige Relation der den Datentyp charakterisierenden Methoden enthalten ist.Durch Vererbung lassen sich aus ihm sofort Objekte erzeugen,die dann eine spezielle Datenimplementation enthalten.

5)Objektorientiertes Programmieren fördert projektbezogenes Arbeiten durch die durch die Struktur der Objekte vorgegebene Modularisierung und begünstigt so interaktive und kooperative Unterichtsformen.Der Schüler lernt im

Team Aufgaben zu lösen und sich bezüglich des gesetzten Ziels mit den anderen Gruppen zu koordinieren sowie die ihm gestellte Aufgabe richtig einzuordnen und im Gesamtzusammenhang zu sehen.Durch die Möglichkeit realitätsbezogene Programme auf Grund des objektorientierten Ansatzes zu erstellen, ist auch die Einbeziehung des Unterrichtsstoff anderer Fächer in ein Projekt,d.h. fächerübergreifender Unterricht und Lernen im Kontext möglich.

6)Die Struktur der Objekte bei objektorientierten Programmiersprachen entspricht der Struktur der Dinge in der Ontologie,d.h. der Ordnungsbeziehung zwischen allgemeinen Objekten der Umwelt.So erweist sich die objektorientierte Struktur als allgemeines Prinzip der realen Welt.

B III Konkrete methodisch-didaktische Bemerkungen zur Durchführung von Unterrichtsreihen zum Thema algorithmische Graphentheorie

Um den Ablauf und die Funktionsweise von Graphenalgorithmen zu verstehen, ist es am besten,die Algorithmen selber zu programmieren.Dazu muß der Graph als geeignete Datenstruktur erzeugt werden.Konventionell wird ein Graph dabei als Array in Form von Adjazenzmatrizen, Inzidenzmatrizen oder Adjazenzlisten gespeichert.Diese Form der Darstellung hat jedoch mehrere Nachteile:

1)Die Darstellungsarten abstrahieren stark von der zeichnerischen Darstellung des Graphen mit Hilfe von Knoten als Kreise und Kanten als Verbindungslinien zwischen den Knoten und sind deshalb unanschaulich.

2)Den Knoten oder Kanten zugeordnete Inhalte (z.B. Bezeichnungen) können in dieser Darstellung schwer und nur mit Hilfe von zusätzlichen Datenstrukturen untergebracht werden,insbesondere dann,wenn Knoten und Kanten mit Bewertungen (z.B. Zahlenwerten) versehen werden sollen.

3)Die Datenstrukturen eignen sich nur für eine statische Arbeitsweise mit Graphen,bei der die Graphenstruktur von vornherein fest vorgegeben ist.Bei dynamischen Anwendungen,bei denen Löschen und Einfügen von Knoten und Kanten jederzeit möglich ist,muß ein großer Teil der Daten in den Arrays neu organisiert und um- bzw. überschrieben werden.Eventuell werden dabei auch die maximal vorgegebenen Grenzen der Darstellungs-Arrays erreicht,oder es wird von vornherein zu viel Speicherplatz verschwendet.

4)Mit Hilfe der genannten Datenstrukturen ist es oft nur möglich,die Algorithmen speziell auf eine bestimmte Aufgabenstellung hin zu entwerfen.Bei einer ähnlich gelagerten neuen Problemstellung muß der gesamte Algorithmus wieder neu erstellt werden, und von der Lösung der vorigen Aufgabe kann nicht profitiert werden.

5)Der Aufbau eines Graphen mit Hilfe einer Programmiersprache ist schon alleine eine recht komplexe Angelegenheit,so dass man erst sehr viel Zeit in die Verwaltungsroutinen des Graphen,die in dem Aufbau und der Strukturierung von Adjazenzmatrizen, Inzidenzmatrizen oder Adjazenzlisten liegen,investieren muß,bevor es möglich ist,zu der Erstellung des eigentlich gewünschten Problemlösungsverfahrens zu gelangen.

Die genannten Mängel lassen sich durch den Einsatz von dynamischen Datenstrukturen (Zeigern) und durch die Verwendung von Objekten zur Darstellung von Graphen,die schon geeignete Verwaltungsmethoden für Knoten und Kanten mittels Vererbung bereitstellen,vermeiden,so dass man sich auf die Erstellung des eigentlichen Algorithmus unter Verwendung dieser Methoden konzentrieren kann.

Diese Art der Programmierung übersetzt die zeichnerische Darstellung des Graphen in eine äquivalente Datenstruktur und ist dadurch viel anschaulicher,weil durch sie das gegebene Problem viel besser in ein isomorphes Modell abgebildet werden kann.Werden noch zusätzliche Methoden zur zeichnerischen Darstellung des Modells durch die Objekte bereitgestellt,ist jederzeit wieder eine Eins zu Eins Übersetzung in die gewohnte zeichnerische Darstellung möglich,und die Ergebnisse der Lösungsverfahren bzw. der Ablauf von Algorithmen können unmittelbar anschaulich verfolgt werden.

Die Kapitel des Abschnittes C dieser Arbeit Unterrichtsskizzen setzen jeweils voraus,dass der Graph schon in der eben beschriebenen Weise vorgegeben ist,und es jetzt nur noch darum geht,den Algorithmus für das jeweilig gestellte Problem der Graphentheorie unter Einsatz von objektorientierten, vorgegebenen Methoden zu implementieren.

Wenn man nicht auf objektorientierte Programmierkenntnisse der Schüler zurückgreifen will oder kann,bietet es sich an,das (fertige) Programm Knotengraph unter Einsatz des dort zur Verfügung gestellten Demo-Modus oder Ein-

zelschrittmodus zur Veranschaulichung der entsprechenden Graphenalgorithmen einzusetzen.Dann ist es erforderlich,den in den Kapitel C Unterrichtsskizzen dargestellten Quellcode durch die dort zusätzlich vorhandenen Verbalbschreibung des Algorithmus zu ersetzen.(Dieses ist die einfachste Art algorithmische Graphentheorie anschaulich zu unterrichten,nämlich das Programm Knotengraph als Fertigversion zu benutzen.)

Durch diese Verbalbeschreibung und Demonstration der Algorithmen oder methodisch noch besser durch die im vorletzten Abschnitt genannte selbstständige objektorientierte Programmierung der Algorithmen und Erstellung eigenen Quellcodes unter Benutzung der durch Vererbung bereitgestellten Verwaltungsmethoden des Graphen kann ein vollkommener Einblick in die **mathematische** Struktur und Funktionsweise der entsprechenden Verfahren der Graphentheorie erreicht werden.Die methodisch-didaktischen Gesichtspunkte dieses Aspekts werden im zweiten Teil dieses Kapitels erörtert.

Will man jedoch darüber hinaus aber auch noch grundlegendes Verständnis in die **informationstechnische** Struktur und Funktionsweise der Verfahren erreichen,ist es sinnvoll die entsprechenden Datenstrukturen selber objektorientiert zu entwickeln und aufzubauen,statt sie wie im vorigen Absatz beschrieben,lediglich fertig zu benutzen.Diese Möglichkeit soll im folgenden zuerst dargestellt werden.

Es bietet sich deshalb hier an,den Aufbau eines Graphen als Programmierungsprojekt im Informatikunterricht als ein Musterbeispiel für die Verwendung und Handhabung von Objektdatentypen und als didaktisch-methodisches Konzept eines Einführungskurses in die objektorientierte Programmierung zu wählen.

Der Umfang der dem Programm Knotengraph auf Grund der graphischen Benutzeroberfläche zu Grunde liegenden Objektmethoden legt es dabei allerdings nahe, nur eine unbedingt notwendige Teilmenge dieser Methoden für ein solches Projekt auszuwählen,um einerseits den Blick auf das Grundlegende nicht zu verlieren und um andererseits in einem für ein Unterrichtsprojekt angemessenen zeitlichen Rahmen bleiben zu können.

Der Aufbau der den Graph beschreibenden Objekt-Datenstrukturen wird schon deutlich,wenn man sich darauf beschränkt,das Programm Knotengraph ohne graphische Benutzeroberfläche als Anwendung mit reiner Textausgabe (in dieser Arbeit auch Consolenanwendung genannt) zu implementieren.Dieses hat außer der Umfangreduzierung außerdem den Vorteil,dass die Verwendung aller ereignisorientierter Methoden und der Einsatz aller visuellen Komponenten entfällt,und ein Programm entsteht,das lediglich die grundlegenden Sprachelemente von Pascal,erweitert um die Möglichkeiten der Erzeugung und Verwendung von Objekten (Objektpascal) benutzt.

(Dabei wird Delphi(Version 1.x Textanwendung,Version ab 2.x Consolenoption) statt Objekt-Pascal benutzt,weil Delphi das bessere Objektmodell enthält.)

Als Zeitpunkt für eine solche Unterrichtsreihe bietet sich im Informatikunterricht entweder die Jahrgangsstufe 12.1 oder die Jahrgangstufe 13 an. In Jahrgangsstufe 13 kann eine solche Sequenz gemäß den Richtlinien (NRW:Komplexere Algorithmen mit allgemeinen Datentypen) als eines der verlangten Programmierungsprojekte durchgeführt werden.

In der Jahrgangsstufe 12.1 läßt sich der objektorientierte Aufbau einer Graphenstruktur entweder anschließen als konsequente Fortführung der Einführung und Besprechung der Datenstrukturen Liste, Keller, Schlange und Baum.Oder aber der objektorientierte Aufbau läßt sich noch besser als einheitliche Unterrichtsreihe mit dem Hauptziel Graphen konzipieren,die teils auf dem Weg dorthin die Strukturen Liste, Keller und Schlange als Hilfsmittel zur Verwaltung der objektorientierten Graphenstruktur einführt, teils aber,nachdem die Graphenstruktur fertig vorliegt,als Anwendungsprojekt die Baumstruktur als Spezialfall eines Graphen von einem allgemeineren Gesichtspunkt aus,als es bisher im Informatikunterricht üblich ist,behandelt (d.h. Betrachtung allgemeiner Nicht-Binär-Bäume in Graphen,Durchlaufverfahren in Bäumen,Konstruktion von Gerüsten usw.).

Eine in sich abgeschlossene Unterichtsreihe,die die zuletzt genannte Möglichkeit verwirklicht,soll im folgenden methodisch-didaktisch dargestellt werden.

Das im folgenden beschriebene Projekt wurde von mir als Unterrichtsreihe in Informatikkursen der Sekundarstufe II sowohl in Jahrgangsstufe 12.1 mit Fortsetzung in 12.2 als auch in Jahrgangsstufe 13.1 durchgeführt (teilweise beginnend auch schon in 11.2).

1)Unterrichtsvoraussetzungen

Voraussetzung für die Durchführung eines Unterrichtsprojekts mittels des Entwicklungsumgebung Knotengraph mit dem Inhalt Graphenalgorithmen sind Kenntnisse der Struktur und Programmierung von Objekten in Delphi sowie der Datenstruktur von einer (universell verwendbaren) einfachverketteten Liste aus Objekten,deren Dateninhalte und Datentypen von einem Anwender noch nachträglich festgelegt werden können.

Wenn der Informatikunterricht der Sekundarstufe II in der Jahrgangsstufe 11 gleich mit der Programmiersprache Delphi beginnt,hat das zwar den Vorteil,dass die Schüler während der gesamten Oberstufenkursfolge nicht mehr unbedingt auf eine andere Entwicklungsumgebung einer Programmiersprache umsteigen brauchen und auch von Anfang an schon gleich Programme mit ansprechenden Windowsoberflächen mit relativ wenig Aufwand erstellen können.

Die Nachteile sind jedoch,dass die Arbeitsweise dieser Programme von einem Anfänger nicht unbedingt durchschaut werden können,weil in den Programmen schon gleich das Prinzip der Programmierung mittels Objekten verwendet wird,indem die Entwicklungsumgebung selber beim visuellen Konstruieren einer Programm-Windows-Oberfläche durch Ziehen entsprechender Oberflächenelemente (Komponenten) wie Buttons, Ausgabelabels, Editierfenster usw. in die Ausgangsform automatisch Objekt-Programmcode erzeugt.(Das gleiche gilt für entsprechende visuelle Entwicklungsumgebungen mit Java oder C++.)

Außerdem wird das Prinzip der Ereignissteuerung der Elemente benutzt,so dass die erstellten Oberflächenelemente mittels ihrer Ereignismethoden auf die verschiedenen Windowsnachrichten reagieren.

Der Mechanismus,nach dem dies geschieht,bleibt für den programmierenden Schüler vollkommen verborgen,weil kein Hauptprogramm geschrieben wird,sondern die Ablaufsteuerung vom Windows-Betriebssystem ausgeführt wird.

Die neuen Richtlinien Informatik für NRW (Lit 67),die im Jahre 1999 für dann in Jahrgangsstufe 11 beginnende Kurse in Kraft treten,sehen u.a. als eine von mehreren unterschiedlichen Wahlmöglichkeiten unter dem Stichwort „Sequenz objektorientiert- visuell“ eine solchen Einstieg in die Entwicklungsumgebung Delphi in der Jahrgangsstufe 11 vor.Die Reflexion darüber,was eigentlich Objekte sind und nach welchen Regeln und Zusammenhängen in der Jahrgangsstufe 11 programmiert wurde,soll dann der Jahrgangsstufe 12 vorbehalten bleiben,in der der von Delphi automatisch bereitgestellte Quellcode analysiert oder manuell selbst erzeugt bzw. abgeändert wird,und so die Prinzipien der objektorientierten Programmierung systematisch besprochen werden.(Nach der Intention dieser Arbeit sollte die Erläuterung dieser Prinzipien dann dort am besten durch Aufbau einer Graphenstruktur und Anwendungen der Struktur an Hand von mathematischen Algorithmen der Graphentheorie geschehen.)

Der Vorteil dieses Einstiegskonzepts liegt darin,dass schon von Anfang an unter Windows in der Oberflächengestaltung relativ aufwändige Oberflächengestaltete Programme erzeugt werden können.Mit deren Hilfe muss dann allerdings zunächst eine Einführung in die nicht-objektorientierten, imperativen Fähigkeiten von Delphi gegeben werden,die denen einer Einführung in Turbo-Pascal,wie sie bisher in der Jahrgangsstufe 11 üblich waren,entsprechen.Diese elementaren Kenntnisse sind dann die Voraussetzun-

gen,um in der Stufe 12 überhaupt den objektorientierten Ansatz verstehen zu können (und lassen sich zunächst einfacher ohne den zusätzlichen Ballast des Umgangs mit Objekten vermitteln:s.u.).

Der didaktische Nachteil liegt,wie gesagt,darin,dass zunächst nur Kochrezepte für den Umgang bezüglich des von Delphi automatisch erzeugten Objekt-Quellcodes vermittelt werden,der bis zur Jahrgangsstufe 12 für den Schüler zumindestens teilweise unverständlich bleibt.

Will man also nicht nur oberflächliche Gebrauchsanleitungen zur Erstellung von Programmen vermitteln,sondern von Anfang an Einsicht in die grundlegenden Arbeitsweise und die Ablaufsteuerung von Programmen erreichen,kann es sinnvoll sein,zunächst nicht mit den komplexen visuellen und objektorientierten Möglichkeiten von Delphi zu beginnen.

Um den Umstieg auf ein völlig andere Syntax später zu vermeiden,empfiehlt es sich daher dann zunächst entweder mit dem einfacheren Turbo-Pascal-Programmsystem (z.B. die letzte Version 7) zu beginnen oder aber noch besser die Möglichkeit zu nutzen mittels der Version 1.x von Delphi,textorientierte Programmanwendungen zu erstellen,die in einem einzigen Fenster ablaufen und somit Turbo-Pascal äquivalent sind.Durch Einsatz der letzten Möglichkeit entfällt auch der Wechsel der Entwicklungsumgebung,sondern sie wird lediglich in ihren Möglichkeiten später erweitert.

Ein solcher Kursus ist ebenfalls als andere Wahlmöglichkeiten in den neuen Richtlinien Informatik (Lit 67) unter dem Stichwort „Sequenz imperativ" vorgesehen,der die Grundlagen der imperativen Programmierung vermittelt.

Mit Hilfe kleinerer Übungsprogramme können hier zunächst imperativ in der Jahrgangsstufe 11 die grundlegenden Kontrollstrukturen,das Proceduren-Konzept mit Paramterübergabe,die Rekursion,die einfachen Datentypen sowie komplexere Datenstrukturen sowie die Ablaufsteuerung durch ein Hauptprogramm vermittelt werden.Zur Einübung dieser Techniken sind im solche Quellcodeumfang reduzierte Programme ohne Belastung durch zusätzliche Objektstrukturen für Anfänger am besten geeignet.

Die Sequenz sieht außerdem für die Jahrgangsstufen 12 und 13 auch „eine vertiefte Auseinandersetzung mit allgemeineren und damit auch komplexeren Baumstrukturen bis hin zu allgemeinen Graphen" vor (Lit 67,S. 60),die allerdings „auch ans Ende und damit in die letzte Spiralwindung des Lehrplans platziert werden" kann.Der Plazierung des Themas Graphen an das Ende der Kursfolge soll mit dieser Arbeit entschieden widersprochen werden.Stattdessen soll (u.a. in diesem Kapitel) aufgezeigt werden,dass die algorithmische Graphentheorie besonders geeignet ist,objektorientierte Programmierungstechniken zu vermitteln,so dass die Beschäftigung mit Graphen in den Mittelpunkt einer Kursfolge Informatik in der Sekundarstufe II gesetzt werden sollte.

Der Nachteil der rein imperativen Sequenz der Richtlinien ist,dass in dieser Kursfolge auch in den Jahrgangsstufen 12 und 13 nicht zwingend die Behandlung von Objekten wohl aber des abstrakten Datentyps ADT vorgesehen ist.Dabei sind Objekte am besten geeignet abstrakte Datentypen zu modellieren.Die Entwicklung einer so komplexen Oberfläche,wie sie zur graphischen Darstellung eines Graphen sowie zur Demonstration von Graphenalgorithmen nötig ist,ist außerdem mit rein imperativer Programmierung für Schüler wegen des zeitlichen Aufwandes nicht zu erreichen.

Eine dritte Wahlmöglichkeit der Sequenzbildung der neuen Richtlinien heißt „Sequenz objektorientiert allgemein" und sieht vor,eine Einführung in die objektorientierte Programmierung unter Erzeugung eigener Objektklassen ohne die zwingende Benutzung einer visuellen Entwicklungsumgebung durchzuführen.Dabei soll auch zwingend der Umgang mit Zeigern („dynamische Verkettungsstruktur",Lit 67,S. 48) vermittelt werden,der im der Konzeption objektorientiert visuell fehlt,worauf aber zum tieferen Verständnis des Umgangs mit Objekt-Listen keinesfalls verzichtet werden kann.Der Vorteil dieses Ansatzes liegt darin,dass die Programmierung mit Objekten und auch die Ab-

laufsteuerung durch ein Hauptprogramm vom Schüler von Anfang an durchschaut werden kann.Der Nachteil liegt darin,dass die Programme nicht in ein graphisch orientiertes Betriebssystem wie Windows eingebunden sind bzw. nicht dessen Möglichkeiten ausnutzen.Graphische Objekte,wie sie von der Komponentenbibliothek von Delphi (Sequenz objektorientiert visuell) schon vorgegeben werden,sind von Schülern mit eigenen Mittel kaum zu realisieren und müssen bereitgestellt werden.(Eventuell müßten diese Möglichkeiten vom Lehrer dann in Form von entsprechenden Units bzw. DLL`s zur Verfügung gestellt werden,so dass man dann auch gleich die von Delphi bereitgestellten Mittel benutzen kann.)

Es liegt daher nahe eine Unterrichtssequenz als Kombination der drei genannten Sequenzen zu konstruieren und zu versuchen,die didaktischen Vorteile zu erhalten und die Nachteile zu vermeiden.Da in den Richtlinien außerdem ein Paradigmenwechsel während des Gesamtkurses vorgeschrieben ist,entspricht diese Kombination von imperativer und objektorientierter Programmierung genau den Vorgaben.

Außerdem soll dabei der Begriff des Graphen sowohl als (zu erzeugende) Datenstruktur als auch als Struktur,mit deren Hilfe Anwendungen (algorithmische Graphentheorie) erzeugt werden können (Anwendungsstruktur),im Mittelpunkt stehen.Um beides zusammen zu erreichen,ist der Begriff des Objektes das geeignete Werkzeug,denn durch ihn wird gerade die Datenstruktur mit den geeigneten Methoden ihrer Anwendungen verknüpft und gekapselt.Gemäß der Konzeption Objektorientiert-Allgemein wird dazu ohne Verwendung von den durch Delphi vorgegebenen visuellen Komponenten zunächst eine Objekt-Listentruktur mit nachträglich vom Anwender zu definierenden Dateninhalten und Datentypen mit Hilfe des Zeigerkonzepts erzeugt und als textorientierte Anwendung mit Hauptprogramm als Steuerung realisiert.So kann der Umgang mit Objekten gemäß dem didaktischen Prinzip der Isolierung von Schwierigkeiten am besten vermittelt werden.Ausgehend von dieser Grundlage als Einheit wird dann die Graphenstruktur objektorientiert aufgebaut (ebenfalls zunächst als Textanwendung und danach als Erweiterung objektorientiert-visuell mit ansprechender graphischer Oberfläche:siehe die Abschnitte 3 bis 6 dieses Kapitels).

Die Wichtigkeit des Lernens im Kontext der Anwendung wird in den Richtlinien NRW (Lit 67,S. 16 u. 17) besonders hervorgehoben:"Die Hereinnahme konkreter informatischer Anwendungen in das Unterrichtsgeschehen ermöglicht es einerseits,die Schülerinnen und Schüler auch längerfristig für eine Lernsequenz zu motivieren,und andererseits,Zusammenhänge,Gesetzmäßigkeiten und Theorien an konkreten Objekten,Systemen bzw. Prozessen zu veranschaulichen.

Fachinhalte unter den Aspekten Modellieren und Konstruieren sowie Analysieren und Bewerten heißt unverzichtbar,Lernende mit Anwendungssituationen zu konfrontieren,um entweder schon bekannte Methoden und Kenntnisse zur modellhaften Bewältigung des Problems anzuwenden oder durch Abstraktion,Verallgemeinerung und Systematisierung zu diesen Methoden zu gelangen.Schülerinnen und Schüler können nur in realen bzw. modellhaften nachempfundenen Anwendungssituationen eine Informatik-Kompetenz erwerben,d.h. informatorische Probleme durchschauen,systematisch lösen und ihren Anwendungs- und Verwertungszusammenhängen richtig beurteilen."

Wie in dieser Arbeit gezeigt wird,liefert gerade die Beschäftigung mit algorithmischer Graphentheorie eine Fülle von Anwendungsproblemen,die teilweise direkt der realen Welt entnommen werden können und deren Modellierung in eine Graphenstruktur unmittelbar einsichtig ist,sowie intuitiv leicht gefunden werden kann.Deshalb ist das Thema algorithmische Graphentheorie besonders geeignet,um sowohl dieser Forderung nach Anwendungsbezug als auch dem Aspekt des Arbeitens mit objektorierten Strukturen gerecht zu werden.

Der Einstieg in die Programmierung sollte,wie oben vorgeschlagen,in der Jahrgangsstufe 11 als Kombination der verschiedenen Sequenzen der Richtlinien vorgeschlagen,zunächst an Hand kleinerer Programme imperativ erfolgen (imperativer Ansatz),um Unterrichtsinhalte wie die Ablaufsteuerung durch Kontrollstrukturen sowie das Konzept des Hauptprogramms vermitteln zu können und nicht durch unnötige Probleme objektorientierten Programmierens be-

lasten zu müssen.Wenn man will,jedoch nicht unbedingt notwendigerweise,kann sich an die imperative Einführung dann noch zusätzlich eine Unterrichtsphase anschließen,in denen die Möglichkeiten mit Delphi visuell und objektorientiert zu programmieren kochrezeptartig vermittelt werden.Auf diese Weise ist der Umgang mit Methoden und Objekten dann schon syntaxmäßig vertraut ist,und die Schüler kennen den Umgang mit der visuellen Komponentenbibliothek sowie Ereignismethoden.Man sollte aber die Schüler darauf hinweisen,dass eine Analyse dieser Art der Programmierung später erfolgen wird.

Eventuell können diese Einstiegssequenzen schon,wie in den Richtlinien vorgesehen,unter Kürzung und Weglassen von bisher üblichen Inhalten wie den Sortierverfahren und Datenstrukturen wie Record und Set sowie selbstdefinierten Datentypen (eventuell sogar des Arrays) schon vorzeitig während oder zu Beginn des 2. Halbjahres 11.2 abgeschlossen sein,so dass mit dem im folgenden beschreibenden Inhalten früher als zu Beginn der Jahrgangsstufe 12.1 begonnen werden kann.

Ein Einstieg in den systematischen,analytischen Umgang mit Objekten mittels der Entwicklungsumgebung von Delphi sollte dann spätestens zu Beginn der Jahrgangsstufe 12.1 (bzw. besser noch früher d.h. in der Jahrgangsstufe 11.2) an Hand eines anschaulichen Beispiels erfolgen.

Dieses Beispiel (siehe Abschnitt 2) sollte gemäß dem objektorientierten visuellen Ansatz bezüglich den für die Jahrgangsstufe 12 vorgesehenen Lerninhalten einerseits die Aufgabe haben systematisch-fundiert in die Grundzüge objektorientierten Programmierens,als auch (falls noch nicht aus Jahrgangsstufe 11 bekannt) in die unter dem Betriebssystem Windows wichtige Verwendung ereignisgesteuerter Komponenten unter Benutzung ihrer Ereignismethoden einzuführen.An Hand dieses Beispiels kann bei schon bisheriger Verwendung der Entwicklungsumgebung die (schon) bisher (gegebenenfalls) kochrezeptartig Verwendung von Objekten analysiert werden.Die Fortführung dieses Unterrichtsabschnitts erfolgt dann,wie schon oben dargestellt,nach der Konzeption Objektorientiert-Allgemein durch die Entwicklung einer Objekt-Listenstruktur auf der Grundlage des Zeigerkonzepts als Textanwendung (siehe dazu die Abschnitte 3 und 4 dieses Kapitels).

Günstig wäre es,wenn durch dieses Einführungsbeispiel als zusätzliches Ziel noch ein erster Einblick in den Aufbau einer Graphenstruktur sowie deren zeichnerische Darstellung gewonnen werden könnte.

Im folgenden wird ein Einstiegsprojekt der genannten Art beschrieben,wobei die wichtigsten Elemente des Programms jeweils kurz skizziert werden.Der komplette Quellcode des Programm kann dem Anhang entnommen werden und ist auf der Installations-CD vorhanden.

(Die hier im folgenden dargestellte Unterrichtsreihe kann mit kleineren Anpassungen aber auch nach Durchführung jeder Sequenz,die sich in Jahrgangsstufe 11 streng an die jeweiligen Vorgaben der Richtlinien der reinen Sequenzen Imperativ,Objektoriert-visuell oder Objektorientiert-allgemein hält und nicht diese Methoden kombiniert,wie in dieser Arbeit favorisiert,in Jahrgangsstufe 12 begonnen werden.)

2) Einführung in die Programmierung mit Objekten mittels einer Graphenstruktur

Ausgehend von der zeichnerischen Darstellung eines Graphen, wobei die Knoten als Kreise und die Knotenbezeichner als Beschriftung der Kreise in ihrem Mittelpunkt dargestellt werden,ergibt sich der Objektbegriff und die Vererbungsbeziehung zwischen Objekten fast von selbst, wenn man die Objekte „Mittelpunkt“ (Punkt) in dem der Knoten gezeichnet wird, “Knoten ohne Bezeichner“ und „Knoten mit Bezeichner“ betrachtet.Jedes Objekt ergibt sich aus dem vorigen durch Hinzufügen einer weiteren Eigenschaft.

Es ist sofort klar,dass für die Darstellung eines (Mittel-) Punktes auf dem Bildschirm die Mittelpunktskoordinaten X und Y wichtig sind.Ausgehend von

dieser Grundeigenschaft entsteht die zeichnerische Darstellung eines Knotens (als Kreis) durch das Hinzufügen der zusätzlichen Eigenschaft Radius.Die Erweiterung vom unbezeichneten zum bezeichneten Knoten erfolgt dann durch das Hinzufügen eines Bezeichners (Inhalt).

Zwischen den weiteren Bestandteilen eines Graphen nämlich den unbezeichneten Kanten bzw. den Kanten mit Bezeichnern besteht die letztgenannte Verwandschaftsbeziehung in gleicher Weise.

Jedoch ist die zeichnerische Darstellung der Kante als Verbindungsstrecke zwischen zwei Knotenmittelpunkten nicht so leicht unter Hinzufügen einer oder mehrerer weiterer Eigenschaften aus den im vorletzten Absatz definierten Objekten abzuleiten, weil ihre wichtige Lageeigenschaft von zwei schon bereits definierten Objekte,nämlich den beiden Randknoten bestimmt wird, und deshalb nicht aus einem einzigen dieser Objekte hervorgehen kann.

Anscheinend ist also eine Kante als Zusammenfassung von zwei schon vorhandenen Knotenobjekten (Instanzen) festzulegen. Dies gibt Anlass neben der direkten Vererbungsbeziehung zwischen Objekten,die auch „Ist"-Beziehung genannt wird (jeder „Inhaltsknoten" ist ein Knoten/der Satz jeder Knoten ist ein Punkt passt hier umgangssprachlich nicht besonders gut) auch die sogenannte „hat"-Beziehung (eine Kante hat zwei Randknoten) zu betrachten,mit der Objekte ineinander verschachtelt werden können (Objekte als Datenfelder eines anderen Objekts).

Betrachtet man nun den Begriff des Graphen als weitere übergeordnete Struktur,ist klar,dass hier eine analoge Beziehung zu konstruieren ist.Denn ein Graph besteht gerade aus einer Ansammlung der beiden Objektarten Inhaltsknoten und Inhaltskanten (bzw. Knoten und Kanten),so dass es auch hier wieder sinnvoll ist, die „hat"-Beziehung anzuwenden.(Ein Graph hat Knoten und Kanten.)Zur Speicherung einer Menge von Knoten und Kanten werden Arrays benutzt,da die Datenstruktur der Liste zu diesem Zeitpunkt (folgt als nächster Schritt,siehe nächster Abschnitt) noch nicht als bekannt vorausgesetzt wird.)

Gibt es jedoch keinerlei Eigenschaften,die allen bisher genannten (Einzel-)Objekten Punkt, Knoten, Inhaltsknoten, Kante, Inhaltskante und Graph gemeinsam sind?

Die Beantwortung dieser Frage führt auf den allgemeinen Begriff der Figur.Allen Objekten ist nämlich gemeinsam, daß sie eine Figur in einer bestimmten Farbe auf ein und derselben Zeichenoberfläche sind.Also sollte jedem der Objekte die Eigenschaften Farbe, mit der es gezeichnet wird, und die Eigenschaft Oberfläche auf der es dargestellt wird, zugeordnet werden.

Sinnvoll ist also,als grundlegendes Objekt,von dem sich alle anderen ableiten lassen,das Objekt Figur zu wählen.Den Objekten Punkt, Knoten und Inhaltsknoten müssen dann zusätzlich,wie oben dargestellt nacheinander die weiteren Eigenschaften X-/Y-Koordinaten,Radius und Bezeichner hinzugefügt werden.Der Übergang von der Kante - einerseits als Nachfolger von Figur und andererseits als zusammengesetztes Objekt aus zwei Knoten (bzw. Inhaltsknoten) - zur Inhaltskante erfolgt ebenfalls durch Erweiterung um die Eigenschaft des Bezeichners.

Bei näherer Betrachtung unter dem Aspekt der Programmierung ergibt sich,dass es notwendig ist,die Eigenschaften der Objekte in Datenfeldern entsprechender Datentypen zu speichern.Bedenkt man dabei, dass die (Lage-) Eigenschaft einer Kante durch die Randknoten bestimmt wird,ist es nützlich das Objekt Knoten ebenfalls als Datentyp aufzufassen und als Feld des Objekts Kante zu implementieren.Daraus folgt, dass es sinnvoll ist, Objekte allgemein als Datentypen festzulegen, wobei ein Objekt mit bestimmten Eigenschaften (z.B. ein bestimmter Inhaltsknoten mit bestimmten Koordinaten,mit bestimmtem Radius und bestimmtem Bezeichner) als Variable dieses Datentyps (mit festgelegten Feldern) anzusehen ist,d.h. der Begriff der Instanz eines Objekts ergibt sich zwanglos aus dieser Überlegung.

Um einen Graphen zu erzeugen,ist es beispielsweise nötig,Knoten- und Kanten-Instanzen in den Graphen einfügen zu können oder auch wieder löschen zu können.Eine Kante soll wiederum zwischen zwei speziellen Knoteninstanzen eingefügt werden.

Also muß es möglich sein, dass Instanzen von Objekten untereinander in Beziehung treten können.

Daraus ergibt sich die dritte Beziehung,die Objekte besitzen können, nämlich die „kennt“-Beziehung.Als Realisierungsart in Delphi/Pascal bietet sich die Verwendung von (als Strukturierungsmöglichkeit den Schülern schon bekannten) Prozeduren (evtl. mit Parametern) an,die allerdings hier den Objekten zugeordnet werden müssen,wodurch sich der Begriff der Methode als Botschaftsübermittlungsart zwischen den verschiedenen Objektinstanzen ergibt.Als die grundlegendsten Methoden eines Objekts sind dabei zunächst einmal das Erzeugen einer Instanz des Objektes (Constructor Create) und das Löschen des Objektes (Destructor Free) aus dem Speicher zu implementieren.

An dieser Stelle ist es dann im Unterricht wesentlich, das Prinzip der Datenkapselung sowie das Prinzip der Zusammenfassung von Daten und Methoden (Unterschiede zu Proceduren) in einem Objekt eingehend zu besprechen.Die Objekte bilden jeweils eine selbstständige Einheit, so dass das Objekt Graph die Objekte Knoten und Kanten nur durch Botschaftsaustausch d.h. durch die von Ihnen bereitgestellten Methoden verändern kann.

Unter Berücksichtigung dieser Grundsätze ergibt sich dann fast zwangsläufig die Art der den Objekten zuzuordnenden Methoden:

Methoden zum jeweils schreibenden und lesenden Zugriff auf die Datenfelder (in späteren Programmen auch als Property-Methoden realisiert),Zeichnen und Löschen des jeweiligen Objekts auf der Zeichenfläche.

Bei dem Objekt Punkt und seinen Nachfolgern ist außerdem eine Methode zur Positionsänderung des Objekt auf der Zeichenoberfläche sinnvoll.

Dies führt auf folgende Objekttypdeklarationen,die natürlich im Unterricht nicht auf einmal vorgegeben,sondern schrittweise auseinander entwickelt werden sollten (siehe dazu den entsprechenden Unterrichtsplan im Anhang):

```
type

  TFigur=class(TObject)
  private
    Zeichenflaeche_:TCanvas;
    Farbe_:TColor;
  public
    constructor Create;
    function Zeichenflaeche:TCanvas;
    function Farbe:TColor;
    procedure NeueFarbe(F:TColor);
    procedure Zeichnen;virtual;abstract;
    procedure Loeschen;virtual;abstract;
  end;

  TPunkt=class(TFigur)
  private
    X_,Y_:Integer;
  public
    constructor Create(Xk,Yk:Integer);
    function X:Integer;
    procedure NeuesX(Xk:Integer);
    function Y:Integer;
    procedure NeuesY(Yk:Integer);
    procedure Zeichnen;override;
    procedure Loeschen;override;
    procedure NeuePosition(Xneu,Yneu:Integer);
  end;


  TKnoten=class(TPunkt)
  private
    Radius_:Integer;
  public
```

```
    constructor Create(Xk,Yk:Integer;Rd:Integer);
    function Radius:Integer;
    procedure NeuerRadius(Rd:Integer);
    procedure Zeichnen;override;
    procedure Loeschen;override;
    {procedure NeuePosition(Xneu,Yneu:Integer); ****}
   end;

  TInhaltsknoten=class(TKnoten)
  private
    Inhalt_:string;
    function Wertlesen:string;
    procedure Wertschreiben(S:string);
  public
    constructor Create(Xk,Yk:Integer;Rd:Integer;Ih:string);
    property Wert.string read Wertlesen write Wertschreiben;
    procedure Zeichnen;override;
    procedure Loeschen;override;
    {procedure NeuePosition(Xneu,Yneu:Integer); ****}
  end;

  TKante=class(TFigur)
  private
    Anfangsknoten_,Endknoten_:TKnoten;
  public
    constructor Create(AKno,EKno:TKnoten);
    procedure Freeall;
    function Anfangsknoten:TKnoten;
    function Endknoten:TKnoten;
    procedure Zeichnen;
    procedure Loeschen;
  end;

  TInhaltskante=class(TKante)
  private
    Inhalt_:string;
    function Wertlesen:string;
    procedure Wertschreiben(S:string);
  public
    constructor Create(AKno,EKno:TKnoten;Ih:string);
    property Wert:string read Wertlesen write Wertschtreiben;
    procedure Zeichnen;
    procedure Loeschen;
  end;

  TGraph=Class(TFigur)
  private
    Knotenliste_:Array[1..Max] of TInhaltsKnoten;
    Kantenliste_:Array[1..Max] of TInhaltsKante;
    Knotenindex_:Integer;
    Kantenindex_:Integer;
   public
    Constructor Create;
    procedure Freeall;
    function GraphKnoten(Kno:TKnoten):TInhaltsKnoten;
    procedure KnotenEinfuegen(Kno:TInhaltsknoten);
    procedure KanteEinfuegen(Ka:TInhaltsKante);
    procedure Knotenloeschen(Kno:TKnoten); {Zusatz}
    procedure Zeichnen;
    procedure Loeschen;
  end;

var Oberflaeche:TCanvas;
```

Man wird zunächst im Unterricht nur die Objekte TFigur und TPunkt definieren,um die Syntax,das Prinzip der Vererbung und das Schreiben von Methoden einzuüben (vgl. dazu den entsprechenden Unterrichtplan im Anhang).

```
constructor TFigur.Create;
begin
  inherited create;
  Zeichenflaeche_:=Oberflaeche;
  Farbe_:=clblack;
  Zeichenflaeche_.Pen.Color:=Farbe_;
  Zeichenflaeche_.Font.Color:=Farbe_;
end;

function TFigur.Zeichenflaeche:TCanvas;
begin
  Zeichenflaeche:=Zeichenflaeche_;
end;
```

```
function TFigur.Farbe:TColor;
begin
  Farbe:=Farbe_;
end;

procedure TFigur.NeueFarbe(F:TColor);
begin
  Farbe_:=F;
  Zeichenflaeche_.Pen.Color:=F;
  Zeichenflaeche_.Font.Color:=F;
end;

constructor TPunkt.Create(Xk,Yk:Integer);
begin
  inherited Create;
  X_:=Xk;
  Y_:=Yk;
end;

function TPunkt.X:Integer;
begin
  X:=X_;
end;

procedure TPunkt.NeuesX(Xk:Integer);
begin
  X_:=Xk;
end;

function TPunkt.Y:Integer;
begin
  Y:=Y_;
end;

procedure TPunkt.NeuesY(Yk:Integer);
begin
  Y_:=Yk;
end;

procedure TPunkt.Zeichnen;
var HilfFarbe:TColor;
begin
  Hilffarbe:=Farbe;
  NeueFarbe(clwhite);
  Zeichenflaeche.Moveto(X-1,Y-1);
  NeueFarbe(clblack);
  Zeichenflaeche.Lineto(X,Y);
  NeueFarbe(HilfFarbe);
end;

procedure TPunkt.Loeschen;
begin
  NeueFarbe(clwhite);
  Zeichenflaeche_.Moveto(X-1,Y-1);
  Zeichenflaeche.Lineto(X+1,Y+1);
end;

procedure TPunkt.NeuePosition(Xneu,Yneu:Integer);
begin
  Loeschen;
  NeuesX(XNeu);
  NeuesY(Yneu);
  Zeichnen;
end;
```

Das nächste Lernziel ist dann das Umgehen mit ereignisgesteuerten Methoden sowie das Erstellen einer einfachen Benutzeroberfläche mit Hilfe von Delphi.

Die Benutzeroberfläche enthält dabei zunächst nur die Menüpunkte „Ende“ und „Punkt zeichen“,wobei der Punkt mittels der Ereignismethode FormMouseDown auf der Zeichenoberfläche per Mausklick verschoben werden kann.In den Methoden TKnotenformular.Create bzw. TKnotenformular.Destroy wird die Constructor -bzw. die Destructormethode aufgerufen.Um den Quelltext für die Einführung in die Programmierung mit Objekten nicht zu kompliziert zu gestalten,wird die Zeichenfläche(_) durch die globale Variable Oberflaeche übergeben,statt sie als Parameter des Constructors zu benutzen.

```
var
  Knotenformular: TKnotenformular;
  Objekt:TObject;
  Punkt:TPunkt;

procedure TKnotenFormular.EndeClick(Sender: TObject);
begin
  Close
end;

procedure TKnotenformular.PunktezeichnenClick(Sender: TObject);
begin
  Objekt:=Punkt;
  Punkt.zeichnen;
end;

procedure TKnotenformular.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  {Punkte zeichnen:}
  if Objekt=Punkt then  Punkt.NeuePosition(X,Y);
end;

procedure TKnotenformular.FormCreate(Sender: TObject);
begin
  {Alle Anwendungen:}
  Oberflaeche:=Knotenformular.Canvas;
  Objekt:=nil;
  {Punkt zeichen:}
  Punkt:=TPunkt.Create(320,240);
 end;

procedure TKnotenformular.FormDestroy(Sender: TObject);
begin
  {Punkt zeichnen:}
  Punkt.Free;
  Punkt:=nil;
end;
```

Die bisher gewonnenen Erkenntnisse über Objekte gestatten es auch den Objekt-Quellcode,den das Delphi-Entwicklungssystem von sich aus automatisch erzeugt,einer Betrachtung zu unterziehen, und dabei zu verstehen,was der Quellcode bedeutet. Bei der Erstellung der Anwendungs-Unit UGraph,die die Objektmethoden aufruft und benutzt,lernt der Schüler gleichzeitig das Zeichnen auf Oberflächen,das Verwenden von Ereignisprozeduren,den Umgang mit Komponenten und Formen sowie das Setzen von Properties,und außerdem natürlich den Umgang mit der Delphi-Entwicklungsumgebung kennen:

```
type
  TKnotenformular = class(TForm)
    MainMenu: TMainMenu;
    Ende: TMenuItem;
    Punktezeichnen: TMenuItem;
    procedure EndeClick(Sender: TObject);
    procedure PunktezeichnenClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);

  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

var
  {Alle Anwendungen:}
  Knotenformular: TKnotenformular;
```

Wie man sieht ist die Oberfläche TKnotenformular ebenfalls ein Objekt,das sich von TForm ableitet und die benutzten Ereignis-bzw. Menümethoden sind die Methoden dieses Objekts.Als Datenfelder treten gerade die festgelegten Menüeinträge (sowie allgemein evtl. weitere ausgewählte Komponenten) auf.Also sind Komponenten und Formen in Delphi Objekte,die ebenfalls ihre

Eigenschaften und Methoden an Nachfolgerobjekte (die durch Ziehen mit der Maus aus einer Komponentenauswahl auf die jeweilige Oberflächenkomponente oder Form erzeugt werden) vererben können (Prinzip der visuellen Vererbung).

Das Hauptprogramm sieht bei allen ereignisgesteuerten Delphi-Programmen bis auf die gewählten Bezeichner (fast) immer gleich aus und wird von Delphi selber erzeugt:

```
program Project1;

uses
  Forms,
  UForm in 'UForm.pas' {Knotenformular},
  UGraph in 'UGraph.pas';

{$R *.RES}

begin
  Application.CreateForm(TKnotenformular,Knotenformular);
  Application.Run;
end.
```

Beim der Positionsänderung von Punkten ist wegen der Kleinheit der Punkte nicht besonders viel auf der Zeichenfläche zusehen.Dies wird anders bei der Positionsänderung eines Knoten.

Dazu werden gemäß den obigen Überlegungen jetzt die Objekte TKnoten und TInhaltsknoten (einschließlich der entsprechenden Methoden) hinzugefügt:

```
TKnoten=class(TPunkt)
  private
    Radius_:Integer;
  public
    constructor Create(Xk,Yk:Integer;Rd:Integer);
    function Radius:Integer;
    procedure NeuerRadius(Rd:Integer);
    procedure Zeichnen;override;
    procedure Loeschen;override;
    {procedure NeuePosition(Xneu,Yneu:Integer); ****}
   end;

  TInhaltsknoten=class(TKnoten)
  private
    Inhalt_:string;
    function Wertlesen:string;
    procedure Wertschreiben(S:string);
  public
    constructor Create(Xk,Yk:Integer;Rd:Integer;Ih:string);
    property Wert read Wertlesen write Wertschreiben;
    procedure Zeichnen;override;
    procedure Loeschen;override;
    {procedure NeuePosition(Xneu,Yneu:Integer); ****}
  end;
```

Dabei kann jetzt der Begriff der Property als abgekürzte Schreibweise für eine Wertzuweisung mittels des Aufrufen der entsprechenden read und write-Methoden Wertlesen und Wertschreiben als Zugriff auf das Feld Inhalt_ durch Wert eingeführt werden.Bei den Objekttypen TFigur,TPunkt und TKnoten sollte dies für die Felder X_,Y_ und Radius noch nicht erfolgen,um überhaupt erstmal das Prinzip des ausschließlichen Zugriffs auf die Datenfelder des Objekts über Methoden (Datenkapselung) deutlich zu machen.

```
function TInhaltsknoten.Wertlesen:string;
begin
  Wertlesen:=Inhalt_;
end;


procedure TInhaltsknoten.Wertschreiben(S:string);
begin
  Inhalt_:=S;
end;
```

Das Beispiel eignet sich auch besonders gut dazu,den Begriff der virtuellen Methode zu veranschaulichen.Deklariert man zunächst alle Methoden als statisch,sind die mit **** gekennzeichneten zusätzlichen Methoden notwendig,da das Zeichnen der Figuren sonst zu falschen Ergebnissen führt.Die Methoden haben dabei alle denselben Quelltext,so dass es sinnvoll ist,diese Methode nur einmal schreiben zu müssen:

```
procedure TPunkt/TKnoten/TInhaltsknoten.NeuePosition(Xneu,Yneu:Integer);
begin
  Loeschen;
  NeuesX(XNeu);
  NeuesY(Yneu);
  Zeichnen;
end;
```

Bei virtueller Deklaration der Methoden Zeichnen und Loeschen ist nur die Methode NeuePosition von TPunkt nötig,die automatisch jeweils die Zeichne- und Loeschen-Methode der Nachfolgerobjekte aufruft.Man sollte die Schüler ausprobieren lassen,dass bei Fehlen der virtuellen Deklaration der Methode NeuePosition (ohne die Methoden ****) tatsächlich nur Punkte verschoben werden.Es bietet sich an,im Unterricht alle drei Programmvarianten (statisch ohne ****- Methoden,statisch mit ****-Methoden und virtual ohne ****-Methoden) auszuprobieren,um das Prinzip der virtuellen Vererbung zu verdeutlichen.

Da Zeichnen und Loeschen schon Operationen einer Figur (des Objekts TFigur) sein könnten,bei diesem Objekt aber,da zu allgemein und ohne Gestalt,noch keine konkrete Zeichnen-oder Loeschen-Methode geschrieben werden kann,bietet es sich an die Methoden bei diesem Objekt zwar beginnen zu lassen ohne sie ausführen zu können.Dies bedeutet sie dort als abstract zu deklarieren.Das heißt,dass kein Quellcode geschrieben werden braucht.

Für die Zeichnen-und Loeschen-Methoden von TKnoten und TInhaltsknoten sind einige Delphi-spezifische Zeichenbefehle nötig:

```
procedure TKnoten.Zeichnen;
var HilfFarbe:TColor;
begin
  HilfFarbe:=Farbe;
  NeueFarbe(clblack);
  Zeichenflaeche.Ellipse(X-Radius,Y-Radius,X+Radius,Y+Radius);
  NeueFarbe(HilfFarbe);
end;


procedure TKnoten.Loeschen;
var HilfFarbe:TColor;
begin
  HilfFarbe:=Farbe;
  NeueFarbe(clwhite);
  Zeichenflaeche.Ellipse(X-Radius,Y-Radius,X+Radius,Y+Radius);
  NeueFarbe(HilfFarbe);
end;

procedure TInhaltsknoten.Zeichnen;
var HilfFarbe:TColor;
begin
 HilfFarbe:=Farbe;
 inherited Zeichnen;
 NeueFarbe(clblack);
 Zeichenflaeche.Textout(X-8*length(Inhalt_) Div 2-Radius Div 2+5,
 Y-Radius Div 2-2,Wert);
 NeueFarbe(HilfFarbe);
end;


procedure TInhaltsknoten.Loeschen;
var HilfFarbe:TColor;
begin
 HilfFarbe:=Farbe;
 inherited Loeschen;
 NeueFarbe(clwhite);
 Zeichenflaeche.Textout(X-8*length(Inhalt_) Div 2-Radius Div 2+5,Y-Radius Div 2,Wert);
 NeueFarbe(HilfFarbe);
end;
```

Man wird auch hier wieder zuerst im Unterricht das Objekt TKnoten alleine definieren,und das Verschieben von Knoten mittels des Menüs Knoten zeichnen (Quelltext: `Knoten.Zeichnen;Objekt:=Knoten;)` sowie dem zusätzlichen Eintrag

```
if Objekt=Knoten then Knoten.NeuePosition(X,Y);
```

in der Methode FormMouseDown von TKnotenformular demonstrieren.Bei dem Objekt TInhaltsknoten verfährt man danach in analoger Weise.(Dabei sind natürlich die entsprechenden Constructor-und Destructoraufrufe in den Methoden Create und Destroy von TKnotenformular hinzuzufügen.)

Spätestens beim Zeichnen von Knoten wird den Schülern auffallen,dass bei einer Größenänderung des Anwendungsfensters die gezeichnete Figur gelöscht wird.Dies macht die Erstellung von Quellcode für die Paint-Methode von TKnotenformular nötig,die immer dann aufgerufen wird,wenn das Fenster neu gezeichnet werden muss (in dem folgenden Quelltext ist auch schon das Objekt Graph berücksichtigt):

```
procedure TKnotenformular.FormPaint(Sender: TObject);
begin
  {Punkt zeichnen:}
  if Objekt=Punkt then Punkt.Zeichnen;
  {Knoten zeichnen:}
  if Objekt=Knoten then Knoten.Zeichnen;
  {Inhaltsknoten zeichnen:}
  if Objekt=InhaltsKnoten then InhaltsKnoten.Zeichnen;
  {Graph zeichnen:}
  if Objekt=Graph then Graph.Zeichnen;
end;
```

Durch die bisherige Entwicklung des Beispiels ist im wesentlichen die oben genannte „ist"-Vererbungsbeziehung mit den Varianten statisch als auch virtual zwischen Objekten verdeutlicht,das Prinzip der visuellen Vererbung der Delphi-Entwicklungsumgebung erläutert,als auch im Hinblick auf das Ziel Aufbau eines Graphen das Zeichnen von Knoten demonstriert worden.

Durch die weiteren Objektdeklaration (einschließlich der zugehörigen Methoden)

```
TKante=class(TFigur)
  private
    Anfangsknoten_,Endknoten_:TKnoten;
  public
    constructor Create(AKno,EKno:TKnoten);
    procedure Freeall;
    function Anfangsknoten:TKnoten;
    function Endknoten:TKnoten;
    procedure Zeichnen;
    procedure Loeschen;
  end;
```

mit Fortsetzung

```
  TInhaltskante=class(TKante)
  private
    Inhalt_:string;
  public
    constructor Create(AKno,EKno:TKnoten;Ih:string);
    function Inhalt:string;
    procedure NeuerInhalt(Ni:string);
    procedure Zeichnen;
    procedure Loeschen;
  end;
```

wird jetzt die zweite Komponente eines Graphen,nämlich die Kante oder Inhaltskante und gleichzeitig die oben erläuterte „hat"-Beziehung zwischen Objekten eingeführt.

Der Objekttyp TInhaltskante wird allerdings erst für die nächste Erweiterung des Programms beim Objekttyp Graph benötigt.

Constructor und Destructor müssen jetzt jeweils berücksichtigen,dass das Objekt aus zwei importierten Objekten,den beiden Randknoten zusammengesetzt ist.Entsprechend hat der Constructor die Aufgabe beide Objekte einzufügen, und der Destructor sie wieder aus dem Speicher zu löschen,bevor TKante sich selber entfernt:

```
constructor TKante.Create(AKno,EKno:TKnoten);
begin
  inherited Create;
  Anfangsknoten_:=AKno;
  Endknoten_:=EKno;
end;

procedure TKante.Freeall;
begin
  Anfangsknoten_.Free;
  Anfangsknoten_:=nil;
  Endknoten_.Free;
  Endknoten_=nil;
  inherited Free;
end;
```

Der einfache (schon durch TObjekt vordefinierte) Destructor Free wird hier daher durch die Methode Freeall ersetzt,um den gesamten Speicher freizugeben.Für alle bisher genannten Objekte genügte der einfache von TObject vererbte) Destructor Free.(Dies ist der Grund,warum sich TFigur von TObject ableitet.Dadurch entfällt das Schreiben eines eigenen Destructors.Die Methode Free hat gegenüber der Methode Destroy den Vorteil,dass Free auch auf Zeiger,die den Wert nil haben und denen kein Objekt mehr zugeordnet ist,ohne Programmabsturz angewendet werden kann.)

Durch die in der Methode Create vorhandenen zwei Parametern nämlich durch die Randknoten der Kante wird nun auch die dritte Beziehung zwischen Objekten nämlich die „kennt"-Beziehung (Austausch von Botschaften) demonstriert.Anschaulich läßt sich das folgendermassen ausdrücken:

Eine (hier allerdings im Stadium des Entstehens befindliche) Kante muß unbedingt zwei Knoten kennen,bevor sie die Knoten als Randknoten besitzen kann.Vor dem Aufruf des Constructors Create müssen nämlich zunächst die beiden Randknoten duch ihre Constructor-Methoden erzeugt worden sein.Durch den Aufruf der Methode Create von TKante oder TIhaltskante erfolgt das Aussenden einer Botschaft an die beiden vorhandenen Knoten sich zum Objekt Gesamtkante zusammenzuschließen.

Die Methoden Zeichnen und Loeschen können dann,weil die Kante die Randknoten besitzt,ebenfalls auf die vom Objekt TInhaltsknoten durch TKnoten mittels Vererbung bereitgestellten Methoden X und Y zurückgreifen (Austausch von Botschaften/"kennt"-Beziehung):

```
procedure TKante.Zeichnen;
var Hilffarbe:TColor;
begin
  if (Anfangsknoten_<>nil) and (Endknoten_<>nil)
  then
  begin
    Anfangsknoten_.Zeichnen;
    Endknoten_.Zeichnen;
    Hilffarbe:=Farbe;
    NeueFarbe(clwhite);
    ZeichenFlaeche.Moveto(Anfangsknoten_.X,Anfangsknoten_.Y);
    NeueFarbe(clblack);
    Zeichenflaeche.Lineto(Endknoten_.X,Endknoten_.Y);
    NeueFarbe(HilfFarbe);
  end;
end;



procedure TKante.Loeschen;
var Hilffarbe:TColor;
begin
  if (Anfangsknoten_<>nil) and (Endknoten_<>nil)
  then
  begin
    Anfangsknoten_.Loeschen;
```

```
    Endknoten_.Loeschen;
    Hilffarbe:=Farbe;
    NeueFarbe(clwhite);
    ZeichenFlaeche.Moveto(Anfangsknoten_.X,Anfangsknoten_.Y);
    Zeichenflaeche.Lineto(Endknoten_.X,Endknoten_.Y);
    NeueFarbe(HilfFarbe);
  end;
end;

procedure TInhaltskante.Zeichnen;
var HilfFarbe:TColor;
begin
  if (Anfangsknoten<>nil) and (Endknoten<>nil)
  then
  begin
    HilfFarbe:=Farbe;
   inherited Zeichnen;
   NeueFarbe(clblack);
   Zeichenflaeche.Textout((Anfangsknoten.X+Endknoten.X) Div 2,
   (Anfangsknoten.Y+Endknoten.Y) Div 2,Wert);
   NeueFarbe(HilfFarbe);
 end;
end;

procedure TInhaltskante.Loeschen;
var HilfFarbe:TColor;
begin
  if (Anfangsknoten<>nil) and (Endknoten<>nil)
  then
  begin
    HilfFarbe:=Farbe;
    inherited Loeschen;
    NeueFarbe(clwhite);
    Zeichenflaeche.Textout((Anfangsknoten.X+Endknoten.X) Div 2,
    (Anfangsknoten.Y+Endknoten.Y) Div 2,Wert);
    NeueFarbe(HilfFarbe);
  end;
end;
```

Der Benutzeroberfläche in Form des Objekts TKnotenformular wird jetzt ein neuer Menüpunkt nämlich „Kante zeichnen“ mit dem Quelltext Objekt:=Kante hinzugefügt.

Der neu hinzuzufügende Quelltext in der Methode TKnotenformular.FormMouseDown gestaltet sich aufwendiger,weil zwei (Rand-) Knoten nacheinander erzeugt werden müssen, und deshalb die Methode zweimal nacheinander aufgerufen werden muss,wobei beim ersten Mal der Anfangsknoten der Kante und beim zweiten Mal der Endknoten der Kante sowie die Kante selber erzeugt werden müssen.

Die verschiedenen Fälle werden durch die Boolesche Variable ZweiterKnoten gesteuert,die nachdem der Anfangsknoten erzeugt worden ist von false auf true gesetzt wird:

```
if Objekt=Kante then
  begin
    if not ZweiterKnoten then
    begin
      Init;
      Objekt:=Kante;
      InhaltsKnoten1:=TInhaltsknoten.Create
        (X,Y,15,InputBox('Eingabe', 'Eingabe:','0'));
      InhaltsKnoten1.Zeichnen;
      ZweiterKnoten:=true;
    end
    else
    begin
      InhaltsKnoten2:=TInhaltsKnoten.Create
        (X,Y,15,InputBox('Eingabe', 'Eingabe:','0'));
      InhaltsKnoten2.Zeichnen;
      ZweiterKnoten:=false;
      Kante:=TKante.Create(Inhaltsknoten1,Inhaltsknoten2);
      Kante.Zeichnen;
      Kante.Freeall;
      Kante:=nil;
    end;
  end;
```

Nachdem Knoten und Kanten definiert wurden,kann jetzt das Objekt TGraph definiert werden.Gemäß der obigen Bemerkung wird die „hat"-Beziehung des Graphen bezüglich seiner Knoten und Kanten durch eine Array-Struktur realisiert,weil die Struktur der Liste den Schülern noch nicht bekannt ist:

```
TGraph=Class(TFigur)
  private
    Knotenliste_:Array[1..Max] of TInhaltsKnoten;
    Kantenliste_:Array[1..Max] of TInhaltsKante;
    Knotenindex_:Integer;
    Kantenindex_:Integer;
   public
    Constructor Create;
    procedure Freeall;
    function GraphKnoten(Kno:TKnoten):TInhaltsKnoten;
    procedure KnotenEinfuegen(Kno:TInhaltsknoten);
    procedure KanteEinfuegen(Ka:TInhaltsKante);
    procedure Knotenloeschen(Kno:TKnoten); {Zusatz}
    procedure Zeichnen;
    procedure Loeschen;
  end;
```

Zusätzlich zur Verwaltung der Operationen Einfügen von Knoten und Kanten sowie Knotenloeschen müssen noch die Felder Knotenindex_ und Kantenindex_ bereitgestellt werden,die jeweils angeben,wieviele Knoten und Kanten momentan im Graphen enthalten sind.

Zum Einfügen und Knotenloeschen enthält das Objekt TGraph die oben genannten geeigneten Methoden.Wie man sieht,handelt es sich wieder um „kennt"-Beziehungen,da die Botschaften an Knoten und Kanten (als Parameter der Methoden) gerichtet werden.Außerdem sind wieder die Methoden Zeichnen und Loeschen vorhanden.Die Methode Graphknoten ermittelt zu einem beliebigen vorgegeben Knoten an Hand der X/Y-Koordinaten einen Knoten, der im Graphen vorhanden ist,falls dieser nicht weiter als der Knotenradius von dem vorgegebenen Knoten entfernt liegt, und dient dazu zu einem Mausklick auf die Zeichenoberfläche einen nahegelegenden Knoten des Graphen zu ermitteln.(Bei dieser Methode ist zu berücksichtigen,dass die Abstandsberechnung nicht mittels Integer-Zahlen durchgeführt werden sollten,da sonst der gültige Zahlenbereich schnell überschritten wird.)

```
function TGraph.GraphKnoten(Kno:TKnoten):TInhaltsKnoten;
label endproc;
var Index:Integer;
    DX,DY:Real;
begin
  GraphKnoten:=nil;
  for Index:=1 to Knotenindex_ do
  begin
    DX:=Kno.X-Knotenliste_[Index].X;
    DY:=Kno.Y-Knotenliste_[INdex].Y;
    if sqrt(sqr(DX)+sqr(DY))
      <=Kno.Radius
    then
      begin
        GraphKnoten:=Knotenliste_[Index];
        goto endproc;
      end;
   end;
   endproc:
end;
```

Die Methoden Zeichnen und Loeschen benutzen die entsprechenden Methoden der Objekte (TInhalts-)Knoten und (TInhalts-)Kanten:

```
procedure TGraph.Zeichnen;
var HilfFarbe:TColor;
    Index:Integer;
begin
  HilfFarbe:=Farbe;
  NeueFarbe(clblack);
  for Index:=1 to Knotenindex_ do
    Knotenliste_[Index].Zeichnen;
  for Index:=1 to Kantenindex_ do
```

```
    Kantenliste_[Index].Zeichnen;
  NeueFarbe(HilfFarbe)
end;

procedure TGraph.Loeschen;
var HilfFarbe:TColor;
    Index:Integer;
begin
  HilfFarbe:=Farbe;
  NeueFarbe(clwhite);
  for Index:=1 to Knotenindex_ do
    Knotenliste_[Index].Loeschen;
  for Index:=1 to Kantenindex_ do
    Kantenliste_[Index].Loeschen;
  NeueFarbe(HilfFarbe)
end;
```

Während sich die Einfuege-Methoden relativ einfach gestalten,

```
procedure TGraph.KnotenEinfuegen(Kno:TInhaltsknoten);
begin
  if Knotenindex_<Max then
  begin
    Knotenindex_:=Knotenindex_+1;
    Knotenliste_[Knotenindex_]:=Kno;
  end
  else
    Showmessage('Keine weiteren Knoten möglich!');
end;

procedure TGraph.KanteEinfuegen(Ka:TInhaltsKante);
begin
  if Kantenindex_<Max then
  begin
    Kantenindex_:=Kantenindex_+1;
    Kantenliste_[Kantenindex_]:=Ka;
  end
  else
    Showmessage('Keine weiteren Kanten möglich!');
end;
```

zeigt die Methode Knotenloeschen die Unzulänglichkeit des Datentyps Array zur Darstellung der Knotenmenge des Graphen.Wie man sieht, erfordert das Löschen das wiederholte Umspeichern innerhalb der Array-Struktur:

```
procedure TGraph.Knotenloeschen(Kno:TKnoten);
var Index,Zaehl,Stelle:Integer;
    IKno:TInhaltsknoten;
begin
  Stelle:=0;
  for Index:=1 to Knotenindex_ do
  begin
    if Knotenliste_[Index]=Kno
    then
      Stelle:=Index;
  end;
  if (Stelle>0) and (Stelle<=Knotenindex_)
  then
  begin
    for Zaehl:=Stelle to Knotenindex_-1 do
    Knotenliste_[Zaehl]:=Knotenliste_[Zaehl+1];
    Knotenindex_:=Knotenindex_-1;
  end;
  if Kantenindex_>0 then Index:=1;
  while Index<=Kantenindex_ do
  begin
    if (Kantenliste_[Index].Anfangsknoten_=Kno)
    or (Kantenliste_[Index].Endknoten_=Kno)
    then
    begin
      Stelle:=Index;
      if (Stelle>0) and (Stelle<=Kantenindex_)
      then
      begin
        for Zaehl:=Stelle to Kantenindex_-1 do
        Kantenliste_[Zaehl]:=Kantenliste_[Zaehl+1];
        Kantenindex_:=Kantenindex_-1;
        Index:=0;
      end;
    end;
```

```
  Index:=Index+1;
  end;
end;
```

Man sollte diese Methode zuletzt besprechen, und ihre Komplexheit zum Anlass nehmen zum nächsten Thema, nämlich dem Aufbau einer wiederverwendbaren Objektliste überzuleiten.

Das Erzeugen eines Graphen in der Methode TKnotenformular.FormMouseDown gestaltet sich jetzt folgendermassen:

```
if Objekt = Graph then
  begin
    {Für Graph Inhaltsknoten erzeugen:}
    if Shift=[ssleft] then
    begin
      IKnoten:=TInhaltsknoten.Create
        (X,Y,15,InputBox('Eingabe', 'Eingabe:','0'));
      Graph.Knoteneinfuegen(IKnoten);
      Graph.zeichnen;
      ZweiterKnoten:=false;
    end;
    {Für Graph Inhaltskanten erzeugen:}
    if Shift=[ssright] then
    begin
      if not ZweiterKnoten then
      begin
        Knoten1:=TKnoten.Create(X,Y,15);
        IKnoten1:=Graph.GraphKnoten(Knoten1);
        Knoten1.free;
        Knoten1:=nil;
        if IKnoten1<>nil
        then
        begin
          Showmessage('1.Knoten: '+IKnoten1.Wert);
          ZweiterKnoten:=true;
        end
        else
          Showmessage('Kein Knoten!')
       end
      else
      begin
        Knoten2:=TKnoten.Create(X,Y,15);
        IKnoten2:=Graph.Graphknoten(Knoten2);
        Knoten2.free;
        Knoten2:=nil;
        if IKnoten2<>nil
        then
        begin
          Showmessage('2. Knoten: '+IKnoten2.Wert);
          ZweiterKnoten:=false;
          Inhaltskante:=TInhaltsKante.Create(IKnoten1,IKnoten2,
            InputBox('Eingabe', 'Eingabe:','0'));
          Graph.Kanteeinfuegen(InhaltsKante);
          Graph.Zeichnen;
        end
        else
        begin
          Showmessage('Kein Knoten!');
          ZweiterKnoten:=false;
        end;
      end;
    end;
    {Für Graph Inhaltsknoten löschen:}
    if  Shift=[ssShift,ssleft]
    then
    begin
      Knoten1:=TKnoten.Create(X,Y,15);
      IKnoten1:=Graph.GraphKnoten(Knoten1);
      Graph.Loeschen;
      Graph.Knotenloeschen(IKnoten1);
      Knoten1.Free;
      Knoten1:=nil;
      Graph.Zeichnen;
    end;
  end;
```

Die Erzeugung der Kanten erfolgt analog dem weiter oben beschriebenen Verfahren.Wenn ein Mausklick mit der linken Maustaste auf die Zeichenoberflä-

che erfolgt,wird ein Inhaltsknoten erzeugt und in den Graph eingefügt.Bei einem Mausklick mit der rechten Maustaste nacheinander auf zwei schon auf der Zeichenoberfläche vorhandenen Graphknoten,wird eine Inhaltskante erzeugt und in den Graph eingefügt.Wenn zusätzlich die Shift-Taste beim Mausklick mit der Linken Maustaste auf einen Graphknoten gedrückt wird,wird der Knoten gelöscht.

Schließlich ist es noch sinnvoll,um die Methoden Zeichnen und Loeschen auch anzuwenden,die Verschiebung der Knoten des Graphen ereignisgesteuert mit der Maus zu programmieren.Dies geschieht durch Benutzung der von Delphi vordefinierten Methode TKnotenformular.MouseMove unter Einsatz der Methode NeuePosition von TKnoten:

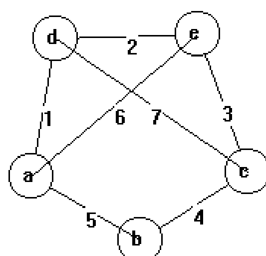
```
procedure TKnotenformular.FormMouseMove(Sender: TObject;
 Shift: TShiftState; X,Y: Integer);
begin
  {Für Graph Inhaltsknoten NeuePosition:}
  if Objekt=Graph
  then
    if Shift=[ssCtrl]
    then
    begin
      Knoten1:=TKnoten.Create(X,Y,15);
      IKnoten1:=Graph.Graphknoten(Knoten1);
      if IKnoten1<>nil then
      begin
        Graph.Loeschen;
        IKnoten1.NeuePosition(X,Y);
        Graph.zeichnen;
      end;
    end;
end;
```

Das Ereignis MouseMove tritt dann ein,wenn die Maustaste gedrückt gehalten bleibt.Zusätzlich muß noch die Strg-(Ctrl-) Taste gedrückt werden,damit ein Knoten verschoben wird.(sonst Überschneidung mit dem Erzeugen eines Knotens beim Drücken der linker Maustaste)

Der komplette Quellcode des Einführungsprojekt ist wie gesagt dem Anhang zu entnehmen oder als Komplettprogramm auf der Installations-CD enthalten.

Nach Durchführung dieses Einführungsprojektes kennen die Schüler die Grundlagen objektorientierten Programmierens,können mit Hilfe der Delphi-Entwicklungsumgebung ereignisgesteuerte Programme entwerfen und haben einen ersten Einblick erhalten,wie die Objektstruktur eines Graphen aus den Unterobjekten Knoten und Kante prinzipiell aufzubauen ist,und der Graph zeichnerisch dargestellt werden kann.

Das folgende Bild zeigt die Benutzeroberfläche des Einführungsprojektes:



Als (dreistündige) Abschlußklausur zu dem Thema wurde dann die folgende Aufgabe gestellt,bei der eine ähnliche Objektfigurbeziehung untersucht werden soll,wie sie im Unterricht besprochen wurde.

Klausuraufgabe:

a)Definieren und erläutern Sie folgende Begriffe der Objektorientierten Programmierung:

Objekt (bzw. Klasse) ,Instanz ,Vererbung ,Datenfeld, Methode

b)Unter Benutzung der im Anhang genannten Datentypen bzw. Methoden soll eine Unit UGraph erstellt werden,die folgende Objekte mit ihren Methoden exportiert:

```
type

  TFigur=class(TObject)
  private
    Zeichenflaeche_:TCanvas;
    Farbe_:TColor;
  public
    constructor Create;
    function Zeichenflaeche:TCanvas;
    function Farbe:TColor;
    procedure NeueFarbe(F:TColor);
    procedure Zeichnen;virtual;abstract;
    procedure Loeschen;virtual;abstract;
  end;


  TPunkt=class(TFigur)
  private
    P_:TPoint;
  public
    constructor Create(Pt:TPoint);
    function Punkt:TPoint;
    procedure NeuerPunkt(Pt:TPoint);
    procedure Zeichnen;override;
    procedure Loeschen;override;
    procedure NeuePosition(Pt:TPoint);
  end;
```

Das Objekt TFigur ist Ihnen aus dem Unterricht bekannt.

Die Methode Punkt gibt für das Objekt TPunkt die X/Y-Koordinaten des Punktes in Form des Datentyps TPoint zurück.(siehe Auszug aus der Hilfe zu Delphi) Die Methode NeuerPunkt setzt das Datenfeld P_ vom Typ TPoint auf einen neuen Wert, und NeuePosition verschiebt den Punkt an eine neue Position mit X/Y-Koordinaten, die jeweils durch Pt vorgegeben sind. Zeichnen bzw. Loeschen zeichnen Instanzen des Objekts Punkt auf der Zeichenoberfläche bzw. löschen sie.

Die Unit UGraph enthält als Variable den Eintrag:

```
var Oberflaeche:TCanvas;
```

und als uses-Anweisung:

```
uses Graphics,Wintypes;
```

Schreiben Sie den Quelltext aller angegebenen Methoden von TPunkt.Welche Aufgabe hat der Constructor?

```
TPunkt:
     .
   P_     (bzw. Pt)
```

c)Von dem Objekttyp TPunkt leitet sich durch Vererbung der Typ TStrecke (waagerechte Strecke) und von diesen Typ der Typ TRechteck (Rechteck,dessen Seiten parallel zum Bildschirmrand sind) ab:

```
TStrecke=class(TPunkt)
  private
    Laenge_:Integer;
  public
    constructor Create(Pt:TPoint;NeueLaenge:Integer);
    function Laenge:Integer;
    procedure NeueLaenge(L:Integer);
    procedure Zeichnen;override;
    procedure Loeschen;override;
  end;

  TRechteck=class(TStrecke)
  private
    Breite_:Integer;
  public
    constructor Create(Pt:TPoint;NeueLaenge,NeueBreite:Integer);
    function Breite:Integer;
    procedure NeueBreite(B:Integer);
    procedure Zeichnen;override;
    procedure Loeschen;override;
  end;
```

TStrecke:

TRechteck:

Schreiben den Quelltext der Methoden Zeichnen und Loeschen der angegebenen Objekte.Definieren Sie eine Property Wert für TStrecke an,durch die auf das Datenfeld Laenge_ zugegriffen werden kann.

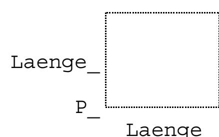
d)Was bedeutet es,dass eine Methode als virtual deklariert wird? Erläutern sie,warum es vorteilhaft ist,die Methoden Zeichnen und Loeschen als virtual zu deklarieren.Was passiert,wenn diese Methoden nicht virtual sind?Was bedeutet die Eigenschaft override?

Welche Bedeutung hat die Deklaration virtual und abstract bei den Methoden Zeichnen und Loeschen von TFigur?

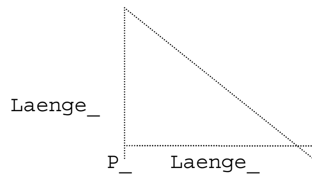
e)Leiten Sie von der Objektklasse TRechteck den Typ TQuadrat (als zu den Begrenzungen der Zeichenoberfläche paralleles Quadrat) ab, und schreiben Sie dafür die Constructor-Methode.Warum sind für diesen Objekttyp keine neuen Datenfelder nötig, und warum fehlen hier die Methoden Zeichen und Loeschen?

```
TQuadrat=class(TRechteck)
    constructor Create(Pt:TPoint;Seite:Integer);
  end;
```

TQuadrat:



f)Vom Datentyp TQuadrat soll schließlich noch der Datentyp TDreieck (**gleichschenkliges** Dreieck mit rechtem Winkel zwischen den Schenkeln,dessen Schenkel zu den Rändern der Zeichenoberfläche parallel sind) durch Vererbung abgeleitet werden.Schreiben Sie selber eine entsprechende Typdeklaration mit den nötigen Datenfeldern und Methoden.Geben Sie den Quelltext der Methoden an.



g)Durch das visuelle Vererben und Erzeugen einer Form sowie des visuellen Erzeugens eines Menüs in dieser Form mit den Einträgen Dreieck und Ende,die ein Dreieck auf der Zeichenoberfläche an der Anfangsposition erzeugen und anzeigen bzw. das Programm beenden,werden in einer Unit UBeisp folgende Deklarationen vorgegeben:

```
unit UBeisp;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, Menus, StdCtrls,UGraph;

type
  TFormneu = class(TForm)
    MainMenu: TMainMenu;
    Dreieckzeichnen: TMenuItem;
    Ende: TMenuItem;
    procedure FormActivate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
     procedure FormPaint(Sender: TObject);
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure DreieckzeichnenClick(Sender: TObject);
    procedure EndeClick(Sender: TObject);

  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

var
  Formneu: TFormneu;
  Dreieck:TDreieck;
  Objekt:TObject;

implementation
.
.
.

end.
```

Erläutern Sie die obige Deklarationen (der Type-Anweisung).Welche Bedeutung haben die Methoden DreieckzeichnenClick und EndeClick sowie FormActivate?

h)Die Methode procedure FormMouseDown(Sender: TObject; Button: TMouseButton;Shift: TShiftState; X, Y: Integer) soll durch Mausklick mit der linken Maustaste auf eine Position der Zeichenoberfläche das Dreieck an der neuen Position zeichnen (und an der alten Position löschen.)Schreiben Sie den Quelltext.

i)Welche Bedeutung kommt der Methode procedure FormPaint(Sender: TObject) zu? Geben Sie auch hierfür den Quelltext an.

k)Erläutern Sie den Quelltext des Hauptprogramm dieser Anwendung:

```
program Klausur;

uses
  Forms,
  UGraph in 'UGraph.pas',
  UBeisp in 'UBeisp.pas' {Formneu};
```

```
{$R *.RES}

begin
  Application.CreateForm(TFormneu, Formneu);
  Application.Run;
end.
```

l)Beschreiben Sie,wie das (Gesamt-)Programm nach seinem Start zu bedienen ist.

Anhang:

Auszug aus der Hilfe zu Delphi:

Verwendung

TCanvas, Objekt (Unit Graphics)

Deklaration

procedure LineTo(X, Y: Integer);

Beschreibung

Die Methode LineTo zeichnet auf die Zeichenfläche eine Linie von der aktuellen Zeichenposition (durch die Eigenschaft PenPos angegeben) zu dem durch X und Y angegebenen Punkt und setzt die Stiftposition auf (X, Y).

Verwendung

TCanvas, Objekt (Unit Graphics)

Deklaration

procedure MoveTo(X, Y: Integer);

Beschreibung

Die Methode MoveTo ändert die aktuelle Zeichenposition in die Koordinaten, die in X und Y übergeben wurden. Die aktuelle Position wird durch die Eigenschaft PenPos angegeben. Sie sollten MoveTo zum Setzen der aktuellen Position verwenden, anstatt PenPos direkt zu setzen.

Verwendung

Unit: WinTypes

Deklaration

```
TPoint = record
  X: Integer;
  Y: Integer;
end;
```

Beschreibung:

Der Typ TPoint definiert die Position eines Pixels auf dem Bildschirm. Der Ursprung des Koordinatensystems liegt in der linken oberen Ecke des Bildschirms. X ist die horizontale Koordinate des Punktes, Y die vertikale.

Deklaration

function Point(AX, AY: Integer): TPoint;

Beschreibung

Die Funktion Point nimmt die in AX und AY übergebenen x- und y-Koordinaten und gibt einen Datensatz TPoint zurück. Häufig wird Point zur Konstruktion eines Parameters für eine Funktion, die einen oder mehrere Datensätze TPoint verlangt, verwendet.

Beschreibung von TCanvas:

Das Objekt TCanvas ist eine Zeichenfläche. Es repräsentiert einen Bereich, in dem Ihr Programm zeichnen kann. Ein Objekt TCanvas kapselt einen Windows HDC Anzeige-kontext ein.
Der Pinsel, der Stift und die Schriftart, die zum Zeichnen auf der Zeichenfläche benutzt werden, sind spezifiziert in den Eigenschaften Brush, Pen (z.B. in der Form Pen.Color (clblack/clwhite/clbtnface)) und Font.

Ende des Auszuges aus der Hilfe zu Delphi

Lösung:

Der Quelltext des Programms Klausur befindet sich im Anhang und auf der Installations-CD.

Die Klausur vollzieht an einem ähnlichen Beispiel noch einmal den im Unterricht besprochenen Stoff nach und erfordert neben Wissen den Transfer von im Unterricht angewendeten Verfahren,d.h. sie beschränkt sich im wesentlichen auf die Stufen 1 bis 4 der Taxonomie von Bloom.

Das Einführungsbeispiel zeigt einen ersten noch relativ unvollständigen Ansatz,wie die Objektdatenstruktur eines Graphen gestaltet werden kann.Während es gemäß der dargestellten Struktur schon möglich ist von den Kanten auf die (Rand-)Knoten zuzugreifen,fehlt z.B. der für die Programmierung von Graphenalgorithmen wichtige umgekehrte Zugriff noch vollständig.Ein anderes Manko ist,wie die Methode Knotenloeschen zeigt (s.o.) die Verwendung der Datenstruktur Array zur Speicherung der Knoten und Kanten. Bedeutend günstiger ist hier die Verwendung einer dynamischen Datenstruktur in Form einer Objekt-Liste,deren gespeicherte Datentypen noch nachträglich von einem Benutzer festgelegt werden können.Das Thema, mit dem gleichzeitig auch die Spezialfälle Keller und Schlange besprochen werden können,entspricht den von den Richtlinien für die Jahrgangsstufen 12 oder 13 geforderten Unterrichtsinhalten.Ein entsprechendes zweites Unterrichtsprojekt wird im nächsten Abschnitt beschrieben.

3) Liste als Objekt zur Vorbereitung der allgemeinen Graphenstruktur

Die zweite Voraussetzung für den Aufbau eines Graphen sind Kenntnisse der Struktur,der Erzeugung und der Anwendung einer Objektliste,deren Elementinhalte und Datentypen von einem Benutzer noch nachträglich beim Einfügen festgelegt werden können.

Ausgehend von der üblichen Deklaration einer Liste mit Inhaltsfeld ,die den Schülern schon aus dem Unterricht der Jahrgangsstufe 11 als Beispiel zur Verwendung von Zeigern bekannt war,

```
type Zeiger = ^Element;
     Element = RECORD
                 Weiter : zeiger;
                 Inhalt    : INTEGER;
               end;
```

oder mit allgemeinerem Datentyp TElement

```
type Zeiger = ^Element;
     Element = RECORD
                 Weiter : Zeiger;
                 Element    : TElement;
               end;
```

wird sofort klar,dass bei der Konstruktion einer objektorientierten Liste, deren Elementtyp zunächst nicht festgelegt wird, das Feld Element zunächst besser fehlen sollte.Dieses führt zu der Deklaration

```
type TList = class(TObject)
     private
      Weiter_:TList;
     end;
```

Welches aber sind jetzt die Elemente der Liste?

Da jetzt kein Inhalts-oder Elementfeld vorhanden ist,ist das Objekt selber als Element aufzufassen,d.h.

```
   TElement=TList;
```

so dass derselbe Zeiger zwei verschiedene Bedeutungen erhält:

1)Zeiger auf eine Teil-Liste oder die Gesamtliste
2)Zeiger auf ein Listenelement,das bisher noch keinen Inhalt enthält.

Zwischen den beiden Auffassungen kann mittels Typkonversion (Type-Casting mittels TList bzw. TElement) hin-und hergeschaltet werden.

Dieser Unterschied sollte den Schülern an dieser Stelle verdeutlicht werden.(Dadurch kann auch das Type-Casting eingeführt werden.)

Um nun Platz für einen Inhalt zu schaffen,wird die obige Deklaration objektorientiert erweitert zu:

```
TElement=class(TList)
   private
     OB_:TObject;
   public
     constructor Create(Ob:TObject);
     function Objekt:TObject;
   end;
```

Damit können in die Liste beliebige Objekte vom Typ TObject eingefügt werden.

Die eigentliche Objekt-Liste,die die Verwaltungsmethoden der Liste enthält,wird schließlich durch einen dritten Typ mittels

```
TListe = class(TList)
```

definiert.

Es sind dann die üblichen Methoden zur Verwaltung einer Liste „Liste ist leer",Anzahl (der Elemente), Einfügen und Löschen am Anfang und am Ende sowie an beliebiger Stelle mit der Nummer Index (Index läuft dabei von 0 bis Anzahl-1) sowie die Ausgabe der Liste zu implementieren. Außerdem ist es sinnvoll,Zugriff auf das Erste und letzte Element als auch das dort enthaltende erste und letzte Objekt (ErstesElement, LetztesElement,Erstes und Letztes) zu haben.

```
  TListe = class(TList)
    constructor Create;
    procedure Freeall;
    function Leer: Boolean;
    function Weiter: TListe;
    function LetztesElement:TElement;
    function ErstesElement:TElement;
    function Erstes: TObject;
    function Letztes: TObject;
    function Anzahl:Integer;
    function Element(Index:Integer):TElement;
    function Objekt(Index:Integer):TObject;
    function Liste(Index:Integer):TListe;
    function Position(Ob:TObject):Integer;
    procedure AmAnfanganfuegen(Ob:TObject);
    procedure AmEndeAnfuegen(Ob:TObject);
    procedure AnPositioneinfuegen(Ob:TObject;Index:Integer);
    procedure AmAnfangLoeschen(var Ob:TObject);
    procedure AmEndeLoeschen(var Ob:TObject);
    procedure Loeschen(Ob:TObject);
    procedure AnPositionLoeschen(Index:Integer);
    procedure FueralleElemente (Vorgang:TVorgang);
    procedure FueralleElementezurueck(Vorgang:TVorgang);
  end;

mit TVorgang=Procedure(X:TObject);
```

Die Methode Weiter gibt Zugriff auf das nächste Listenelement und Element bzw. Liste auf das Element oder die Liste an der Stelle Index.Position gibt den Index eines Elementes,das das vorgegebene Objekt beinhaltet,zurück.Mittels der letzten beiden Methoden FueralleElemente bzw. FueralleElementzurueck läßt sich eine Operation (z.B. die Ausgabe der Liste,sobald konkrete Elementinhalte implementiert sind) auf allen Elementen

oder Objekten der Liste durchführen.Somit sind alle üblichen Methoden zur Verwaltung einer Liste vorhanden.

Es ist nicht unbedingt notwendig alle Methoden im Unterricht zu implementieren,sondern sollte sich auf eine Auswahl beispielhafter Methoden beschränken.Z.B. sollten die Methoden zum Anfügen und Löschen von Elementen am Anfang und Ende der Liste sowie die Methoden ErstesElement und LetztesElement sowie Leer und Anzahl vorhanden sein,wenn man anschließend die Datenstrukturen Keller und Schlange besprechen will,die sich dann folgendermaßen deklarieren lassen:

```
TKeller=class(TListe)
  private
    Stack_:TListe;
  public
    constructor Create;
    procedure Freeall;
    function Leer:Boolean;
    function Top:TObject;
    procedure Pop;
    procedure Push(Ob:TObject);
  end;

  TSchlange=class(TListe)
  private
    Tail_:TListe;
  public
    constructor Create;
    procedure Freeall;
    function Leer:Boolean;
    function Top:TObject;
    procedure Unqueue;
    procedure Queue(Ob:TObject);
  end;
```

Natürlich lassen sich die Strukturen Keller und Schlange auch vor der allgemeinen Listenstruktur im Unterricht besprechen,indem man zunächst nur die für diese Objekte benötigten Methoden der Liste implementiert.

Nachfolgend Beispiele für die Implementation einiger Methoden.Der Gesamtquelltext befindet sich im Anhang.

```
constructor TElement.Create(Ob:TObject);
begin
  inherited Create;
  Ob_:=Ob;
end;

function TElement.Objekt:TObject;
begin
  Objekt:=Ob_;
end;

constructor TListe.Create;
begin
  inherited Create;
  Weiter_ := nil;
end;

procedure TListe.Freeall;
begin
  if (Weiter_<>nil) and (Weiter_.Weiter_<>nil)
  then
    TListe(Weiter_).Freeall;
  if Weiter_<>nil then
  begin
    Weiter_.Free;
    Weiter_:=nil;
  end;
  inherited Free;
end;

function TListe.Leer: Boolean;
begin
  Leer := (Weiter_=nil);
end;
```

```
function TListe.Weiter:TListe;
begin
  if not Leer
  then
    Weiter:= TListe(Weiter_);
end;

function TListe.LetztesElement:TElement;
begin
  if Leer
  then
    LetztesElement:=TElement(self)
  else
    LetztesElement:= TListe(Weiter_).LetztesElement;
end;

function TListe.ErstesElement:TElement;
begin
  if not Leer
  then
    ErstesElement := TElement(Weiter_)
end;

function TListe.Erstes:TObject;
begin
    Erstes := ErstesElement.Objekt;
end;

function TListe.Letztes:TObject;
begin
    Letztes:= LetztesElement.Objekt;
end;

function TListe.Anzahl:Integer;
begin
  if Leer
  then
    Anzahl:=0
  else
    Anzahl:= TListe(Weiter_).Anzahl+1;
end;

function TListe.Objekt(Index:Integer):TObject;
begin
  if (Index<0)or(Index>Anzahl-1)
  then
    Objekt:=nil
  else
    if Leer
    then
      Objekt:=nil
    else
      if Index=0
      then
        Objekt:=ErstesElement.Objekt
      else
        Objekt:=TListe(Weiter_).Objekt(Index-1);
  end;

  function TListe.Liste(Index:Integer):TListe;
begin
  if (Index<0) or (Index>Anzahl-1)
  then
    Liste:=nil
  else
    if Leer
    then
      Liste:=nil
    else
      if Index=0
      then
        Liste:=self
      else
        Liste:= TListe(Weiter_).Liste(Index-1);
end;

procedure TListe.AmAnfangAnfuegen(Ob:TObject);
var X:TElement;
begin
  X:=TElement.Create(Ob);
  X.Weiter_:=Weiter_;
```

```
  Weiter_:=X;
end;

procedure TListe.AmEndeAnfuegen(Ob:TObject);
var X:TElement;
begin
  X:=TElement.Create(Ob);
  X.Weiter_:=nil;
  LetztesElement.Weiter_:=X;
end;

procedure TListe.AmAnfangLoeschen(var Ob:TObject);
var X:TElement;
begin
  if not Leer
  then
  begin
    X :=ErstesElement;
    Weiter_:=X.Weiter_;
    X.Weiter_:=nil;
    Ob:=X.Objekt;
    X.Free;
    X:=nil;
  end;
end;

procedure TListe.AmEndeLoeschen(var Ob:TObject);
var X,Y:TListe;
begin
  Y:=self;
  While (not Y.Leer) and (not TListe(Y.Weiter_).Leer) do
  Y:=TListe(Y.Weiter_);
  X:=TListe(Y.Weiter_);
  Y.Weiter_:=TListe(Y.Weiter_.Weiter_);
  X.Weiter_:=nil;
  Ob:=TElement(X).Objekt;
  X.Free;
  X:=nil;
end;

procedure TListe.FueralleElemente(Vorgang:TVorgang);
begin
  if not Leer
  then
  begin
    Vorgang(ErstesElement.Objekt);
    TListe(Weiter_).FueralleElemente(Vorgang);
  end;
end;

constructor TKeller.Create;
begin
  Stack_:=TListe.Create;
end;

procedure TKeller.Freeall;
begin
  Stack_.Freeall;
  Stack:=nil;
  inherited Free;
end;

function TKeller.Leer;
begin
  Leer:=stack_.Leer;
end;

function TKeller.Top:TObject;
begin
  Top:=Stack_.ErstesElement.Objekt;
end;

procedure TKeller.Pop;
var Ob:TObject;
begin
  Stack_.AmAnfangLoeschen(Ob);
  Ob.Free;
  Ob:=nil;
end;


procedure TKeller.Push(Ob:TObject);
begin
```

```
  Stack_.AmAnfangAnfuegen(Ob);
end;

constructor TSchlange.Create;
begin
  Tail_:=TListe.Create;
end;

procedure TSchlange.Freeall;
begin
  Tail_.Freeall;
  Tail:=nil;
  inherited Free;
end;

function TSchlange.Leer;
begin
  Leer:=Tail_.Leer;
end;

function TSchlange.Top:TObject;
begin
  Top:=Tail_.LetztesElement.Objekt;
end;

procedure TSchlange.Unqueue;
var Ob:TObject;
begin
  Tail_.AmEndeLoeschen(Ob);
  Ob.Free;
  Ob:=nil;
end;

procedure TSchlange.Queue(Ob:TObject);
begin
  Tail_.AmAnfangAnfuegen(Ob);
end;
```

Viele Methoden sind von relativ einfacher Struktur und können von Schülern,wenn das Zeiger- und Objektkonzept klar ist,und man zwei oder drei der komplexeren Methoden (z.B. Liste und AmEndeloeschen) als Muster besprochen hat,selber erstellt werden.

Durch Erzeugen eines neuen Objekttyps (Container),der beispielsweise Inhalte vom Datentyp string speichern kann,sowie durch Deklaration einer von TListe durch Vererbung abgeleiteten neuen Liste mit geeigneten Ausgabemöglichkeiten,läßt sich nun den Elementen der Liste ein Inhalt sowie dessen Verwaltungsmethoden nachträglich hinzufügen:

```
type TSObject = class(TObject)
     private
       Inhalt_: string;
     public
       constructor Create(S:String);
       function Inhalt:string;
       procedure NeuerInhalt(S:String);
       procedure Ausgabe;
    end;

   TSListe=class(TListe)
     procedure Eingabe;
     procedure Ausgabevorwaerts;
     procedure Ausgaberueckwaerts;
   end;

constructor TSObject.Create(S:string);
begin
  inherited Create;
  Inhalt_:= S;
end;

function TSObject.Inhalt:string;
begin
  Inhalt:=Inhalt_;
end;

procedure TSObject.NeuerInhalt(S:string);
begin
  Inhalt_:=S;
end;
```

```
procedure TSObject.Ausgabe;
begin
    Write(' ',Inhalt);
end;

procedure TSListe.Eingabe;
var Eingabe:string;
begin
  WriteLn('Wörter eingeben (Return zum Beenden):');
  Writeln;
  Readln(Eingabe);
  while Eingabe <> '' do
  begin
    AmEndeAnfuegen(TSObject.Create(Eingabe));
    Readln(Eingabe);
  end;
end;

procedure Ausgabe(X:TObject);
begin
  if X is TSObject
  then
    Write(' ', TSObject(X).Inhalt);
end;

procedure TSliste.Ausgabevorwaerts;
begin
  FueralleElemente(Ausgabe);
end;

procedure TSliste.Ausgaberueckwaerts;
begin
  FueralleElementezurueck(Ausgabe);
end;
```

Ein Listenelement mit dem Inhalt des strings Eingabe läßt sich nun mittels

```
SListe.AmAnfang/EndeAnfuegen(TSObject.Create(Eingabe))
```

an eine Liste vom Typ TSListe am Anfang oder Ende anfügen.

Bei Keller und Schlange geschieht das Einfügen entsprechend:

```
Keller.Push(TSObject.Create(Eingabe));
Schlange.Queue(TSObject.Create(Eingabe));
```

Um die Elemente der SListe komplett einzugeben,kann man die Methode

```
Sliste.Eingabe;
```

benutzen.

Die Ausgabe der Liste erfolgt mittels

```
Sliste.Ausgabevorwaerts/Ausgaberueckwaerts
```

bzw. von einzelnen Elementen:

```
TSObject(SListe.Erstes/Letztes).Ausgabe
```

Hier muß eine Typkonversion vorgenommen werden:

Alternativ könnten die Methoden Erstes und Letztes in dem Objekt TSListe mittels einer Typkonversion unter Zugriff auf die Vorgängermethode neu implementiert werden:

Quelltext:

```
TSliste.Erstes:=TSObjekt(Erstes)
```

Ein kleiner Schönheitsfehler des Compilers von Delphi ist, daß für die Übergabe einer Procedure als Parameter an die Methode FueralleElemente

bzw. FueralleElementezurueck eine globale Procedure Ausgabe benötigt wird, und weder eine lokale Procedure noch z.B. die Methode Ausgabe von TSObjekt (geeignet umgeschrieben) verwendet werden können, da bei Methoden und Proceduren als Parameter nur global definierte Proceduren zulässig sind.

Wie aus der Procedure bzw. Methode Ausgabe hervorgeht,ist das Hauptprogramm zur Vereinfachung als Text-bzw. Consolenanwendung (unter Verwendung der komfortablen Unit WinCrt mit Delphi 1,aber auch ohne diese Unit als Consolenanwendung unter Delphi >=2.x möglich) konzipiert,weil es hier auf eine Ereignisgesteuerte Oberfläche nicht ankommt.Es kann aber leicht auch unter Umschreibung der Ein-und Ausgabeart als Projekt mit Ausgabeform realisiert werden.

Das Hauptprogramm,von dem im folgenden nur ein Ausschnitt wiedergegegen wird, befindet sich komplett in der Dokumentation im Anhang.Das Gesamtprogramm ist ebenfalls auf der Installations-CD enthalten.

```
begin
  InitWincrt;
  SListe := TSListe.Create;
  SListe.Eingabe;
  Write(SListe.Anzahl,' String(s)');
  Writeln;
  Writeln;
  Write('Vorwärts:         ');
  SListe.Ausgabevorwaerts;
  Writeln;
  Write('Rückwärts:        ');
  SListe.Ausgaberueckwaerts;
  Writeln;
  Writeln;
  SListe.Freeall;
  SListe:=nil;
  Writeln('Keller und Schlange erzeugen:');
  Keller:=TKeller.Create;
  Schlange:=TSchlange.Create;
  Writeln;
  Writeln;
  WriteLn('Wörter eingeben (Return zum Beenden):');
  Writeln;
  Readln(Eingabe);
  while Eingabe <> '' do
  begin
    Keller.Push(TSObject.Create(Eingabe));
    Schlange.Queue(TSObject.Create(Eingabe));
    Readln(Eingabe);
  end;
  Write('Keller:');
  while not Keller.Leer do
  begin
    TSObject(Keller.Top).Ausgabe;
    Keller.Pop;
  end;
  Writeln;
  Writeln;
  Write('Schlange: ');
  while not Schlange.leer do
  begin
    TSObject(Schlange.Top).Ausgabe;
    Schlange.Unqueue;
  end;
  Keller.Freeall;
  Keller:=nil;
  Schlange.Freeall;
  Schlange:=nil;
  Writeln;
  Readln;
  Clrscr;
  Writeln('Ende');
  Readln;
  DoneWincrt;
end.
```

Als Abschluss dieser Unterrichtseinheit wurde dann folgende (dreistündige) Klausuraufgabe gestellt:

Klausuraufgabe:

Gegeben sei folgende Objekt-Datenstruktur einer wiederverwendbaren Liste (Programmiersprache Delphi):

```
type

  TList = class(TObject)
  private
    Weiter_:TList;
  end;

  TElement=class(TList)
   private
     OB_:TObject;
   public
     constructor Create(Ob:TObject);
     function Objekt:TObject;
   end;

  TListe = class(TList)
    constructor Create;
    procedure Freeall;
    function Leer: Boolean;
    function Weiter: TListe;
    function ErstesElement:TElement;
    function Erstes: TObject;
    function Anzahl:Integer;
    procedure AmAnfanganfuegen(Ob:TObject);
  end;
```

a)Erläutern Sie die Datenstruktur und geben Sie die Bedeutung aller aufgeführten Datenfelder und Methoden an.

b)Schreiben Sie den Delphi-Quellcode zu folgenden Methoden:

```
constructor TListe.Create;
function TListe.Leer: Boolean;
function TListe.Weiter: TListe;
function TListe.ErstesElement:TElement;
function TListe.Erstes:TObject;
function TListe.Anzahl:Integer;
procedure TListe.AmAnfangAnfuegen(Ob:TObject);
```

c)Leiten Sie von dem Datentyp TObject durch Vererbung einen Datentyp TZiffernObject der folgenden Typdeklaration ab,indem sie die Constructor-Methode Create,die Methode NeueZiffer und die Functions-Methode Ziffer implementieren.Erläutern Sie die Typdeklaration.

```
type

  TZiffer=0..1;

  TZiffernObject = class(TObject)
  private
    Ziffer_:TZiffer;
  public
    constructor Create(Z:TZiffer);
    function Ziffer:TZiffer;
    procedure NeueZiffer(Z:TZiffer);
  end;
```

Für die folgenden Aufgaben stehen die oben angegebenen Methoden von TListe und TZiffernObject (und nur diese) zur Verfügung:

Außerdem sei das folgende Objekt definiert:

```
type

  TZiffernListe=class(TListe)
    procedure Gib_zahl_ein;
    procedure Gib_zahl_aus;
    function Paritaet:Boolean;
    procedure Addition(Zahl:TZiffernliste);
  end;
```

d)Mit Hilfe der obigen Typdeklarationen soll eine Addition von beliebig großen Binärzahlen implementiert werden.

Dazu werden die Ziffern (0 oder 1) der Binärzahlen fortlaufend als Listenelemente der Liste TZifferliste gespeichert.

Die vorderste Ziffer wird dabei zuerst in die Liste eingefügt.Da das Anfügen jeweils am Anfang erfolgt,steht die vorderste Ziffer einer Zahl am Schluss der Liste.

Implementieren Sie die folgenden Methoden:

```
procedure TZiffernliste.Gib_zahl_ein;

procedure TZiffernliste.Gib_zahl_aus;;
```

Die Methode Gib_zahl_ein liest Ziffern solange von der Tastatur ein,bis eine Zahl ungleich 0 oder 1 eingegeben wird (Abbruchkriterium der Einfachheit halber) und speichert sie im Datentyp TZiffernliste.

Die Methode Gib_zahl_aus gibt alle Ziffern die in der Liste TZiffernliste (in der umgekehrten Reihenfolge) gespeichert in der richtigen Reihenfolge mit Hilfe der Write bzw. Writeln-Anweisung aus.

e)Implementieren Sie die Methode

```
function TZiffernliste.Paritaet:Boolean;
```

die die Quersumme aller Ziffern, die in der Liste TZiffernliste gespeichert sind, berechnet, und zurückgibt,ob die Quersumme gerade (true) oder ungerade (false) ist.

(Diese Eigenschaft bezeichnet man als die Parität der Binärzahl.Verwenden Sie die in Delphi vordefinierte Function odd(Z),um festzustellen,ob die Zahl Z ungerade ist,dann odd=true.)

f)Schreiben Sie eine Methode

```
procedure TZiffernListe.Addition(Zahl:TZiffernListe);
```

die eine Binärzahl Zahl vom Typ TZiffernListe zur (Binär-) Zahl,die in TZiffernliste selber gespeichert ist, hinzuaddiert,und das Ergebnis in der Liste TZiffernListe speichert.

Berücksichtigen Sie dabei die in den Stellen auftretenden Überträge.Benutzen Sie dazu eine fertige Unterprocedure

```
Procedure Stelle(Ziffer1,Ziffer2:TZiffer;var Neuziffer,Uebertrag:TZiffer);
```

die zwei (Binär-)Ziffern mit Übertrag addiert.

Es soll zunächst vorausgesetzt werden,dass beide Listen nämlich die Liste TZiffernliste und Zahl dieselbe Anzahl von Elementen haben.(dass beide Zahlen dieselbe Stellenzahl haben.)

g)Erläutern Sie die Wirkungsweise der Unterprocedure Stelle mit dem folgenden Quelltext:

```
  procedure Stelle(Ziffer1,Ziffer2:TZiffer;var Neuziffer,Uebertrag:TZiffer);
  begin
     Neuziffer:=(Ziffer1+Ziffer2+Uebertrag) mod 2;
     Uebertrag:=(Ziffer1+Ziffer2+Uebertrag) div 2;
   end;
```

Zeigen Sie,dass alle Fälle,die bei der Addition von zwei Binärzifferstellen auftreten können,berücksichtigt werden.
Wie muß die Procedure geändert werden,damit sie für die Addition von Dezimalzahlenstellen geeignet wäre?

h)Damit auch Binärzahlen ungleicher Stellen-/Elementezahl (Listen mit unterschiedlicher Anzahl) addiert werden können,muß die Liste mit der kleineren Stellen-/Elementezahl durch Hinzufügen von Nullen auf die gleiche Stellenzahl wie die längere Liste gebracht werden.
Schreiben Sie dazu eine Methode

```
procedure TZiffernliste.GleicheLaenge(Zahl:TZiffernliste)
```

die je nachdem welches Objekt TZiffernliste (self) oder Zahl die kleinere Elementeanzahl hat,bei diesem Objekt entsprechend viele Nullen hinzufügt.Benutzen Sie dazu, die (nur) für diese Teilaufgabe zusätzlich vorgegebene Methode

```
procedure TListe.AmEndeAnfuegen(Ob:TObject);
```

i)Schreiben Sie eine Hauptprogramm BinaerZahlen (als Textanwendung),das zwei (Binär-) Zahlen Zahl1 und Zahl2 als TZiffernliste einliest,die Parität beider Zahlen berechnet und jeweils ausgibt,sowie die Summe beider Zahlen berechnet und anzeigt.

Der Gesamtquelltext und die Lösungen dieser Klausur sind im Anhang dieser Arbeit aufgelistet und das Programm ist auch auf der Installations-CD vorhanden.

Die Klausur vollzieht im ersten Teil noch mal den im Unterricht besprochenen Stoff an einem ähnlichen Beispiel nach,wobei hier allerdings abweichend ein Element mit Zifferninhalt zu erzeugen ist.(Taxonomie:Wissen)

Im zweiten (anspruchsvolleren Teil ab Aufgabe d)) sind die vorgegebenen bekannten Objektmethoden auf ein bisher nicht im Unterricht besprochenes Problem anzuwenden.Dieser Teil erfordert zur Lösung Transfer des bisher Gelernten und teilweise auch kreatives Denken (Taxonomie:Synthese und Analyse).

Nachdem jetzt grundlegende Techniken im Umgang mit Objekten, die Programmierung einer Benutzeroberfläche mittels ereignisorientierten Methoden und vorgegebenen Komponenten in Delphi,sowie die Struktur einer wiederverwendbaren Objektliste bekannt sind,kann nun im Unterricht mit dem Aufbau einer objektorientierten Graphenstruktur begonnen werden.

4)Der Aufbau eines Graphen als Programmierungsprojekt

(Konzeption CAK:siehe Einleitung)

Der Quellcode des Projekts gliedert sich in die vier Units UListC,UGraphC, UInhgrphC,UMathC und das Hauptprogramm Crtapp.Der Buchstabe C steht dabei jeweils für Consolen- d.h. Text-Anwendung (textorientierte Ein-und Ausgabe).

Die Unit UGraphC ist eine reduzierte Version der Unit UGraph der graphikorientierten Entwicklungsumgebung des Programms Knotengraphs (EWK), die nur die zum Ablauf der in der Unit UMathC enthaltenen ausgewählten Graphenalgorithmen nämlich geschl./offene Eulerlinie, Graphenfärbungen (Färbung der Knoten) sowie tiefer Baumdurchlauf und Minimaler Pfad als auch Alle Pfade zwischen zwei Knoten unbedingt benötigten Methoden enthält.

Sie greift auf eine Unit UListC zurück,in der sich die grundlegenden Listenalgorithmen in Form der Objektstruktur TListe befinden,die in diese Unit ausgelagert wurden.Diese Unit enthält gerade eine Auswahl der im vorigen Abschnitt besprochenen Methoden zum Aufbau einer Objektliste,deren Ele-

mente durch nachträglich (zum Einfügezeitpunkt) vom Anwender ausgewählte Objekte festgelegt werden können.Die Methoden werden lediglich noch ergänzt um die Methoden WertsummederElemente und WertproduktderElemente,deren Bezeichner die Wirkungsweise der Methode beschreiben.

Da der Aufbau und die Verwaltung einer universell verwendbaren Objektliste zuvor (siehe voriger Abschnitt) ausführlich im Unterricht besprochen wurde, und damit den Schülern bekannt ist,kann jetzt das Objekt der wiederverwendbaren Liste vom Typ TListe mittels der Unit UListC den Schülern fertig zur Verfügung gestellt werden.

Deshalb kann diese Unit jetzt auch vereinfacht auf dem schon in Delphi vordefinierten Objekttyp TList aufbauen, und lediglich deren Methodenbezeichner wurden an die zuvor benutzte Notation angeglichen.

Die Unit UInhgrphC erweitert die Knoten- und Kantendatenstruktur des Graphen der Unit UGraphC objektorientiert um jeweils ein Inhaltsfeld zur Aufnahme von Knoten- und Kanteninhalten,und stellt Methoden zu deren Verwaltung bereit.

Das Hauptprogramm Crtapp erstellt ein Menü zum Aufruf der entsprechenden Methoden der Units UInhgrphC und UMathc zur Erstellung eines Graphen und zum Aufruf der genannten Graphenalgorithmen.

Zunächst soll der Quellcode der Unit UGraphC sowie dessen Erarbeitung im Unterricht beschrieben werden.

Die Methoden der Unit bilden,wie schon erwähnt, eine vereinfachte Teilmenge der Methoden der Unit UGraph der Entwicklungsumgebung des Programms Knotengraph,wobei als wesentliche Veränderung alle Aufrufe des Ausgabedialogfesters Showmessage durch eine entsprechende Writeln-Anweisung ersetzt wurden.Außerdem wurde die Ausgabeverwaltung durch die Property Wert vereinfacht,da jeweils nur ein Datenfeld zu berücksichtigen ist (keine Wertlistenverwaltung).

Die Unit UGraphC (bzw. die Unit UGraph) enthält also die grundlegende objektorientierte Datenstruktur eines Graphen, ohne daß Knoten-und Kanteninhalte berücksichtigt werden.

Der Aufbau der Unit UGraphC

Vor Beginn jeglicher Quellcodeerstellung ist zunächst die grundlegende Struktur eines Graphen zu erörtern.Hier kann jetzt auf den Vorkenntnisse des Einführungsprojektes,auf das an dieser Stelle natürlich zurückgegriffen werden sollte, aufgebaut werden.

Falls dieses Beispiel nicht in dieser Weise durchgeführt wurde (z.B. weil schon Kenntnisse objektorientierter Datenstrukturen anderweitig vorhanden waren),ist nach der Reflektion einiger Beispiele zu Graphen wie Straßen- oder Wegesysteme bzw. Wasserrohr-oder Elektrizitätsleitungssysteme usw. den Schülern sofort klar,dass ein Graph aus den verschiedenen Objekten Knoten und Kanten besteht.

Außerdem lassen sich Begriffe,soweit noch nicht aus dem Einstiegsprojekt, bekannt,wie ungerichteter und gerichteter Graph, Pfade, Pfadrichtung,schon besuchte Kanten oder Knoten, Anfangs-und Endknoten und Quell-bzw. Zielknoten sowie Schlingenkanten durch die im vorigen Absatz erwähnten Beispiele anschaulich klären,ohne dass diese Begriffe unbedingt im mathematisch strengen Sinne definiert werden müssen.

Es wird im folgenden davon ausgegengen,dass der Unterrichtsgang gemäß den bisherigen Ausführungen dieses Kapitels erfolgt ist.Da die grundlegenden Objekte Kanten und Knoten sind,die dem Graphen neu hinzugefügt und evtl. in ihm wieder gelöscht werden,und unter Berücksichtigung des schon im Unterricht zuvor behandelten Themen Listenverwaltung sowie der aus dem Einstiegsprojekt bekannten noch bezüglich der Array-Felder zu verbessernden Graphenstruktur von TGraph (falls in dieser Weise behandelt: siehe die

dortige Methode TGraph.Knotenloeschen),liegt der Vorschlag nahe,diese Objekte jetzt in einer (Objekt-) Liste zu verwalten.Dies ergibt folgende Definition (Listen anstelle der Arrays):

(vgl. dazu auch den entsprechenden Unterrichtsplan im Anhang)

```
type

TGraph = class(TObject)

    Knotenliste_:TKnotenliste;
    Kantenliste_:TKantenliste;
.
.
.
end;
```

Diese Definition lässt sich unter Berücksichtigung des Kapselungsprinzips der objektorientierten Programmierung mittels Propertys sowie unter Ergänzen sinnvoller Methoden (die teilweise schon im Einstiegsprojekt vorhanden waren) erweitern zu folgender Deklaration:

```
type

TGraph = class(TObject)
  private
    Knotenliste_:TKnotenliste;
    Kantenliste_:TKantenliste;
    function WelcheKnotenliste:TKnotenliste;
    procedure SetzeKnotenliste(Kl:TKnotenliste);
    function WelcheKantenliste:TKantenliste;
    procedure SetzeKantenliste(Kl:TKantenliste);
  public
    property Knotenliste:TKnotenliste read WelcheKnotenliste Write SetzeKnotenliste;
    property Kantenliste:TKantenliste read WelcheKantenliste Write SetzeKantenliste;
    constructor Create;virtual;
    procedure Free;
    procedure Freeall;
    function  Leer:Boolean;
    procedure KnotenEinfuegen(Kno:TKnoten);
    procedure Knotenloeschen(Kno:TKnoten);
    procedure KanteEinfuegen (Ka:TKante;Anfangsknoten,EndKnoten:TKnoten;
    gerichtet:Boolean);
    procedure Kanteloeschen(Ka:TKante);  {****}
    procedure LoescheKantenbesucht;      {****}
    procedure LoescheKnotenbesucht;      {****}
    function AnzahlKanten: Integer;      {****}
    function AnzahlSchlingen:Integer;    {****}
    function AnzahlKantenmitSchlingen:Integer; {****}
    function AnzahlKnoten:Integer;             {****}
  end;
```

Die mit {****} bezeichneten Methoden sind im ersten Ansatz zum Aufbau und zur Verwaltung des eigentlichen Graphen nicht unbedingt nötig und können deshalb zunächst weggelassen werden und bei Bedarf später eingefügt oder auch vom Lehrer dann fertig vorgegeben werden.Die Notwendigkeit der anderen Methoden ist sofort einsichtig.Sie können auch als Methoden exemplarisch für alle anderen behandelt werden.

Außerdem werden an Hand dieser Deklaration die in Delphi möglichen Propertys sowie die dazugehörenden Property-Methoden read und write erneut benutzt (sie sind den Schülern schon aus Einstiegsprojekt in Abschnitt 2 bekannt),um einerseits dem Kapselungsprinzip der Objekte gerecht zu werden (Zugriff auf die Datenfelder nur mittels Methoden),aber um andererseits gleichzeitig eine vereinfachte Syntax als Zugriff auf ein Datenfeld verwenden zu können.
(z.B. Graph.Knotenenliste:=... benutzt die Methode SetzeKnotenliste, um das Feld Knotenliste_ zu verändern)

In der bisherigen TypDeklaration sind jetzt schon die noch zu deklarierenden neuen Objekte nämlich die Knoten und Kanten als Listen vorhanden.Nun ist noch festzulegen,was die Typen TKnotenliste_ und TKantenliste_ bedeuten sollen.

Wenn jetzt in einer Unterrichtsgruppe noch einmal in Erinnerung gerufen wird,dass die Knoten und Kanten,d.h. die Elemente der Knoten- und Kantenliste vom späteren Benutzer durch Vererbung mit beliebigen Datenfeldern erweitert werden sollen,ist für Schüler sofort als Transfer von der zuvor im Unterricht behandelten Objektliste einsichtig, Kanten-und Knotenlisten in der gleichen Weise zu deklarieren.Knoten und Kanten sind also gemäß des im Unterricht (voriger Abschnitt) besprochenen Projekts Liste als Objekt als Teillisten einer Liste ohne Datenfeld,d.h. als Listenzeiger der Objektliste aufzufassen,deren Objekten nachträglich durch Vererbung die Eigenschaft des Elements (Knoten bzw. Kanten) zugewiesen werden kann.

Dieses ergibt folgende Typdeklarationen:

```
TKantenliste = class(TListe)
   .
   .
 end;

TKante =  class(TKantenliste)
 .
 .
end;

TKnotenliste = class(TListe)
   .
   .
 end;

TKnoten =  class(TKnotenliste)
 .
 .
end;
```

An dieser Stelle bietet es sich jetzt an,den Zugriff auf die einzelnen Listenelemente,der ja für die Verwaltung des Graphen von entscheidender Bedeutung ist,mit Hilfe der komfortablen Möglichkeiten von Delphi einfacher zu gestalten.Es erweist sich jetzt als Vorteil,dass sich TListe von dem in Delphi vordefinierten Datentyp TList ableitet,wodurch mittels der erweiterten Property-Syntax von Delphi folgende Deklaration möglich wird,die es gestattet auf die Elemente der Listen Kanten bzw. Knoten als Array-Element nämlich mittels Kantenliste(Index) bzw. Knotenliste(Index) zuzugreifen,so dass sich die Vorteile einer dynamischen Objektliste mit der des Arrays verbinden lassen:

```
TKantenliste = class(TListe)
    function Kante(Index:Integer):TKante;
    property Items[Index:Integer]:TKante read Kante;
end;

TKnotenliste = class(TListe)
    function Knoten(Index:Integer):TKnoten;
    property Items[Index:Integer]:TKnoten read Knoten;
 end;
```

Diese Vereinfachung entspricht dem im Projekt Liste als Objekt (siehe voriger Abschnitt) realisierten Zugriff Element(Index) bzw. Objekt(Index) bzw. Liste(Index),ist daher der Art nach prinzipiell schon bekannt und muß an dieser Stelle gemäß der veränderten Deklarationssyntax den Schülern natürlich als Lehrervorschlag mitgeteilt werden.

(So läßt sich auf die Knoten und Kanten indiziert wie im Einstiegsprojekt zugreifen.)

Die bei TKante und TKnoten zu ergänzenden weiteren Typdeklarationen wurden durch die Deklaration des Typs TGraph schon teilweise vorbestimmt bzw. sind anschaulich an Hand der Graphenstruktur sofort einsichtig,so dass sie von Schülern jetzt leicht ergänzt werden können(vgl.auch die Kenntnisse aus dem Einführungsprojekt):

1)Jede Kante besitzt einen Anfangs-und Endknoten.Zu jeder Kante muß angegeben werden,ob sie gerichtet oder ungerichtet ist, und außerdem ist bei Einbindung in einen Pfad die Angabe der Pfadrichtung in Form des Zielknotens

sowie die Angabe,ob die Kante schon besucht oder nicht besucht worden ist,sinnvoll.

2)Von jedem Knoten gehen (evtl. gerichtete) Kanten aus bzw. laufen in ihn ein.Es ist daher sinnvoll,dass die Objektklasse des Knotens zwei Objektlisten in Form der ein-und auslaufenden Kanten,d.h. also vom Typ TKantenliste enthält.Eine gerichtete Kante gehört also sowohl zur auslaufenden Kantenliste des Quellknotens als auch zur einlaufenden Kantenliste des Zielknotens.Im ungerichteten Fall ist eine Kante sowohl ein-bzw. auslaufend,so dass sie in beide Kantenlisten sowohl des Quell- als auch des Zielknoten eingefügt wird.Im Hinblick auf spätere Pfadalgorithmen ist die Hinzufügung eines Datenfeldes Besucht(_) sinnvoll.

(Das Einfügen eines Verweises auf den Gesamtgraph in TKnoten in Form des Feldes Graph(_) ist nützlich,um von überall Zugriff auf die Gesamtstruktur zu haben.)

Dies alles führt zu folgender Typdeklaration,die in ca. zwei Unterrichtsstunden mit den Schülern erarbeitet werden konnte:

```
TKantenliste = class(TListe)
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    function Kante(Index:Integer):TKante;
    property Items[Index:Integer]:TKante read Kante;
  end;

  TKante =  class(TKantenliste)
  private
    Anfangsknoten_:TKnoten;
    EndKnoten_:TKnoten;
    Pfadrichtung_:TKnoten;
    Gerichtet_:Boolean;
    Besucht_:Boolean;
    function WelcherAnfangsknoten:TKnoten;
    procedure SetzeAnfangsknoten(Kno:TKnoten);
    function WelcherEndknoten:TKnoten;
    procedure SetzeEndknoten(Kno:TKnoten);
    function WelchePfadrichtung:TKnoten;
    procedure  SetzePfadrichtung(Kno:TKnoten);
    function Istgerichtet:Boolean;
    procedure Setzegerichtet(G:Boolean);
    function istbesucht:Boolean;
    procedure SetzeBesucht(B:Boolean);
    public
    property Anfangsknoten:TKnoten read WelcherAnfangsknoten
    Write SetzeAnfangsknoten;
    property Endknoten:TKnoten read WelcherEndknoten Write SetzeEndknoten ;
    property Pfadrichtung:TKnoten read WelchePfadrichtung Write SetzePfadrichtung ;
    property Besucht:Boolean read Istbesucht Write Setzebesucht ;
    property Gerichtet:Boolean read Istgerichtet Write Setzegerichtet;
    constructor Create;virtual;
    procedure Free;
    procedure Freeall;
    function Wertlesen:string;virtual;abstract;
    procedure Wertschreiben(s:string);virtual;abstract;
    property Wert:string read Wertlesen Write Wertschreiben;
    function Zielknoten(Kno:TKnoten):TKnoten;
    function Quellknoten(Kno:Tknoten):Tknoten;
    function KanteistSchlinge:Boolean;
  end;

TKnotenliste = class(TListe)
    constructor create;
    procedure Free;
    procedure Freeall;
    function Knoten(Index:Integer):TKnoten;
    property Items[Index:Integer]:TKnoten read Knoten;
  end;

  TKnoten = class(TKnotenliste)
    private
    Graph_:TGraph;
    EingehendeKantenliste_:TKantenliste;
    AusgehendeKantenliste_:TKantenliste;
    Besucht_:Boolean;
```

```
    function WelcherGraph:TGraph;
    procedure SetzeGraph(G:TGraph);
    function WelcheEingehendeKantenliste:TKantenliste;
    procedure SetzeEingehendeKantenliste(Kl:TKantenliste);
    function WelcheAusgehendeKantenliste:TKantenliste;
    procedure SetzeAusgehendeKantenliste(Kl:TKantenliste);
    function Istbesucht:Boolean;
    procedure Setzebesucht(B:Boolean);
    public
    property Graph:TGraph read WelcherGraph Write setzeGraph;
    property EingehendeKantenliste:TKantenliste
    read WelcheEingehendeKantenliste Write SetzeEingehendeKantenliste;
    property AusgehendeKantenliste:TKantenliste
    read WelcheAusgehendeKantenliste Write SetzeAusgehendeKantenliste;
    property besucht:Boolean read Istbesucht Write setzebesucht;
    constructor Create;virtual;
    procedure Free;
    procedure Freeall;
    function Wertlesen:string;virtual;abstract;
    procedure Wertschreiben(S:string);virtual;abstract;
    property Wert:string read Wertlesen Write Wertschreiben;
  end;
```

Sie ergibt sich automatisch aus den obigen Überlegungen unter Berücksichtigung des Prinzips der Objektkapselung mit Hilfe von Propertys.

An dieser Stelle kann jetzt im Unterricht das Prinzip der virtuellen Methode wiederholt werden,und unter Berücksichtigung der Forderung,dass den Knoten und Kanten später mittels Vererbung ein (oder in der Entwicklungsumgebung Knotengraph sogar beliebig viele) Dateninhaltsfeld(er) hinzugefügt werden sollen,können jetzt schon entsprechende Vorbereitungen für den Zugriff auf dieses/diese Feld(er) in Form der folgenden Methoden und der Property Wert getroffen werden.

```
function Wertlesen:string;virtual;abstract;
procedure Wertschreiben(S:string);virtual;abstract;
property Wert:string read Wertlesen write Wertschreiben;
```

Durch die abstrakte Methodendeklaration braucht hier noch kein Quellcode für die Methode erstellt zu werden.Er wird vom späteren Anwender für seine Zwecke geschrieben.(Entsprechende Vorkenntnisse zu abstracten und virtuellen Methoden sind bei den Schülern wieder aus dem Einstiegsprojekt vorhanden,siehe:die Methoden Zeichnen und Loeschen von TFigur.) Die Property Wert steht auf Grund ihrer abstracten Definition nun ab sofort zur Einbindung in Programmcode zur Verfügung.

Nach der ausführlichen Besprechung der Datenstruktur des Graphen kann die Implementation der Methoden weitgehend als Schülerprojekt organisiert werden,wobei verschiedene Gruppen verschiedene Aufgabenteile übernehmen.Lediglich die komplizierteren Methoden von TGraph Knotenloeschen, KanteEinfuegen und Kanteloeschen sollten im Lehrer-Schüler-Dialog zuvor gemeinsam an der Tafel entwickelt werden.Hier stellt sich nämlich das Problem,zugleich mit dem zu löschenden oder einzufügenden Objekt in einer Objektliste noch andere Objekte in anderen Objektlisten löschen oder einfügen zu müssen,was in einem Schülerentwurf möglicherweise zunächst übersehen wird.

Eine kommentierte Übersicht über die Implementation der oben genannten Methoden der Unit UGraphC wird im Anhang dargestellt.

Darüber hinaus befindet sich der gesamte Quellcode des Programms als Listing im Anhang und ist auch auf der Installations-CD vorhanden.

Die Erstellung dieser Unit einschließlich Vorbereitung der Datenstruktur läßt sich in ca. 9 Unterrichtsstunden durchführen.Bevor die Methoden ausgetestet werden können,ist es nötig in einer zweiten aufbauenden Unit den Knoten und Kanten Dateninhalte durch Vererbung zuzuweisen,da sonst keine Ausgabe-oder Anzeige der Knoten und Kanten erfolgen kann.Dies geschieht durch Erweiterung der Objektstruktur in der Unit UInhgrphC.

Will man möglichst schnell zur Progranmierung mathematischer Algorithmen (Abschnitt 5 und 6 dieses Kapitels) gelangen,kann man die Unit UGraphC zusammen mit der Unit UListC den Schülern auch unter Besprechung der Bedeutung der enthaltenden Methoden und Strukturen fertig zur Verfügung stellen und erst mit der Programmierung der Unit UInhgrphC einsteigen.Dieser Weg kann insbesondere dann beschritten werden,wenn das Einstiegsprojekt Programmierung eines Graphen (Abschnitt 2) und das Projekt Liste als Objekt (Abschnitt 3) gründlich besprochen bzw. programmiert worden sind,weil dann den Schülern die prinzipielle Konzeption der Unit UGraphC schon bekannt ist.Das gleiche empfiehlt sich auch,wenn der Aufbau der Graphenstruktur den Schülern deshalb möglichst rasch bekannt gemacht,geübt und angewendet werden soll,um zur Konzeption EWK (Abschnitt 7 sowie Kapitel C) überzugehen.(Diese Konzeption kann allerdings auch ganz ohne Behandlung der der textorientierten Version begonnen werden.)

Die Unit UInhgrphC

Die Erzeugung eines Knoten bzw. einer Kante mit Datenfeld zur Aufnahme eines Inhalts kann jetzt analog der als Vorbereitung besprochenen Objektdeklaration im Einführungsprojekt (s.o.) erfolgen,und die Datenstruktur kann deshalb von den Schülern sofort entwickelt werden:

```
TInhaltsknoten = class(TKnoten)
    Inhalt_:string;
   ...
  end;

TInhaltskante = class(TKante)
    Inhalt_: String;
    ...
  end;
```

In der Unit UGraphC wurde schon die Ausgabe mit Hilfe der Property Wert durch die virtuellen und abstrakten Methoden Wertschreiben und Wertlesen vorbereitet,deren Quellcode jetzt für diese Datentypen festgelegt werden muß.

Sinnvollerweise werden jetzt auch noch zwei Methoden hinzugefügt,die es gestatten,den Inhalt der Datenfelder für Knoten und Kante einzugeben:

```
Knotenwerteinlesen und Kantenwerteinlesen
```

Außerdem ist für spätere Anwendungen eine Methode zur Ausgabe von Pfadknoteninhalten (Werten) in geordneter Pfadknotenreihenfolge (Durchlaufrichtung des Pfades) notwendig,wobei Pfade als Kantenlisten dargestellt werden können.Die Methode Listenausgabe kann als Methode von TInhaltskante implementiert werden,da der Zeiger auf eine Kante,d.h. ein Kantenlistenelement identisch ist mit dem Zeiger auf die Restliste der Kantenliste.

Um die neuen Knoten-und Kantentypen nutzen zu können,muß jetzt noch ein Nachfolgertyp von TGraph,nämlich TInhaltsgraph definiert werden.Dieser Typ muß Methoden (Knotenerzeugenundeinfuegen, KnotenausGraphloeschen, Kanteerzeugenundeinfuegen,KanteausGraphloeschen) bereitstellen,um unter Verwendung der Methoden von TInhaltsknoten und TInhaltskante die Knoten und Kanten neuen Datentyps in den Graph einzufügen oder aus ihm zu löschen.Sinnvollerweise greift man dabei auf die Vorgängermethoden von TGraph zurück.Um das deutlich zu machen,und weil das Einfügen und Löschen von Kanten wegen der Überprüfung,ob ihre Randknoten schon oder noch nicht im Graph vorhanden sind,komplexer als das Einfügen und Löschen der Knoten ist,werden die Zusatzmethoden FugeKanteein und LoescheKante als Zwischenschritte zusätzlich implementiert.

Es empfiehlt sich die beiden letztgenannten komplexen Methoden im fragendentwickelnden Lehrer-Schüler-Unterrichtsgespräch zu entwickeln,während die Erstellung aller anderen Methoden wieder als reines Schülerprojekt nach einer ausführlichen Besprechung der Datenstruktur gestaltet werden kann.Hinzugefügt werden sollte durch die Schüler außerdem noch eine Methode

ZeigeGraph,die die Knoten und Kanten des Graphen mit Hilfe ihrer Inhalte auf dem Bildschirm (als Textausgaben) darstellt.

Die Methode Graphknoten dient dazu,zu einem vorgegebenen Inhalt (der in dem Knoten Kno als Inhalt (Wert) gespeichert ist) einen (Zeiger auf einen) Knoten des Graphen zu suchen.(Auch diese Methode ist in analoger Form schon aus dem Einstiegsprojekt bekannt.)

Dies ergibt folgende Typdeklaration:

```
type
  TInhaltsknoten = class(TKnoten)
  private
    Inhalt_:string;
    procedure Wertschreiben(S:string);override;
    function Wertlesen:string;override;
  public
    constructor Create;override;
    procedure Free;
    procedure Freeall;
    procedure KnotenWerteinlesen;
  end;

  TInhaltskante = class(TKante)
  private
    Inhalt_: String;
    procedure Wertschreiben(S:String);override;
    function Wertlesen:string;override;
  public
    constructor Create;override;
    procedure Free;
    procedure Freeall;
    procedure Kantenwerteinlesen;
    procedure Listenausgabe;
  end;


  TInhaltsgraph      =     class(TGraph)
  public
    constructor Create;override;
    procedure Free;
    procedure Freeall;
    function  Graphknoten(Kno:TInhaltsknoten):TInhaltsknoten;
    procedure Knotenerzeugenundeinfuegen;
    procedure KnotenausGraphloeschen;
    procedure FuegeKanteein(Kno1,Kno2:TInhaltsknoten;gerichtet:Boolean;
    Ka:TInhaltskante);
    procedure Kanteerzeugenundeinfuegen;
    procedure LoescheKante(Kno1,Kno2:TInhaltsknoten;Ka:TInhaltskante);
    procedure KanteausGraphloeschen;
    procedure ZeigeGraph;
  end;

  function Bewertung(X:TObject):Extended;
```

Als Bewertungsfunktion wird schließlich sinnvollerweise die Zuordnung der Knoten oder Kanten zu ihrem numerischen Inhalt,umgewandelt in eine Real-(Extended-)Zahl definiert.

Eine kommentierte Übersicht über die Implementation der oben genannten Methoden der Unit UInhgrphC wird im Anhang dargestellt.

Darüberhinaus befindet sich der gesamte Quellcode des Programms als Listing im Anhang und ist auch auf der Installations-CD vorhanden.

Schließlich ist noch ein Hauptgramm zur Erzeugung und Verwaltung des Graphen zu schreiben,das jetzt leicht z.B. von einer Schülergruppe alleine als Teil des Projektes erstellt werden kann:

```
program Crtapp;

uses WinCrt,
  SysUtils,
  Ugraphc in 'UGRAPHC.PAS',
  UInhgrphC in 'UINHGRPHC.PAS',
  UMathc in 'UMATHC.PAS',     {****}
```

```
Ulistc in 'ULISTC.PAS';

 Var Auswahl,s:string;
     Wahl:Integer;
     Graph:TInhaltsgraph;
begin
  Strcopy(WindowTitle, 'Programm Knotengraph');
  Windowsize.x:=800;
  Windowsize.y:=600;
  Screensize.x:=100;
  Screensize.y:=35;
  Initwincrt;
  Graph:=TInhaltsgraph.create;
  repeat
    repeat
      Clrscr;
      Graph.ZeigeGraph;
      Writeln;
      Writeln('Menü:');
      Writeln;
      Writeln('Knoten einfügen......1');
      Writeln('Knoten löschen.......2');
      Writeln('Kante einfügen.......3');
      Writeln('Kante löschen........4');
      Writeln('Neuer Graph..........5');
      Writeln('Eulerlinie...........6');    {****}
      Writeln('Hamiltonkeis.........7');    {****}
      Writeln('Graph faerben........8');    {****}
      Writeln('Tiefe Baumpfade......9');    {****}
      Writeln('Minimaler Pfad......10');    {****}
      Writeln('Programm beenden.....q');
      Writeln;
      Write('Welche Auswahl......?');
      Readln(Auswahl);
      Writeln;
      Wahl:=StringtoInteger(Auswahl);
    until  ((Wahl IN [1..10])or (Auswahl='q')or (Auswahl='Q'));
    Case Wahl of
      1:Graph.Knotenerzeugenundeinfuegen;
      2:Graph.KnotenausGraphloeschen;
      3:Graph.Kanteerzeugenundeinfuegen;
      4:Graph.KanteausGraphloeschen;
      5:Graph:=TInhaltsgraph.create;
      6:TEulerCgraph(Graph).Menu;            {****}
      7:THamiltonCgraph(Graph).Menu;         {****}
      8:TFarbCgraph(Graph).Menu;             {****}
      9:TPfadCgraph(Graph).Menu1;            {****}
      10:TPfadCgraph(Graph).Menu2;           {****}
      else if (Auswahl<>'Q') and (Auswahl<>'q') then Writeln('Eingabe wiederholen');
    end
  until (Auswahl='Q')or (Auswahl='q');
  Writeln;
  Writeln('Zum Beenden Taste drücken!');
  Readln;
  Graph.Freeall;
  Graph:=nil;
  Donewincrt;
end.
```

Die mit {****} bezeichneten mathematischen Anwendungen werden erst später hinzugefügt.

Als Unterrichtszeit für diesen Projektabschnitt können ca. 12 Unterrichtsstunden veranschlagt werden.

Wenn man zu Beginn der Jahrgangsstufe 12.1 im Informatikunterricht mit dem Einstiegsprojekt des Abschnittes 2) dieses Kapitels beginnt,kann man die bisher geschilderten Unterrichtssequenz Aufbau und Testen der Graphenstruktur nach der Konzeption CAK am Ende dieses Halbjahres beendet haben,so dass man in der Jahrgangsstufe 12.2 mit der Besprechung und Programmierung von mathematischen Anwendungen nach den Konzeptionen CAK oder EWK (Abschnitt 5 und 6 oder Abschnitt 7) beginnen kann.Dies gilt insbesondere dann,wenn man sich bei der Programmierung der Unit UGraphC,wie oben beschrieben,auf exemplarische Methoden beschränkt oder aber diese Unit den Schülern bei Zeitknappheit bzw. wenn man mehr auf Anwendungen den Schwerpunkt setzen möchte unter Erläuterung deren Methoden und Strukturen fertig zur Verfügung stellt,was auf der Grundlage einer gründlicher Besprechung des Einstiegsprojekts (Abschnitt 2) ohne weiteres möglich ist.

5)Mathematische Anwendungen auf der Grundlage der Graphenstruktur mit der Konzeption CAK

Nachdem bisher der informationstechnische Aspekt in Form des objektorientierten Aufbaus einer vererbbaren,d.h. somit offenen und an beliebige Anwendungen anpaßbaren Graphenstruktur im Vordergrund stand,sollen jetzt die mathematischen Aspekte von Graphen in Form von algorithmischer Graphentheorie als Unterrichtsgegenstand besprochen werden.

Als Beispiele bieten sich hier die von R. Baumann als unverzichtbar für die Unterrichtsbehandlung genannten 5 Probleme der Graphentheorie,nämlich Euler-Problem, Labyrinthproblem, Hamiltonproblem, Traveling-Salesman-Problem und das Problem des kürzesten Weges an. (vgl. Kapitel A I)

I) Das Hamilton-und das Travelling-Salesman-Problem

Genug Motivation für die Beschäftigung mit Hamiltonlinien als Unterrichtsgegenstand ergibt sich sofort aus der Aufgabenstellung des Handlungsreisenden,der alle ihn interessierenden Geschäfts-Städte auf einer einzigen Rundtour besuchen möchte und wieder zum Ausgangspunkt zurückkehren will,ohne dass er einen Weg doppelt durchfahren muß.Zudem sollte der Rundweg noch die Eigenschaft haben,der kürzeste Weg unter allen Möglichkeiten sein.

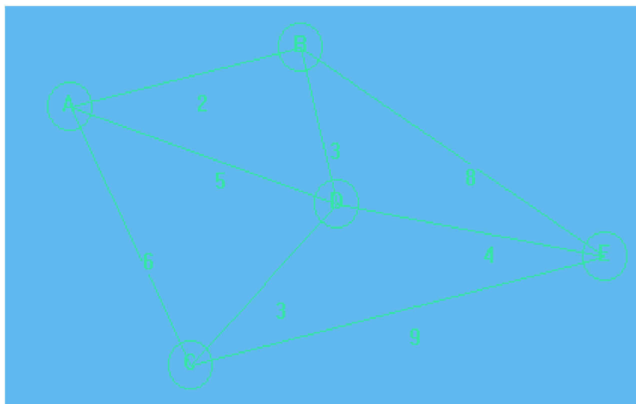
Probleme dieser Art werde heute auch Routing-Probleme genannt und spielen eine wichtige Rolle bei Programmen,die Verkehrsverbindungen in Karten suchen,wobei sich hierbei u.a. auch das weitere Problem des kürzesten Weges zwischen zwei Orten (Knoten) stellt.Sie sind bekannt und benötigen deshalb kaum einer näheren Erläuterung der Aufgabenstellung.

Der Unterrichtsgang gestaltet sich am besten zunächst gemäß der im Kapitel C V Euler-und Hamiltonlinien (siehe dort) skizzierten Unterrichtsgang für Hamiltonlinien unter Einsatz der dortigen Beispiele 6 und 7 (Aufgabe C V 6 und C V 7) und führt bis zur mathematischen Definition des Hamiltonlinienbegriffs (vgl. dazu auch den Unterrichtplan im Anhang).

Einstiegsaufgabe:

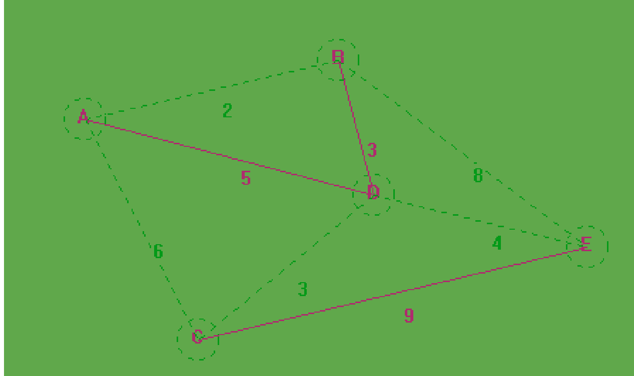
Gesucht wird eine Hamiltonlinie,d.h. ein Kreis,der jeden Knoten genau einmal enthält,in den folgenden beiden Graphen:

Aufgabe C V 6:



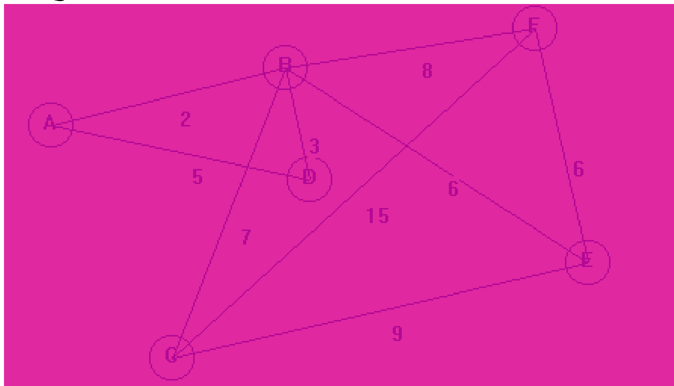
G028.gra

Eine mögliche Lösung:



G028.gra

Aufgabe C V 7:



G029.gra

Lösung:

Es gibt keinen Hamiltonkreis.

Das fertige Programm Knotengraph sollte hier allerdings zunächst nicht zur Lösung der genannten beiden Beispiele eingesetzt,weil es Ziel dieser Unterrichtsabschnitts sein sollte,den Quellcode selber zu erstellen.Die Hamiltonkreise für die genannten Beispiele 6 und 7 werden vielmehr zunächst an Hand einer Zeichnung bzw. per Hand gesucht.

Es stellte sich jetzt zunächst die Frage nach einem Verbalalgorithmus,mit dem alle Hamiltonlinien gesucht werden können.Gleichzeitig sollte an Hand dieses Algorithmus das Backtracking-Verfahren als neues algorithmisches Verfahren bei den Schülern bekannt gemacht werden.

Es bietet sich dazu an das Suchen durch Probieren der Hamiltonkreise in Aufgabe 6 zu systematisieren:

Verfolgt man zunächst einfach einen Pfad an Hand der ausgehenden Kanten eines schon erreichten Knotens immer weiter,bis man keine Möglichkeit mehr hat,als wieder auf einen schon besuchten Knoten zu treffen,ergibt sich beispielsweise als einfachster Rundweg die Knotenfolge ABECA.Dieser Pfad hat jedoch den Nachteil,dass nicht alle Knoten eingebunden werden konnten.Es empfiehlt sich daher,vom Endnoten A zum Knoten C zurückzukehren, und stattdessen von dort aus den Pfad über die zweite ausgehende Kante zum Knoten D zu suchen.Von D kann man dann wiederum über die Kante DA nach A gelangen und hat damit einen möglichen Hamiltonkreis gefunden.Nun kann man,um weitere Hamiltonkreise zu finden,den Pfad wiederum um die Kante DA verkürzen (d.h. zu D zurückgehen) und nach einer anderen Möglichkeit suchen,den Startknoten A zu erreichen.Eine solche Möglichkeit existiert jedoch nicht,da alle anderen Knoten zu denen Kanten führen,schon besucht wurden.Also wird man wiederum den Pfad um die schon durchlaufenden Kante CD

verkürzen,und steht dann beim momentanen Knoten C vor der gleichen Situation wie oben.Bei einer nochmaligen Verkürzung um die Kante EC hat man nun aber die Möglichkeit von E aus über D und C den neuen Gesamtpfad ABEDCA zu finden.(vgl.obige Lösung)

Reflektiert man die Vorgehensweise,geht man,solange sich noch kein Hamiltonkreis erreichen läßt,weil alle vom momentanen Knoten über ausgehende Kanten erreichbaren Zielknoten schon besucht wurden,oder aber zwar der Startknoten vom momentanen Knoten erreicht werden kann,aber noch nicht alle Knoten im bisherigen Pfad eingebunden wurden,über die zuletzt passierte Kante rückwärts und versucht von dem neuen Ausgangsknoten aus einen Pfad über eine Kante zu einem im bisherigen Pfad noch nicht besuchten Knoten zu finden.

Das heißt man probiert durch Rückwärtsgehen,wenn man in eine Sackgasse gekommen ist,und erneutes Vorwärtsgehen auf einen noch unbesuchten Zielknoten zu,systematisch alle möglichen Pfade durch,und gibt,wenn ein zulässiger Hamiltokreis gefunden ist,die Knotenfolge dieses Pfades aus.Das Verfahren bricht ab,wenn alle Zielknoten des Startknotens schon vom Startknoten aus besucht wurden.

Diese Verfahren wird Backtracking-Verfahren genannt und kann auch bei der Lösung von anderen (NichtGraphen-)Problemen erfolgreich angewendet werden.

Es empfiehlt sich daher an dieser Stelle ausgehend von diesem Beispiel ein allgemeines Schema für einen Backtracking-Algorithmus im Unterricht in der folgenden Form zu erarbeiten (und schriftlich als Übersichtsschema festzuhalten):

Allgemeines Schema eines Backtracking-Algorithmus:

Wähle zunächst eine geeignete Größe als Stufenparameter Stufe vom Datentyp Integer.Der Parameter hat die Eigenschaft,dass eine Lösung des gestellten Problems nur dann erreicht wird,wenn Stufe seinen maximalen Wert Max einnimmt.

Lege einen geeigneten Datentyp vom Typ TTyp mit endlich vielen Möglichkeiten von der aktuellen Stufe aus fest, und definiere eine entsprechende Variable Auswahl dieses Typs.

Prinzipieller Methoden-bzw. Procedurenaufbau:

```
procedure backtrack(var Stufe:Integer;evtl. weitere Parameter);
var Auswahl:TTyp;

begin

  Wiederhole

    Wähle die nächste Auswahl vom Typ TTyp auf der aktuellen
    Stufe.

    Ist ein Schritt mit Hilfe dieser Auswahl möglich,um die
    nächste Stufe (Stufe+1) zu erreichen?

    Wenn ja

    begin

      Wähle diese Auswahl und zeichne den Schritt auf.

      Ist dann eine Lösung des Problems gefunden,
      und hat Stufe seinen maximalen Wert Max erreicht?

      dann

        Gib die Lösung aus.
```

```
    andernfalls

      Wenn Stufe<Max dann

        Rufe backtrack(Stufe+1;evtl.weitere Parameter) auf.
        (Dann backtrack!)

    Lösche die Aufzeichnung des vorigen Schrittes
    auf Grund von Auswahl.

  end {wenn ja}

  andernfalls {wenn nein}

 Bis alle Auswahlen auf dieser Stufe ausgewählt.

end;
```

Das Backtracking-Verfahren legt es nahe, rekursive Programmierung einzusetzen,und so den jeweils immer neuen Versuchen des Vorwärtsgehen eine neue Rekursionsebene zuzuorden,die dann beim Rückwärts gehen wieder verlassen wird.Eine Stufenvariable (hier die Anzahl der schon besuchten verschiedenen Knoten) zählt die Anzahl der Ebenen und bestimmt das Finden eines Hamiltonkreises,wenn die Zahl gleich der Zahl der Gesamtknoten ist.

Für das Problem Suchen der Hamiltonkreise ergibt sich so folgender Verbalalgorithmus (vgl. dazu auch das spätere Kapitel C V Euler-und Hamiltonlinien),der als Schema für die Erstellung des Quellcodes schriftlich (z.B. an der Tafel) festgehalten werden sollte:

Algorithmus Hamiltonkreise:

Als Stufe des Backtrackingalgorithmus dient die Anzahl der schon gewählten Knoten des Graphen.

Da eine Kantenfolge nämlich der Hamiltonkreis gesucht wird,werden die schon ausgewählten Kanten in einer Kantenliste gespeichert.

I)

Setze Stufe (Anzahl der gewählten besuchten Knoten) gleich 1. Wähle einen beliebigen Knoten als Start-und Zielknoten.Wähle eine leere Kantenliste zur Aufnahme des Hamiltonkreises.

Rufe mit diesen Parameter die Backtrackingmethode auf.

Backtrackingmethode:

II)

Wähle vom aktuellen Knoten die in der Kantenreihenfolge nächste ausgehenden Kante aus.Wenn alle ausgehenden Kanten des aktuellen Knotens untersucht worden sind,verlasse diese Rekursionsebene der Backtrackingmethode.Wenn es die Anfangsebene ist,beende den Algorithmus.

III)

Prüfe,ob der Wert für Stufe kleiner oder gleich der Anzahl der Knoten (bzw. ob Stufe kleiner als Anzahl+1 ist) des Graphen ist,und der Zielknoten der aktuellen Kante noch nicht als besucht markiert ist.Wenn ja,setze den Algorithmus bei IV) fort.Wenn nein,setze den Algorithmus bei VIII) fort.

IV)

Markiere den Zielknoten als besucht.Füge die Kante in die Kantenliste ein.Erhöhe die Stufenzahl (Zahl der ausgewählten Kanten) um 1.

V)

Prüfe, ob der Zielknoten dieser Kante gleich dem Zielknoten ist,und gleichzeitig der Wert für Stufe gleich der Anzahl der Knoten des Graphen vergrößert um 1 ist sowie Stufe größer als 2 ist.(Dann sind alle Knoten besucht und der Hamiltonkres besteht aus mehr als 2 Knoten.Mit dem Letzteren wird ausgeschlossen,dass ein Hamiltonkreis nur aus dem Durchlaufen zweier paralleler Kanten besteht.)

VI)

Wenn ja,gebe die Lösung aus (oder füge die Kantenliste in die Pfadliste des Zielknotens zur Ausgabe bei Verwendung der graphischen Oberfläche ein vgl. Kapitel C V).Dann setze den Algorithmus bei VIII) fort.Wenn nein,setze bei den Algorithmus bei VII) fort.

VII)

Rufe die nächste Rekursionsebene der Backtrackingprocedure mit dem aktuellen Wert für Stufe (erhöht um 1),dem Zielknoten der aktuellen Kante als aktuellem Knoten und der aktuellen Kantenliste (sowie dem Zielknoten) als Parameter auf.
(Dann backtrack!)

VIII)

Vermindere Stufe (Anzahl der ausgewählten Knoten) um 1,und lösche die letzte Kante aus der Kantenliste.Lösche die Besucht-Markierung des Zielknotens dieser Kante.Setze den Algorithmus bei II) fort.

Nach Ende des Algorithmus sind alle möglichen Hamiltonkreise ausgegeben.

(Bzw. in der Pfadliste des Zielknotens zwecks Ausgabe bei graphischer Oberfläche gespeichert.Wenn diese leer ist,gibt es keine Lösung.vgl Kapitel C)

Wenn das Backtracking-Verfahren den Schülern noch unbekannt ist,wie es in dieser Unterrichtsreihe der Fall war,erfordert die Entwicklung des Algorithmus sowohl in der verbalen Form als auch in der Quellcodeerstellung der gezielten Lenkung durch den Lehrer.

Quellcodeerstellung:

Da es dazu nötig ist,auf die vorigen Methoden von TInhaltsgraph zuzugreifen,da aber andererseits der gerade besprochene Algorithmus jetzt speziell für das Hamilton-Problem (Stufe bedeutet die Anzahl der Knoten) implementiert werden soll,empfiehlt es sich durch Vererbung einen neuen,besonderen Graphtypen

THamiltonCGraph=class(TInhaltsgraph)

zu erzeugen,der den oben genannten Algorithmus in Form der Methode Hamilton enthält.

Außerdem gibt es die Methode Hamiltonkreise,die die Methode Hamilton aufruft.Schließlich stellt die Methode Menu ein Menü zum Aufrufen der Hamiltonkreise zur Verfügung.Die letzten beiden Methoden konnten von den Schülern selbst erstellt werden.

Es entsteht dann folgender Quellcode:

```
THamiltonCgraph = class(TInhaltsgraph)
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    procedure Hamilton(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;
      var Kliste,Salesliste:TKantenliste;var Gefunden:Boolean);
    procedure Hamiltonkreise;
    procedure Menu;
  end;

constructor THamiltonCgraph.Create;
begin
   inherited create;
end;

procedure THamiltonCgraph.Free;
begin
  inherited Free;
end;

procedure THamiltonCgraph.Freeall;
begin
  inherited Freeall;
end;

procedure THamiltonCgraph.Hamilton(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;
var Kliste,Salesliste:TKantenliste;var Gefunden:Boolean);
var Index:Integer;
    Zkno:TInhaltsknoten;
    Ka:TInhaltskante;
    Zaehl:Integer;

begin
  if not Kno.AusgehendeKantenliste.leer then
    for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
    begin
      Ka:=TInhaltskante(Kno.AusgehendeKantenliste.Kante(Index));
      Zkno:=TInhaltsknoten(Kno.AusgehendeKantenliste.Kante(Index).Zielknoten(Kno));
      if ((Not Zkno.besucht) and (Stufe<Self.AnzahlKnoten+1)) or
        ((Zkno=Zielknoten)and (Stufe=Self.AnzahlKnoten)and(Stufe>2) )
      then
      begin
        Ka.Pfadrichtung:=Zkno;
        Zkno.besucht:=true;
        Kliste.AmEndeanfuegen(Ka);
        Stufe:=Stufe+1;
        if (Zkno=Zielknoten)and (Stufe=self.AnzahlKnoten+1)
        then
        begin
          TInhaltskante(Kliste).Listenausgabe;
          {****}
          if (Salesliste.WertsummederElemente(Bewertung)>=
          Kliste.WertsummederElemente(Bewertung))
            and (not Kliste.leer)
          then
          begin
            Gefunden:=true;
            Salesliste.Free;
            Salesliste:=nil;
            Salesliste:=TKantenliste.Create;
            for Zaehl:=0 to Kliste.Anzahl-1 do
              Salesliste.AmEndeanfuegen(Kliste.Kante(Zaehl));
            Writeln;
          end;
          {****}
          end
          else
            Hamilton(Stufe,Zkno,Zielknoten,Kliste,Salesliste,Gefunden);
          Stufe:=Stufe-1;
          if Zkno<>Zielknoten then Zkno.Besucht:=false;
          if not Kliste.Leer then
            Kliste.AmEndeloeschen(TObject(Ka));
      end;
    end;
end;

procedure THamiltonCgraph.Hamiltonkreise;
var Kno:TInhaltsknoten;
    MomentaneKantenliste,Salesliste:TKantenliste;
    Index:Integer;
```

```
 Gefunden:Boolean;
begin
   if Leer then exit;
   MomentaneKantenliste:=TKantenliste.Create;
   LoescheKnotenbesucht;
   Kno:=TInhaltsknoten(self.Knotenliste.Knoten(0));
   Kno.besucht:=true;
   Salesliste:=TKantenliste.create;
   if not Kantenliste.leer then
     for Index:=0 to Kantenliste.Anzahl-1 do
       Salesliste.AmEndeAnfuegen(Kantenliste.Kante(Index));
   Gefunden:=false;
   Hamilton(1,Kno,Kno,MomentaneKantenliste,Salesliste,Gefunden);
   Writeln('Traveling-Salesman-Lösung:');
   Writeln;
   if Gefunden then
      TInhaltskante(Salesliste).Listenausgabe
   else
     Writeln('keine Lösung!');
   MomentaneKantenliste.Free;
   MomentaneKantenliste:=nil;
   Salesliste.Free;
   Salesliste:=nil;
end;

Procedure THamiltonCgraph.Menu;
begin
   if self.leer
   then
   begin
      Writeln('leerer Graph');
      Readln;
      exit;
   end;
  Clrscr;
  Writeln('Hamiltonkreise:');
  Writeln;
  Hamiltonkreise;
  Writeln;
  Readln;
end;
```

(Eine kommentierte Beschreibung des Quellcodes der Methode Hamilton ist in Kapitel C V enthalten.Der Quellcode der dort besprochenen Methode ist bis auf die Ausgabeart mittels der Pfadlisten der Knoten gleich der hier benutzten Methode.)

In der Methode Hamilton wird gleichzeitig das Traveling-Salesman-Problem mit gelöst.Dazu wird eine Kantenliste der Bezeichnung Salesliste als Referenzparameter übergeben,wobei jedesmal,wenn ein Hamiltonkreis neu gefunden wurde,geprüft wird,ob die Kantensumme dieses Kreises kleiner als die Kantensumme des zuletzt gespeicherten Kreises ist.Dann wird dieser Kreis als neue Salesliste gespeichert.

In der einfacheren Version,die nur die Hamiltonkreise sucht,kann der Quellcode zwischen den Markierungen {****} also entfallen (und wird später z.B. als Ergänzung hinzugefügt).Beim Testen der Programme sollten auch solche mit größeren Knotenzahl über 20 Knoten gewählt werden,wodurch evident wird,dass das Verfahren dann kein gutes Laufzeitverhalten hat.Das Problem der NP-Vollständigung kann hier diskutiert werden.

Für diesen Unterrichtsabschnitt (ohne Diskussion des NP-Vollständigkeitsproblem) wurden ca. 8 Unterrichtsstunden gebraucht.

Nach Beenden dieser Sequenz wurde eine erneute Klausur über die im Unterricht behandelten Themen Aufbau der Graphenstruktur (CAK) und Anwendung dieser Struktur zur Implementation von mathematischen Algorithmen geschrieben,deren Aufgabenstellung im folgenden wiedergegegeben ist.

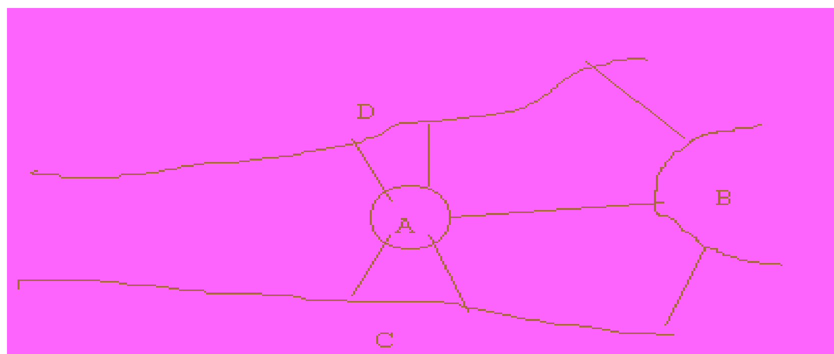
Da sich das Eulerproblem nämlich mittels eines analogen,fast gleichen Backtrackingalgorithmus wie das Hamiltonproblem lösen läßt,wobei die Stufenzahl lediglich die Anzahl der schon besuchten Kanten bedeutet,liegt es nahe dieses Problem als Anwendung der bisher im Unterricht behandelten Inhalte in der Form eines didaktisch-methodisch verwandten Themas,bei dem Transfer möglich ist,als Klausurthema zu stellen.

Eine Ausweitung dieser Aufgabenstellung auf offene Eulerlinien mit ebenfalls verwandter Problemstellung,so dass das Hauptgewicht der Aufgabe den Kategorien Wissen und Transfer und nur ein kleinerer Teil dem Bereich Analyse und Synthese zugeordnet werden kann,bot sich dann als Abituraufgabe an (siehe unten).

(Bei dem Umfang der Abituraufgabe ist zu berücksichtigen,dass im Abitur noch eine zweite Aufgabe aus einem anderem Themenbereich zu stellen ist.)

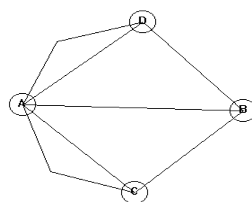
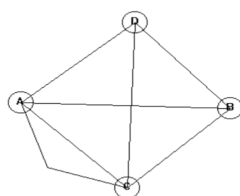
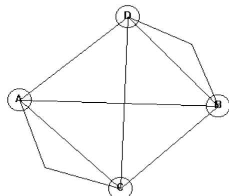
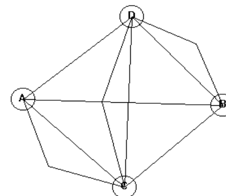
II) Das Eulerproblem

Klausuraufgabe: (Graphenproblem von Euler)



Von dem Mathematiker Euler wurde 1736 die Frage gestellt,ob es über die 7 Brücken in Königsberg,die über den alten und neuen Pregel sowie zu einer im Zusammenfluss der beiden Flüsse gelegenen Insel führen,ein Spaziergang möglich ist,bei dem jede Brücke nur einmal überquert wird und der zum Ausgangspunkt zurückführt.

Graph 1: Graph 2: Graph 3: Graph 4:

Werden die Gebiete,die mit A,B,C und D bezeichnet werden,als Knoten aufgefaßt und die Brücken als Kanten zwischen ihnen,ergibt sich Graph 1.

a)In dem obigen Graphen Graph 1 sind also alle geschlossenen Kantenzüge gesucht,die jeweils alle Kanten genau einmal enthalten.

Falls Kantenzüge mit den gewünschten Eigenschaften existieren,geben Sie einen dieser Kantenzüge als Folge von Knotenbezeichnungen an (z.B.: ACBADBA,dies ist ein falsches Beispiel).

Die Brücken werden jeweils anders angeordnet und auch durch weitere Brücken ergänzt.Den Anordnungen entsprechen jeweils die obigen drei Graphen Graph 2,Graph 3 und Graph 4.Untersuchen Sie ebenfalls,ob für die Graphen 2 bis 4 geschlossene Kantenzüge mit den oben genannten Eigenschaften existieren,und geben Sie bei Existenz einen entsprechenden Kantenzug an.

Definition:

Gibt es in einem Graphen G einen geschlossenen Kantenzug w,der jede Kante von Graph G genau einmal enthält,heißt w geschlossener Eulerweg oder Eulerlinie von G und G ein Eulerscher Graph.

Im folgenden soll untersucht werden,wann eine geschlossene Eulerlinie existiert.

Führen sie dazu mit Graph 3 und 4 folgendes Verfahren durch.Suchen Sie von einem beliebigen Knoten aus einen Kreis im Graphen,und löschen sie die Kanten dieses Kreises aus dem Graphen.

<u>Zeichnen sie den neu entstehenden (um Kanten reduzierten) Graphen neu.</u>

Wiederholen Sie das Verfahren mit dem reduzierten Graphen wiederum in der gleichen Art und Weise, bis sich kein Kreis mehr finden läßt.(Jeweils Zeichnung der reduzierten Graphen)

Welcher Unterschiede zeigen sich zwischen den Graphen 3 und 4?

b)Unter dem Knotengrad eines Knoten versteht man die Anzahl der von ihm ausgehenden bzw. in ihn einlaufenden Kanten.

Geben Sie die Knotengrade aller Knoten von Graph 3 und 4 an.

Es gilt folgender Satz:

Die folgenden Aussagen sind äquivalent:

1)Die Kanten des (zusammenhängenden,ungerichteten) Graphen G lassen in disjunkte (d.h. nicht in den anderen Kreisen enthaltende,nur zu einem Kreis gehörende) Kantenmengen aufteilen,die jeweils einen Kreis bilden.

2)G ist ein (zusammenhängender,ungerichteter) Eulerscher Graph mit geschlossener Eulerlinie.

3)Alle Knoten des Graphen haben geraden Knotengrad.

Begründen Sie,dass aus Aussage 1 die Aussage 2,aus Aussage 2 die Aussage 3 und schließlich aus Aussage 3 die Aussage 1 folgt.

c)Vorgegegeben seien die aus dem Unterricht bekannten Units UListC,UGraphC und UInhgrphC,deren Methoden und Datenstukturen für die nachfolgenden Aufgaben benutzt werden dürfen.

(Eine Übersicht über den Interfaceteil der Unit UGraphC befindet sich im Anhang)

Erläutern Sie die prinzipielle Datenstruktur (d.h. die Objektbeziehungen und die Bedeutung der Datenfelder sowie Popertys) der durch die Unit UGraphC vorgegebenen Objekte.(Eine Beschreibung der Methoden wird nicht verlangt.)

Geben Sie den Quelltext der Methode TGraph.Knoteneinfuegen(Kno:TKnoten) an.

In der Unit UInhgrphC werden die Datentypen TInhaltsknoten und TInhaltskante,die ein zusätzliches Datenfeld Inhalt_ zur Aufnahme eines Knoten- bzw. Kanteninhalts als string besitzen,durch Vererbung von den Typen TKnoten und TKante erzeugt.

Geben Sie entsprechende Typdeklarationen an.

Von TGraph leitet sich der Typ TInhaltsgraph ab,der die Methode TInhaltsgraph.Knotenerzeugenundeinfuegen enthält.Schreiben Sie den Quellcode für diese Methode.

d)Das Objekt TKnoten soll um die Methode

```
function TKnoten.Knotengrad:Integer;
```

die als Resultat den Grad eines Knotens zurückgibt,ergänzt werden.Geben Sie den Quelltext dieser Methode an.
Schreiben Sie eine Methode

```
function TInhaltsgraph.IstEulergraph:Boolean;
```

die gemäß dem obigen Kriterium von Aufgabe c) Aussage 3 unter Benutzung der Methode TKnoten.Knotengrad ermittelt,ob der Graph eine Eulerlinie enthält oder nicht.

e)Im Unterricht wurde durch Vererbung folgender Graphtyp implementiert:

```
  THamiltonCgraph = class(TInhaltsgraph)
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    procedure Hamilton(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;Kliste:TKantenliste);
    procedure Hamiltonkreise;
    procedure Menu;
  end;
```

Mittels der Methode Hamilton (die von Hamiltonkreise aufgerufen wird) konnten alle Hamilton-Kreise des Graphen ermittelt werden.

Schreiben Sie in analoger Weise die Typdeklaration (nur die Typdeklaration ist hier verlangt,nicht der Quellcode von Methoden) eines Objekttyps TEulerCGraph (samt der Deklaration aller notwendigen Methoden),der u.a. eine Methode

```
procedure Euler(Stufe:Integer;Kno:TInhaltsknoten;Kliste:TKantenliste);
```

enthält.Diese Methode ermittelt alle (geschlossenen) Eulerlinien des Graphen nach einem Backtrackingverfahren mit der Zahl der besuchten Kanten als Stufenparameter.

Erläutern Sie die Wirkungsweise aller übrigen Methoden und die Bedeutung der Parameter aller Methoden (einschließlich der Methode Euler).

f)Schreiben Sie zu der Methode TEulerCgraph.Euler den Quellcode.

Setzen Sie dazu eine vorgegebene Methode TInhaltskante.Listenausgabe voraus,die die Kanteninhalte Inhalt_ vom Typ TInhaltskante einer Kantenliste vom Typ TKantenliste als Folge von Knoteninhalten auf dem Bildschirm ausgibt.

Hinweis:Der Quellcode dieser Methode läßt sich durch leichte Veränderungen (allerdings an entscheidenden Stellen) des Quellcodes der Methode THamiltonGraph.Hamilton erzeugen.

g)Geben Sie allgemein (durch verbale Beschreibung) den Aufbau und die Struktur eines Backtracking-Algorithmus an.

Alternativ kann hier auch der Algorithmus Eulerfix (S.242) gestellt werden.

Anhang:Als Anhang wurde der Interfaceteil der Unit UGraphC vorgegeben.

Die Lösungen dieser Klausur befinden sich im Anhang.

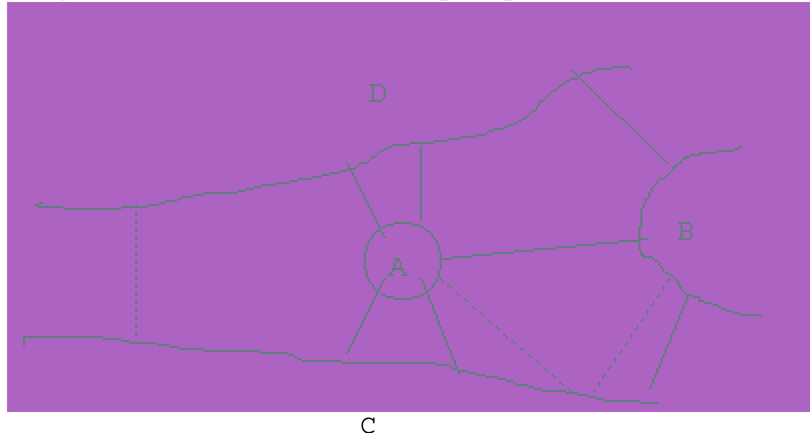
Die Bearbeitungszeit für diese Klausur betrug 3 Stunden.Die Übertragung des Hamilton-Backtracking-Verfahrens auf den Euler-Fall erfordert Transfer eines schon bekannten Verfahrens.Die geführte Erarbeitung eines Kriteriums für die Existenz von Eulerlinien sowie deren Begründung in Teilaufgabe a) und b) reicht in den Bereich kreatives Denken hinein (Analyse und Synthese).Die Aufgabenteile d),e) und f) erfordern Transfer im Unterricht besprochener Methoden,und die Aufgaben c) und g) gehören zur Kategorie Wissen und Verstehen.

Da die Aufgabenstellung sich auf geschlossene Eukerlinien bezog,liegt es nahe das verwandte Problem der offenen Eulerlinien als Abituraufgabe zu stellen,da die darin enthaltenden Aufgabenstellungen dann nicht mehr vollkommen Neuland sind und hauptsächlich Transfer und Wissen und nur noch zum geringeren Teil kreatives Denken erfordern.

Im folgenden ist die Aufgabenstellung der Abituraufgabe wiedergegeben:

Abituraufgabe:

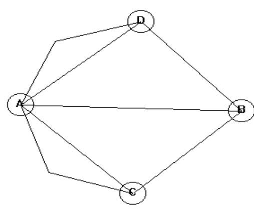
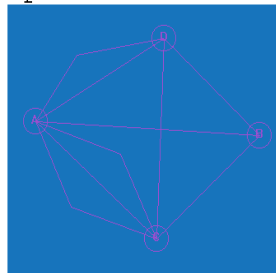
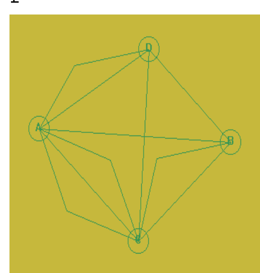
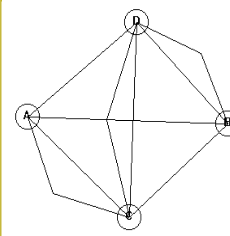
Aufgabe: (Brücken- bzw. Graphenproblem von Euler)



C

Gegeben sind folgende Graphen:

Graph 1: Graph 2: Graph 3: Graph 4:

(Graph 1:ursprüngliches Brückenproblem,Graph 2 bis 4:Erweiterung der Wegmöglichkeiten durch Zusatz-Brücken über den Pregel in Königsberg)

Das Königsberger Brückenproblem läßt sich,wie Ihnen bekannt,auf die Aufgabe reduzieren,ob in dem Graph 1 eine geschlossene Linie existiert,die jede Kante genau einmal durchläuft.Dies ist wie Ihnen ebenfalls bekannt ist,nicht der Fall.Es soll deshalb jetzt die Frage untersucht werden,ob und wann eine offene Eulerlinie existiert.

Definition : Gibt es in einem Graphen G eine offenen Knoten-Kantenfoge w zwischen zwei verschiedenen Knoten des Graphen,die jede Kante von G genau einmal enthält,heißt w offene Eulerlinie von G.

a)Untersuchen Sie daher (durch Probieren),ob die Graphen 1 bis 4 offene Eulerlinien zwischen zwei verschiedenen Knoten besitzen.Falls existent,geben Sie jeweils eine dieser Linien als Folge von Knotenbezeichnungen (z.B.: ACBADB,dies ist ein falsches Beispiel) an.Graph 4 entsteht aus Graph 3 durch Hinzufügen einer Zusatzkante.
Einer der beiden Graphen hat eine geschlossene Eulerline.Wieso folgt daraus die Existenz einer offenen Eulerlinie für den anderen Graph?Wie läßt sich diese Aussage auf beliebige Graphen verallgemeinern?

Unter dem Knotengrad eines Knoten versteht man die Anzahl der von ihm ausgehenden bzw. in ihn einlaufenden Kanten.Bestimmen Sie die Knotengrade der

Graphen 1,2, 3,und 4,und stellen Sie die Ergebnisse in einer Tabelle zusammen:

Knotengrad:

Graph	A	B	C	D
1				
2				
3				
4				

Wodurch unterscheidet sich die Graphen 2 und 3 bezüglich der Knotengrade von den übrigen Graphen?Wie lassen sich die Knoten eines Graphen ermitteln,zwischen denen eine offene Eulerlinie existiert?Welche Eigenschaften haben Graphen,in denen eine offene Eulerlinie existiert?

Es gilt der Satz:
G sei ein zusammenhängender,ungerichteter Graph.Die folgenden beiden Aussagen sind äquivalent:

1)Graph G ist ein Eulerscher Graph mit offener Eulerlinie zwischen zwei Knoten Kno1 und Kno2.

2)Der Grad von genau zwei Knoten Kno1 und Kno2 ist ungerade.Der Grad jedes anderen Knotens in dem Graph G ist gerade.

Beweisen Sie den Satz unter Bezugnahme auf den Satz über die Existenz geschlossener Eulerlinien in Graphen (mittels des Knotengrads).

Vorgegeben seien die aus dem Unterricht bekannten Units UListC,UGraphC und UInhgrphC,deren Methoden und Datenstukturen für die nachfolgenden Aufgaben benutzt werden dürfen.

b)Gegeben sei die folgende Functionsmethode von UGraph:

```
functionTGraph.Wastueich(var Kno1,Kno2:TKnoten):Boolean;
var EKg,ZKg,IE:Boolean;
    Zaehler,Kantenzahl,Index:Integer;
begin
  Zaehler:=0;
  EKg:=false;
  ZKg:=false;
  if not Knotenliste.Leer then
  begin
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kantenzahl:=Knotenliste.Knoten(Index).Kantenzahl
      if odd(Kantenzahl) then Zaehler:=Zaehler+1;
      if (Zaehler=1)and (not EKg) then
      begin
        Kno1:=Knotenliste.Knoten(Index);
        EKg:=true;
      end;
      if (Zaehler=2)and (not ZKg) then
      begin
        Kno2:=Knotenliste.Knoten(Index);
        Zkg:=true;
      end;
    end;
  end;
  if Zaehler=2
  then
    IE:=true
  else
  begin
    IE:=false;
    Kno1:=nil;
    Kno2:=nil;
  end;
  Wastueich:=IE;
end;
```

Analysieren sie den Quelltext,und erläutern Sie,was die Methode insgesamt bewirkt,und wozu Sie im Zusammenhang mit dem Eulerlinienproblem eingesetzt werden könnte.Welchen Bezeichner könnte man der Methode geben?Erläutern Sie Funktionsweise und Progammablauf des Quelltextes.

c)Vorgegeben ist die folgende Objecttypdeklaration:

```
TEulergraph   = class(TInhaltsgraph)
  constructor Create;
  procedure Free;
  procedure Euler(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;
  var Kliste:TKantenliste;);
  procedure BestimmeEulerlinie;
 end;
```

Die Methode

```
procedure TEulergraph.Euler(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;Kliste:TKantenliste);
```

ermittelt die (offenen) Eulerlinien des Graphen zwischen den Knoten Kno und Zielknoten vom Typ TInhaltsknoten.KListe ist dabei eine Kantenliste,die die Eulerlinie als Liste von Kanten aufnehmen soll.Beim Aufruf der Methode ist diese Liste leer.
Schreiben Sie zu dieser Methode den Quellcode nach einem Backtrackingverfahren,das die Zahl der (besuchten) Kanten als Stufenparameter benutzt.Setzen Sie dazu eine vorgegebene Methode TInhaltskante.Listenausgabe voraus,die die Kanten einer Kantenliste vom Typ TKantenliste als Folge von Anfangs-bzw. Endknotenbezeichnern auf dem Bildschirm ausgibt.
Erläutern Sie den prinzipiellen Ablauf eines Backtracking-Algorithmus.

d)Implementieren Sie die Methode BestimmeEulerlinie,die die Methode TEulerGraph.Euler aufruft,um die Eulerlinie,falls existent,zu bestimmen.Benutzen Sie dabei in geeignete Weise die Methode Wastueich (evtl. mit neuem Bezeichner) von Aufgabe c).

Ersetzt man die Units UListC,UGraphC und UInhgrphC durch die Units ULIst,UGraph und UInhgraph läßt sich die Klausur auch als Klausur zur Konzeption EWK (s.u. Abschnitt 6) stellen.
Alternativ oder zusätzlich kann hier auch der Algorithmus Eulerfix (S.242) gestellt werden.Die Lösungen dieser Abiturklausur befinden sich im Anhang.

Auf Grund der den Schülern bekannten Lösung der Klausuraufgabe über geschlossene Eulerlinien, die (als Berichtigung) im Unterricht ausführlich besprochen wurde,kann der Quellcode des Backtrackingalgorithmus für die offene Eulerinie in genau der gleichen Weise erstellt werden,so dass diese Teilaufgabe eigentlich nur Wissen und die Erkenntnis, dass die Lösung übertragbar (Transfer) ist,erfordert. Neu ist nur das (durch die Art der Aufgabenstellung geführte) Entdecken von Kriterien für die Existenz von offenen Eulerlinien,deren allgemeiner Beweis (Analyse und Synthese) sowie die Interpretation des Quellcodes der Methode Wastueich,die die Existenz einer offenen Eulerline in einem Graph an Hand dieser Kriterien untersucht (Transfer und Analyse).Zu der gleichen Kategorie gehört auch die Lösung von Teilaufgabe d).

Beim Umfang der Abituraufgabe ist zu berücksichtigen,dass noch eine zweite Aufgabe mit anderem Inhalt gestellt werden muss.(Dies war eine Aufgabe zum Thema Objekt-Liste,nämlich einen einer Kellerstruktur verwandten ADT objektorientiert zu realisieren.)

Nach Ende dieses Unterrichtsabschnitts hat man jetzt verschiedene Möglichkeiten den Unterrichtsgang fortzusetzen.

Eine Möglichkeit ist es auf der Grundlage des bisher benutzten Objekt-Graphenmodells unter Benutzung der textorientierten Ausgabe (d.h. als Consolenanwendung) weitere mathematische Anwendungsalgorithmen zu erstellen.

Dabei bieten sich die in dem Katalog von Baumann verbliebenen noch nicht behandelten Aufgaben Labyrinthproblem bzw. Problem des kürzesten Pfades (als auch aller Pfade) zwischen zwei Knoten an.

Die Aufgaben dieses Katalogs seien dabei noch ergänzt um das interessante Färbungsproblem der Graphentheorie,d.h. das Suchen der chromatischen Zahl eines Graphen,worunter man die kleinste Anzahl von Farben versteht,mit denen sich die Knoten eines Graphen färben lassen,ohne dass benachbarte Knoten dieselbe Farbe erhalten.Das Problem hängt unmittelbar mit dem Vierfarbenproblem zum Färben von Landkarten zusammen,dessen Behauptung es ist,dass die chromatische Zahl bei zusammenhängenden,ebenen Graphen vier ist.

Eine andere Möglichkeit ist es,nach Behandlung dieses Unterrichtsabschnitt zur Entwicklungsumgebung von Knotengraph (Konzeption EWK) überzugehen,wodurch eine Benutzeroberfläche zur graphischen (und nicht mehr nur textorientierten) Ausgabe (Anzeige) von Knoten und Kanten zur Verfügung gestellt wird,und die den Schülern jetzt bekannten Algorithmen Hamilton-und Eulerlinien mit geringen Änderungen objektorientiert in dieses System einzubinden,so dass der Ablauf dieser Algorithmen nun komfortabel und zeichnerisch anschaulich verfolgt werden kann.

Beide Möglichkeiten sollen in den folgenden Abschnitten besprochen werden.

III)Weitere mathematische Anwendungen (Tiefe Baumpfade,Kürzester Pfad und alle Pfade zwischen zwei Knoten,Färbung von Graphen)

(Konzeption CAK)

Pfade:

Beim offenen Eulerlinienproblem wird ein Kantenzug zwischen zwei Knoten gesucht,der **alle** Kanten genau einmal enthält.Läßt man die Forderung **alle** Kanten weg,und sucht einen (z.B. auch minimalen) Pfad oder alle Pfade zwischen zwei Knoten,erhält man das Labyrinthproblem.Der momentane Standort und der Ausgang sind nämlich die beiden Knoten des als Graph dargestellten Labyrinths,aus dem ein Weg zum Ausgang gesucht wird.

Eng verwandt mit diesem Problem ist die Aufgabe alle Pfade zu allen Knoten des Graphen von einem festen Ausgangsknoten aus zu suchen.Beschränkt man sich wiederum darauf,zu jedem Knoten des Graphen vom Ausgangsknoten nur einen Pfad zu suchen,wobei die Strategie,nach der das geschieht,darin bestet,den am weitesten entfernten Knoten jeweils zuerst zu besuchen,erhält man das TiefeBaumpfad-Problem.

Es ist am günstigsten mit dem letzten Problem im Unterricht zu beginnen.Auf diese Weise läßt sich nämlich das bisher im Informatikunterricht der Jahrgangsstufe 12 (Stichwort Datenstrukturen und Algorithmen) im Anschluss an die Listen behandelte Thema Struktur von Binär-Bäumen und Suchalgorithmen in Bäumen als Spezialfall des TieferBaum-Durchlauf-Algorithmus erfassen.

Zunächst werden dazu im Unterricht die im Kapitel C I „Knoten,Kanten,Bäume,Wege und Komponenten“ im Anschluss an die Definition des Graphen (besonders ab der Aufgabe C I 3) dargestellten Beispiele,Beispielsaufgaben,Suchstrategien und Definitionen in Bezug auf Bäume behandelt.

Die Beispiele geben Definitionen und Kriterien für Bäume und Gerüste an und zeigen,dass die geringe Anzahl von Vergleichen das Suchen in (geordneten) Bäumen (gegenüber Listen) besonders vorteilhaft machen.

Danach ergibt sich die Frage,wie ein Baum bzw. allgemein ein Graph besonders günstig durchsucht bzw. durchlaufen werden kann.Diese Frage wird im Kapitel C II „Suchverfahren in Graphen,Baum-und Pfadalgorithmen,Kreise, Gerüste" ausführlich dargestellt.

(Die folgenden Ausführungen überschneiden sich teilweise mit den Ausführungen in Kapitel C II und C III und werden hier nochmals aufgeführt,damit der Leser nicht zu blättern braucht.Sie eignen sich natürlich auch zur Verwendung für die Konzeption CAK und nicht nur für die Konzeption EWK bzw. DWK,von denen in Kapitel C II ausgegangen wird.)

Der von Schülern leicht zu findende (intuitive) Grobalgorithmus lautet:

a)Markiere den Startknoten.

b)Solange noch Kanten von markierten zu unmarkierten Knoten existieren,wähle eine solche Kante und markiere den unmarkierten Knoten.

Die Suchverfahren unterscheiden sich in der Weise,wie die Kanten ausgewählt werden.Da keine markierte Kante zweimal besucht wird,entsteht auf diese Weise bei einem zusammenhängenden Graph ein Suchbaum.

Die vom Lehrer dann vorzugebende Durchlaufstrategie für den TiefenBaum-Algorithmus lautet:

Bei der Tiefensuche wird versucht,die am weitesten vom Startknoten entfernten Knoten zuerst zu besuchen.Daher wird,sobald ein Knoten besucht und markiert wurde,direkt nach der (in der Kantenfolge) ersten von diesem Knoten wieder auslaufenden Kante gesucht,um einen noch weiter entfernt befindlichen Knoten als Zielknoten dieser Kante aufsuchen und markieren zu können.Erst wenn sich keine Kanten mehr finden lassen,die von einem besuchten Knoten ausgehen (Randknoten) oder alle möglichen Zielknoten dieses Knotens schon markiert wurden,werden die (in der Kantenfolge) nächsten Kanten (und Zielknoten) des als letztes markierten Knoten berücksichtigt.

Bei der Preorder-Reihenfolge werden dem oben genannten Markierungsalgoritmus gemäß jeweils in der zeitlichen Reihenfolge die Inhalte der markierten Knoten zuerst vor den Knoten(-Inhalten) des danach markierten Teilbaums (Teilgraphs) ausgegeben.Bei der Postorder Reihenfolge werden jeweils die Inhalte der Knoten jedes Teilbaums vor dem Knoten(-Inhalt) des Knotens,von dem der Teilbaum ausgeht,ausgegeben.

Es empfiehlt sich,diese Suchstrategie an Hand von geeigneten Graphen und auch speziell Bäumen von Schülern **manuell** durchführen zu lassen.Geeignet sind hierzu z.B. die Graphen der Aufgabe C II.1 im Kapitel Suchverfahren in Graphen,Baum-und Pfadalgorithmen,Kreise,Gerüste".

Der für die graphische Anzeige der tiefen Baumpfade eines Graphen geeignete Algorithmus des Kapitels C II wird bezüglich der Anzeigeart leicht modifiziert,so dass er jetzt für eine textorientierte Ausgabe geeignet ist.Außerdem wird auch die Methode procedure TPfadKnoten.ErzeugeAllePfadeundMinimalenPfad als Consolenanwendung entsprechend umgeschrieben.

Der Quellcode der Methoden wird im folgenden wiedergegeben.

```
TPfadCknoten=class(TInhaltsknoten)
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    procedure ErzeugeTiefeBaumPfade(Preorder:Boolean);
    procedure AllePfadeundMinimalerPfad(Kno:TInhaltsknoten;
    var Minlist:TKantenliste);
  end;

procedure TPfadCknoten.ErzeugeTiefeBaumPfade(Preorder:Boolean);
var Index:Integer;
    MomentaneKantenliste:TKantenliste;

  procedure GehezuNachbarknoten(Kno:TKnoten;Kna:TKante);
  label Endproc;
  var Ob:Tobject;
      Index:Integer;
  begin
    if  Kna.Zielknoten(Kno).Besucht=false then
    begin
```

```
      Kna.Pfadrichtung:=Kna.Zielknoten(Kno);
      MomentaneKantenliste.AmEndeanfuegen(Kna);
      if Preorder then TInhaltskante(MomentaneKantenliste).Listenausgabe;
      Kna.Zielknoten(Kno).Besucht:=true;
      if not Kna.Zielknoten(Kno).AusgehendeKantenliste.Leer then
        for Index:=0 to Kna.Zielknoten(Kno).AusgehendeKantenliste.Anzahl-1 do
          GehezuNachbarknoten(Kna.Zielknoten(Kno),Kna.Zielknoten(Kno).
          AusgehendeKantenliste.Kante(Index));
       if not preorder then TInhaltskante(MomentaneKantenliste).Listenausgabe;
      {ka.Zielknoten(kno).besucht:=false;Zusatzzeile für Erzeugung aller Pfade ****}
      MomentaneKantenliste.AmEndeloeschen(Ob);
    end;
    Endproc:
  end;

begin
  MomentaneKantenliste:=TKantenliste.Create;
  if Preorder then Writeln(Wert,' ');
  Graph.LoescheKnotenbesucht;
  Besucht:=true;
  if not AusgehendeKantenliste.Leer then
  for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
    GehezuNachbarknoten(self,self.AusgehendeKantenliste.Kante(Index));
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
  if not Preorder then Writeln(Wert,' ');
end;
```

Zur Erläuterung des Quellcodes siehe die entsprechenden,fast Quellcode-gleichen Methoden im Kapitel C II Suchverfahren im Kapitel „Graphen,Baum- und Pfadalgorithmen,Kreise,Gerüste".Die Ausgabe geschieht hier vereinfacht im Gegensatz zu der Methode in Kapitel C II durch die **textorientierte** Methode Listenausgabe.Die Speicherung der Kantenlisten in den Pfadlisten der Knoten ist in dieser einfacheren Textversion entbehrlich.

Die rekursive Procedure GehezuNachbarknoten realisiert den oben beschriebenen Grobalgorithmus.Je nach Durchlaufverfahren geschieht jetzt die Listenausgabe des Pfades vor (Preorder) bzw. nach (Postorder) dem Aufruf der nächsten Rekursionsebene (neuer Zielknoten).

Da die Schüler jetzt schon Übung im Umgang mit der rekursiven Programmierung durch die Backtracking-Algorithmen des letzten Abschnitts (Hamilton- und Eulerlinien) gewonnen haben,genügt es,die Grobstruktur des Quelltextes der Methode zu skizzieren und die Ausgestaltung als Schüleraufgabe zu stellen.

Der TiefeBaum-Algorithmus ist sehr flexibel in der Variation der Lösungsmöglichkeiten.

Fügt man nämlich z.B. die Zeile **** (ohne Kommentarklammern hinzu} erweitert sich der Algorithmus,so dass jetzt alle Pfade (vom momentanen Knoten aus) zu allen Knoten des Graphen durchlaufen werden.

Verändert man nun den Algorithmus alle Pfade so,dass nur solche Pfade unter allen Pfaden ausgegeben werden,die als Endknoten einen vorgegebenen zweiten Knoten Kno besitzen,so werden jetzt nur noch alle Pfade zwischen diesen beiden Knoten angezeigt.

Wenn noch zusätzlich unter diesen Pfaden stets der Pfad in einer Kantenliste Minlist gespeichert wird,der jeweils die kleinste Kantenwertsumme besitzt,ist der minimale Pfad zwischen diesen beiden Knoten nach Beenden der Methode in dieser Liste gespeichert.

Die folgende Methode läßt sich mit wenigen Zeilen aus der vorigen Methode ableiten.Der Quelltext zwischen den beiden Kommentarzeilen,die mit {****} bezeichnet sind,enthält die Änderungen.

```
procedure TPfadCknoten.AllePfadeundMinimalerPfad(Kno:TInhaltsknoten;
var Minlist:TKantenliste);
var Index:Integer;
    MomentaneKantenliste:TKantenliste;
```

```
 procedure GehezuNachbarknoten(K:TKnoten;Ka:TKante);
  var Ob:Tobject;
      Index,Zaehl:Integer;
  begin
    if  Ka.Zielknoten(K).Besucht=false then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(K);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
      if Kno=Ka.Pfadrichtung then                {****}
      begin
        TInhaltskante(MomentaneKantenliste).Listenausgabe;
        if ((Minlist.WertsummederElemente(Bewertung)>=
        MomentaneKantenliste.WertsummederElemente(Bewertung)))
          and (not MomentaneKantenliste.leer)
      then
      begin
        Minlist.Free;
        Minlist:=TKantenliste.create;
        for Zaehl:=0 to MomentaneKantenliste.Anzahl-1 do
          Minlist.AmEndeanfuegen(MomentaneKantenliste.Kante(Zaehl));
      end;
      end;                                         {****}
      Ka.Zielknoten(K).Besucht:=true;
      if not Ka.Zielknoten(K).AusgehendeKantenliste.Leer then
        for Index:=0 to Ka.Zielknoten(K).AusgehendeKantenliste.Anzahl-1 do
          GehezuNachbarknoten(Ka.Zielknoten(K),
          Ka.Zielknoten(K).AusgehendeKantenliste.Kante(Index));
      Ka.Zielknoten(K).Besucht:=false;
      MomentaneKantenliste.AmEndeloeschen(Ob);
    end;
  end;

begin
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.LoescheKnotenbesucht;
  Besucht:=true;
  if not AusgehendeKantenliste.Leer then
    for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
      GehezuNachbarknoten(self,AusgehendeKantenliste.Kante(Index));
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
end;
```

Es ist wieder empfehlenswert,die Ergebnisse der obigen Algorithmen bei Anwendung auf geeignete Beispielsgraphen durch die Schüler zunächst durch Ausführung der Algorithmen per Hand vorhersagen zu lassen,so dass man die vom Programm erzeugten Ergebnisse beurteilen und das Programm dadurch auf Richtigkeit überprüfen kann.

Werden als Beispiele Bäume und speziell geordnete Binärbäume gewählt,erkennt man,dass hier alle im Baum enthaltenden Pfade durchlaufen werden und man kann die Durchlaufordnungen verfolgen.Verfolgt man das Ziel einen geordneten Suchbaum bezüglich der Knotenwerte aufzubauen,zeigt sich dass neu hinzuzufügende Knoten nur an ganz bestimmten Stellen als Blätter in den Baum eingefügt werden können,um die Durchlaufordnungen zu erhalten.Entsprechend ist auch das Löschen von Knoten unter dem Gesichtspunkt der Durchlaufordnung problematisch.Auch lassen sich bezüglich einer vorgegebenen festen Menge von Knotenwerte verschiedene Suchbäume konstruieren,die alle dieselbe Ausgabeordnung haben,aber jeweils eine unterschiedliche Baumtiefe besitzen und daher unterschiedlich effektiv bezüglich des Erreichens eines bestimmten Knotenwerts gemäß einem Suchalgorithmus sind,der bei jedem Knoten einen Größenvergleich der Werte vornimmt und den jeweils richtigen Teilbaum wählt.

Man kann jetzt zusätzlich,falls gewünscht,die bekannten Algorithmen des Binären Suchens und eines Inorder-Durchlaufs in Bäumen implementieren,und damit die bisher im Informatikunterricht behandelten üblichen Verfahren zum Thema Bäume in den Unterrichtsgang integrieren.Die entsprechenden Methoden sind in Kapitel C II im Abschnitt Inorder-Durchlauf und Binäres Suchen beschrieben,und können mit wenigen Änderungen in eine textorientierte Version umgeschrieben werden.

Überhaupt können alle in Kapitel C beschrieben Methoden auch in eine textorientierte Version umgeschrieben werden,wobei lediglich stets die Ein-und Ausgabemethoden verändert werden müssen und die Algorithmen ansonsten er-

halten bleiben.Ein weiteres Beispiel dafür wird jetzt noch abschließend am Beispiel Färbbarkeit eines Graphen dargestellt.

(Eine andere Möglichkeit ist es,einen Binärbaum als allgemeiner Datentyp ADT als neues Projekt objektorientiert als Fortsetzung des Projekts Liste als Objekt aufzubauen,wobei die Inhalte (Werte) der Knoten und ihr Datentyp erst nachträglich vom Benutzer festgelegt werden können.)

Färbbarkeit:

Das Kapitel C IV beschreibt eine problemorientierte Unterrichtsreihe,die die Themen Plättbarkeit eines Graphen und Färbbarkeit zum Inhalt hat,und zunächst von der Eulerbeziehung zwischen der Zahl der Knoten und Kanten in einem ebenen Graph ausgeht.Wegen der gegenseitigen Beziehungen der verschiedenen Themen Plättbarkeit,Eulerbeziehung und Färbbarkeit eignet sich diese Unterrichtsreihe besonders für einen wissenschaftsorientierten oder genetischen Unterricht (gemäß Wagenschein und Wittenberg).

Der Hauptalgorithmus dieses Kapitels dient der Ermittlung der Zahl der Farben,mit der die Knoten eines Graphen gefärbt werden können,so dass durch Kanten benachbarte Knoten verschiedene Farben besitzen.Die kleinste dieser Farbzahlen ist die chromatische Zahl eines Graphen,die sich mit dem Algorithmus durch Experimentieren finden läßt.Nach dem berühmten Vierfarbensatz ist die chromatische Zahl für ebene (zusammenhängende) Graphen stets maximal vier.

Im folgenden wird die textorientierte Version des graphikorientierten Algorithmus aus Kapitel C IV erläutert,die sich von dieser nur wieder,wie schon bei den weiter oben beschriebenen Algorithmen im wesentlichen durch die Anzeige-und Ausgabeart unterscheidet.Das Problem ist dazu geeignet,noch einmal das Prinzip des Backtracking-Algorithmus im Unterricht an einem geeigneten Beispiel einzuüben.Außerdem tritt das Prinzip der objektorientierten Vererbung dadurch noch deutlicher als bei den vorigen Graphenstrukturen hervor,da ein erweiterter Knotentyp mit neuen Datenfelder benötigt wird.

Dazu wird wieder ein von TInhaltsknoten abgeleiterter Knoten TFarbknoten erzeugt,der die neuen Datenfelder Farbzahl_ und Ergebnis_ enthält,auf die nach außen durch die Propertys Farbzahl und Ergebnis mittels der im private-Teil vereinbarten Methoden zugegriffen werden kann.Als Farbzahl_ wird die Farbe des Knoten in Form einer Integerzahl gespeichert,und Ergebnis_ dient zur Aufnahme eines auszugebenden Ergebnis,das aus der Knotenbezeichnung (Knotenwert) und der Farbzahl besteht.

```
TFarbCknoten = class(TInhaltsknoten)
  private
    Farbzahl_:Integer;
    Ergebnis_:string;
    function WelcheFarbzahl:Integer;
    procedure SetzeFarbzahl(Fa:Integer);
    function WelchesErgebnis:string;
    procedure SetzeErgebnis(S:string);
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    property KnotenFarbe:Integer read WelcheFarbzahl Write SetzeFarbzahl;
    property Ergebnis:string read WelchesErgebnis Write SetzeErgebnis;
  end;
```

Eine kommentierte Beschreibung der Methoden Create,Free,Freeall sowie der in der Objektdeklaration enthaltenden Property-Methoden befindet sich im Anhang.

Der von TInhaltsgraph abgeleitet Graph TFarbgraph enthält die zum Erzeugen der zulässigen Farbverteilung der Knoten des Graphen notwendigen Methoden.Die Hauptmethode ist die Methode Farbverteilung,die nach einem Backtracking-Algorithmus mit der Anzahl der Knoten als Stufenzahl arbeitet.Diese Methode wird von FaerbeGraph aufgerufen und benutzt selber die

Methode Knotenistzufaerben,die ermittelt,ob ein Knoten mit einer Farbe unter Berücksichtigung seiner Nachbarknoten noch gefärbt werden kann.

```
TFarbCgraph = class(TInhaltsgraph)
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    procedure SetzebeiallenKnotenAnfangsfarbe;
    function Knotenistzufaerben(Index:Integer;AnzahlFarben:Integer):Boolean;
    procedure Farbverteilung(Index:Integer;Anzahlfarben:Integer;
    var Gefunden:Boolean;EineLoesung:Boolean;var Ausgabeliste:TStringlist);
    procedure ErzeugeErgebnis;
    procedure FaerbeGraph(var Sliste:TStringlist);
    procedure Menu;
  end;
```

Eine kommentierte Beschreibung der Methoden Create, Free, Freeall, ErzeugeErgebnis sowie der Methode SetzbeiallenKnotenAnfangsfarbe befindet sich wieder im Anhang.

Die Methode Knotenistzufaerben ermittelt,ob ein Knoten mit der Nummer Index in der Knotenliste in einer möglichen Farbe bezüglich seiner Nachbarknoten zu färben ist.

Dazu wird die Knotenfarbe des Knoten um eins modulo der gewählten maximalen Anzahl der Farben plus 1 erhöht und überprüft,ob die Zielknoten der ein-und ausgehenden Kanten jeweils dieselbe Farbzahl haben.Die Methode ist Quellcodegleich der in der Konzeptionen EWK und DWK eingesetzten Methode und wird im Kapitel C IV genauer beschrieben.

Im folgenden wird eine Übersicht über die Hauptmethoden Farbverteilung und Faerbegraph sowie Menu gegeben:

Die Methode Farbverteilung bestimmt nach einem Backtrackingverfahren mit den Knotennummer in der Knotenliste als Stufenzahl eine oder alle Farbverteilungen des Graphen:

```
procedure TFarbCgraph.Farbverteilung(Index:Integer;AnzahlFarben:Integer;
var Gefunden:Boolean;EineLoesung:Boolean;var Ausgabeliste:TStringlist);
label Endproc;
var Knotenzufaerben:Boolean;
    Zaehl:Integer;
    Kno:TFarbCknoten;
    S:string;

begin
  if Gefunden and EineLoesung then goto Endproc;
  repeat
    Knotenzufaerben:=Knotenistzufaerben(Index,AnzahlFarben);
    if (Knotenzufaerben) and (Index<AnzahlKnoten-1)
    then
      Farbverteilung(Index+1,AnzahlFarben,Gefunden,EineLoesung,Ausgabeliste)
    else
      if (Index=Anzahlknoten-1) and Knotenzufaerben then
      begin
        Gefunden:=true;
        ErzeugeErgebnis;
        S:='';
        for Zaehl:=0 to Knotenliste.Anzahl-1 do            ****
        begin
          Kno:=TFarbCknoten(Knotenliste.Knoten(Zaehl));
          S:=S+' ' +Kno.Ergebnis;
        end;
        Ausgabeliste.Add(S);
        if Gefunden and EineLoesung then goto Endproc;
      end
  until (not Knotenzufaerben) or (Gefunden and EineLoesung);
  Endproc:
end;
```

Auch die Wirkungsweise dieser Methode ist im Kapitel C IV ausführlich erläutert.Der Quelltext der oben beschrieben Methode ist lediglich um Anweisungen zur Färbung und zeichnerischen Darstellung des Graphen gekürzt und

enthält deshalb auch nicht die Function Farbe,die eine Knotenzeichenfarbe auswählt.

Der Abschnitt **** ist deshalb in Kapitel C IV erweitert zu:

```
for Zaehl:=0 to Knotenliste.Anzahl-1 do
begin
  Kno:=TFarbknoten(Knotenliste.Knoten(zaehl));
  Kno.Farbe:=Farbe(Kno.Knotenfarbe);
  S:=S+' ' +Kno.Ergebnis;
end;
Knotenwertposition:=2;
ZeichneGraph(Flaeche);
Demopause;
Knotenwertposition:=0;
```

Die Methode FaerbeGraph ruft die Methode Farbverteilung mit den richtigen Parametern auf:

```
procedure TFarbCgraph.FaerbeGraph(var Sliste:TStringlist);
var AnzahlFarben:Integer;
    StringFarben,Antwort:string;
    Gefunden,EineLoesung:Boolean;
    Index:Integer;
begin
  if Leer then exit;
  SetzebeiallenKnotenAnfangsfarbe;
  repeat
    Write('Eingabe Farbzahl: ');
    Readln(StringFarben);
    AnzahlFarben:=StringtoInteger(StringFarben);
    if (AnzahlFarben<0) or (AnzahlFarben>19) then Writeln('Fehler: 0<Anzahl Farben <20 !');
  until (AnzahlFarben>0) and (AnzahlFarben<20);
  Write('Nur eine Lösung? (j/n): ');
  Readln(Antwort);
  if (Antwort='j')or (Antwort='J')
  then
    EineLoesung:=true
  else
    EineLoesung:=false;
  Gefunden:=false;
  Farbverteilung(0,AnzahlFarben,Gefunden,EineLoesung,Sliste);
  ErzeugeErgebnis;
end;
```

Dazu wird die Anzahl der Farben von der Tastatur mittels Readln eingelesen.Außerdem wird abgefragt,ob nur eine Lösung oder alle Lösungen ausgegeben werden sollen.In der Stringliste SListe werden die Ergebnisse gespeichert.

Diese Methode Menu erzeugt schließlich eine neues (zunächst) leeres Anzeigefenster,ruft FaerbeGraph auf und sorgt für die Ausgabe (mittels Writeln) der in Sliste gespeicherten Ergebnisse:

```
procedure TFarbCGraph.Menu;
var Sliste:TStringlist;
    Index:Integer;
    Farbgraph:TFarbCgraph;
    KnoC:TFarbCknoten;
    Kno:TInhaltsknoten;
    Ka,K:TInhaltskante;
    S:string;
begin
   Clrscr;
   Writeln('Graph färben');
   if Leer
  then
  begin
     Writeln('leerer Graph');
     Readln;
     exit;
  end;
  Farbgraph:=TFarbcGraph.Create;                                    ****
  for Index:=0 to Knotenliste.Anzahl-1 do
  begin
    Kno:=TInhaltsknoten(Knotenliste.Knoten(Index));
    Knoc:=TFarbCknoten.create;
    Knoc.Wert:=Kno.Wert;
    Farbgraph.Knotenliste.AmEndeanfuegen(Knoc);
```

```
  end;
  if not Kantenliste.leer then
  for Index:=0 to Kantenliste.Anzahl-1 do
  begin
    Ka:=TInhaltskante(Kantenliste.Kante(Index));
    K:=TInhaltskante.Create;
    K.Wert:=Ka.Wert;
    Farbgraph.FuegeKanteein(Graphknoten(TFarbCknoten(Ka.AnfangsKnoten)),
    Graphknoten(TFarbCknoten(Ka.Endknoten)),Ka.gerichtet,K);
  end;                                                            ****
  Sliste:=TStringlist.Create;
  Farbgraph.FaerbeGraph(Sliste);
  Writeln;
  if Sliste.count>0 then
  begin
    Writeln('Die Farbverteilung ist:');
    Writeln;
    for Index:=0 to Sliste.count-1 do
    begin
      S:=Sliste.Strings[Index];
      Writeln(S);
    end
  end
  else
    Writeln('Keine Lösung!');
  Writeln;
  Readln;
  Sliste.Free;
  SListe:=nil;
  Farbgraph.Freeall;
  Frabgraph:=nil;
end;
```

Es wird zunächst ein strukturgleicher Graph Farbgraph durch Einfügen von Knoten und Kanten in die entsprechenden Knoten- und Kantenlisten mittels der Anweisungen **** erzeugt,so dass der ursprüngliche Graph (für andere Anwendungen) erhalten bleibt.(Die Methode InhaltskopiedesGraphen fehlt in der textorientierten Version.)Auf den Farbgraph wird dann der Algorithmus FaerbeGraph angewendet.

Ein Unterrichtsgang sollte auf die Beispiele und Anwendungsaufgaben des Kapitels C IV„Eulerbeziehung,ebene Graphen und Färbbarkeit“ zurückgreifen,die dann natürlich soweit es Anwendungen des Färbungalgorithmus betrifft,nur textorientiert interpretiert werden können.Dort findet sich, wie oben mehrfach gesagt,eine nähere Erläuterung des Backtrackingalgorithmus sowie des Quellcodes (bis auf die Methode Menu) der oben vorgegebenen Methoden (allerdings der geringfügig verschiedenen graphisch orientierten Version).

Interessanter wird eine Problemlösung natürlich dadurch,dass die Knoten und Kanten als graphische Objekte auf einer Zeichenoberfläche dargestellt werden und die Farbverteilung sowie der Ablauf des Algorithmus anschaulich zeichnerisch dargestellt werden kann.

Dies ist möglich durch den Übergang von der textorientierten zur graphisch orientierten Objektumgebung,wie sie das Entwicklungssystem Knotengraph mittels der zu vererbenden Benutzeroberfläche bereitstellt.

Die didaktischen und methodischen Möglichkeiten dieses Systems sollen deshalb im nächsten Abschnitt beschrieben werden.Wie schon gesagt,kann auch gleich von den textorientierten Algorithmen der Euler-und Hamiltonlinien auf dieses System umgestiegen werden,um beispielsweise diese beiden bisher textorientiert realisierten Algorithmen dann zusätzlich graphikorientiert zu implementieren.

Andererseits könnte eine Unterrichtsreihe zum Thema Graphen und Algorithmik,die das Verstehen des objektorientierten Aufbau der Graphenstruktur in den Vordergrund stellt und allgemein eine Einführung in die objektorientierte Programmierung geben will,nach einem Unterrichtsgang gemäß den Abschnitten 2 bis 4 dieses Kapitels auch ihren Abschluß (unter Verzicht der graphischen Darstellung des Graphen) mit der Besprechung von ein oder zwei in diesem und dem vorigen Abschnitt besprochenen Anwendungsalgorithmen erhalten.

6)Der Unterrichtseinsatz der vererbbaren objektorientierten Entwicklungsumgebung des Programmsystems Knotengraph als didaktisches und methodisches Werkzeug

(Konzeption EWK)

Mit Hilfe der Entwicklungsumgebung des Programms Knotengraphs lassen sich die bisher nur durch textorientierte Ein-und Ausgabe realisierten Algorithmen komfortabler unter Verwendung der unter dem Betriebssystem Windows üblichen graphischen Ein- und Ausgabemöglichkeiten (Fenster und Komponenten) mittels zeichnerischer Darstellung des Graphen gestalten.Dadurch läßt sich methodisch eine größere Anschaulichkeit des Ablaufs der benutzten Verfahren und damit eine größere Motivation zur Beschäftigung mit dem Thema Algorithmische Graphentheorie und der Erstellung von entsprechendem Quellcode erreichen.Die jetzt vorhandene größere Komplexheit kann durch die Bereitstellung der benötigten graphischen Elemente mittels objektorientierter Vererbung ausgeglichen werden.

Die Entwicklungsoberfläche von Knotengraph stellt dazu eine Form,nämlich Knotenform bereit,die schon alle funktionsfähigen Menüs zur graphischen Verwaltung einer Objekt-Graphenstruktur (Einfügen, Löschen, Editieren, Verschieben von Knoten und Kanten usw.) enthält.Die Hauptmenüs der Knotenform mit den Bezeichner Datei,Bild,Knoten,Kanten,Eigenschaften und Ausgabe sind dabei noch weiter unterteilt in entsprechende Untermenüs,um die unterschiedlichen Aufgaben ausführen zu können (zur Beschreibung siehe das Kapitel Bedienungsanleitung im Anhang).

Alle Untermenüs beinhalten als Ereignismethoden schon komplett den entsprechenden Quellcode für die ihnen zugewiesene Wirkungsweise,so dass nach dem Compilieren und Starten der Entwicklungsumgebung (ohne dass es nötig ist, eine Zeile Quellcode zu schreiben) sofort schon die komplette graphischorientierte Verwaltung eines Graphen vorhanden ist.

Diese Entwicklungsumgebung kann jetzt um entsprechende Anwendungsalgorithmen der Graphentheorie objektorientiert erweitert werden.

Durch die von Delphi bereitgestellte visuelle Vererbung von Formen ist es möglich,die komplette Benutzeroberfläche den Schülern wiederum als Objekt zur Verfügung zu stellen,so dass das Hinzufügen von Anwendungsmethoden in einer neuen durch Vererbung erzeugten Form erfolgt,deren zugehörige Unit noch keinen Quellcode enthält (außer dem vorgegebenen Objekt Knotenformular vom Typ TKnotenformular s.u.).

Um durch Vererbung eine neue Form (fast) ohne Quellcode mit Hilfe der Entwicklungsumgebung von Delphi (ab Version 2.0) zu erhalten,die alle Eigenschaften der Form Knotenform enthält,wählt man nach Laden der Knotengraphoberfläche mittels des Menüs Neu den Eintrag KProjekt in dem darauf folgenden Fenster aus.Durch Auswahl von Knotenform und Markieren des Eintrags Vererben entsteht nach der Bestätigung durch OK eine neue Leerform und eine dazugehörige Unit1,die außer dem üblichen von Delphi normalerweise vorgegebene Quellcode-Leergerüst einer leeren Unit noch die folgenden Zeilen enthält:

```
type
  TKnotenform1 = class(TKnotenform)
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

var
  Knotenform1: Tknotenform1
```

Dadurch wird ein neuer Objekttyp TKnotenform1 und eine neue zugehörige Instanz Knotenform1 deklariert,die alle Eigenschaften der vorigen Form Knotenform enthält,und insbesondere die oben erwähnte Verwaltung eines Graphen durch die genannten Menüs und Untermenüs bzw. deren Quellcode bereitstellt Man sollte danach nicht vergessen,die neue Form Knotenform1 mittels des

entsprechenden Menüpunkts im Fenster Projektoptionen zur Hauptform (statt der vorigen Form Knotenform) zu machen.

Natürlich lassen sich die Bezeichner Unit1, TKnotenform1 und Knotenform1 jetzt durch neue aussagekräftigere Bezeichner ersetzen: z.B Unit UForm, TKnotenformular und Knotenformular:

```
type
  TKnotenformular = class(TKnotenform)
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

var
  Knotenformular: TKnotenformuler
```

(Bei der Installation von CD liegt diese Version schon fertig im Verzeichnis EWK vor.)

Das Objekt Knotenformular enthält nun wiederum durch Vererbung von Knotenform eine Property Graph vom Typ TInhaltsgraph,die den Graphen der Entwicklungsumgebung darstellt,wodurch unter Einbindung der entsprechenden Units eine objektorientierte Klassenbibliothek zur Verwaltung des Graphen sowie zur Programmierung von Graphenalgorithmen zur Verfügung gestellt wird.Dadurch wird die Durchführung von Unterrichtsprojekten zum Thema Algorithmische Graphentheorie erst möglich gemacht,da man nun wiederum ausgehend von Graph neue eigene Graphen mit gewünschten Eigenschaften objektorientiert aufbauen kann,die auf die Klassenbibliothek zugreifen können und sich auf der Form Knotenformular darstellen.

Übersicht über den kompletten bereitgestellten Quellcode der Unit UForm:

```
unit UForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, Buttons, StdCtrls, ExtCtrls,UList,UGraph,UInhgrph,UAusgabe,
  UKante, UKnoten;

type
  TKnotenformular = class(TKnotenform)
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

var
  Knotenformular: TKnotenformular;

implementation

{$R *.DFM}


end.
```

In diese Leerform können jetzt z.B. neue Menüs (aber auch alle anderen von Delphi bereitgestellten Komponenten) eingefügt werden,die neuen Quellcode z.B. für den Ablauf von mathematischen Graphenalgorithmen enthalten können.

Dazu werden die in der Entwicklungsumgebung von Knotengraph zur Verfügung stehenden Units UList,UGraph,UInhgrph,UKante,UAusgabe mittels uses-Anweisung eingebunden.Die ersten beiden genannten Units sind dabei erweiterte Versionen der schon besprochenen Units UListC und UGraphC. Die Unit UInhgrph erweitert die Unit UGraph um Methoden zur graphischen Darstellung eines Graphen auf der Zeichenoberfläche der Form Knotenform oder Knotenformular und hat dabei eine analoge Wirkungsweise wie sie die Unit UInhgrphC für die textorientierte Ausgabe hatte.Die Units UKante und UAusgabe dienen

der Unterstützung der graphisch orientierten Eingabe von Kantenwerten bzw. der Ausgabe von Algorithmenergebnissen.

So kann sich jetzt der Unterricht hauptsächlich auf das Entwerfen von mathematischen Graphenalgorithmen und das Erstellen des zugehörigen Quellcodes konzentrieren.Die geeigneten Methoden zur Verwaltung des Graphen stehen entweder schon in Form der fertigen Menüs und Untermenüs oder in Form von geeigneten Methoden der Objektstrukturen der erwähnten Units zur Verfügung.

Eine Beschreibung der Datenstruktur des Objektgraphen,der in den genannten Units enthalten ist,befindet sich im Kapitel Datenstruktur F I im Anhang. Sie stellt im wesentlichen,aufbauend auf den Objekten TKnoten,TKante und TGraph (vgl. Unit UGraphC) eine graphisch-ausgerichtete Änderung und Erweiterung der Graphenstruktur der Objektklassen der Unit UInhgrphC bestehend aus TInhaltsknoten, Tinhaltskante und TInhaltsgraph dar.Deshalb ist der Umstieg auf das graphisch orientierte Objektsystem der Entwicklungsumgebung EWK für Schüler nicht besonders schwer,weil die Datenstruktur im Prinzip bekannt ist,und nur noch ergänzt werden muß.

Andererseits ist es möglich,ohne Graphenalgorithmen mit textorientierter Ausgabe nach der Konzeption CAK erstellt zu haben,gleich mit der Entwicklungsumgebung von Knotengraph (EWK) zu beginnen,um mathematische Algorithmen objektorientiert in geeigneten Quellcode unter zeichnerischer Darstellung des Graphen zu transformieren.Dazu ist jedoch zunächst eine eingehende Besprechung der Datenstruktur des Graphen und eine Erörterung der Bedeutung bzw. der Wirkungs-und Einsatzweise der zum Erzeugen der mathematischen Algorithmen benötigten Objektmethoden der bereitgestellten Klassenbibliothek wesentlich.Eine Übersicht über die Methoden des Entwicklungsumgebung Knotengraph (EWK) sowie der Methoden des (fertigen) Programms Knotengraphs (DWK) mit Beschreibung,die soweit sie die Algorithmen der Menüs Pfade und Anwendungen betreffen,als Musterbeispiele für eigenen zu erstellenden (Schüler-)Quellcode dienen können,findet sich im Anhang.Außerdem sollten bei den Schülern unbedingt Grundkenntnisse der objektorientierten Programmierung mit Delphi (z.B. mittels Durchführung des Einstiegsprojektes) und Kenntnisse einer Objektlistenstruktur (z.B. mittels Durchführung des Projekts Liste als Objekt) vorhanden sein.

Im folgenden wird eine Kurzübersicht über die Wirkungsweise der am meisten benötigten Methoden der Units UList,UGraph, UInhgrph und UKnoten gegeben,wie sie den Schülern (zusätzlich zur Übersicht über die Objekt-Datenstruktur des Graphen) bei einem Sofort-Einstieg in die Entwicklungsumgebung von Knotengraph (ohne zuvor die textorientierte Version behandelt zu haben) zur Verfügung gestellt werden sollte:

TListe

Methoden:

procedure AmEndeanfuegen(Ob:TObject)
Objekt am Ende der Liste anfügen

procedure AmAnfanganfuegen(Ob:TObject)
Objekt am Anfang der Liste anfügen

procedure AmAnfangloeschen(var Ob:TObject)
Objekt am Anfang der Liste löschen

procedure AmEndeloeschen(var Ob:TObject)
Objekt am Ende der Liste löschen

procedure LoescheElement(Ob:TObject)
Objekt aus der Liste löschen

function Anzahl:Integer
Anzahl der Elemente der Liste

function WertSummederElemente(Wert:TWert):Extended
Summe der Elemente bezüglich Wert

function WertproduktderElemente(Wert:TWert):Extended
Produkt der Elemente bezüglich Wert

function Leer:Boolean
Gibt an,ob die Liste leer ist.

Functionen zur Datenkonvertierung (selbsterklärend,die beiden letzten Methoden runden jeweils auf die angegebene Stellenzahl Stelle):

function StringtoReal(S:string):Extended

function RealtoString(R:Extended):string

function Integertostring(I:Integer):string

function StringtoInteger(S:string):Integer

function RundeStringtoString(S:string;Stelle:Integer):string

function RundeZahltoString(R:Real;Stelle:Integer):string

TKantenliste:

Methoden:

function Kante(Index:Integer):TKante
Kante der Liste mit Nummer Index (0<= Index<=Anzahl-1)

function Kopie:TKantenliste
Kopie der Kantenliste mit Orginalkanten

function Graph:TGraph
Erzeugt den der Kantenliste äquivalenten Graph

TKante:

Propertys:

Anfangsknoten:TKnoten
Anfangsknoten der Kante

Endknoten:TKnoten
Endknoten der Kante

Pfadrichtung:TKnoten
Durchlaufrichtung der Kante durch Angabe des Zielknoten

Gerichtet:Boolean
Gerichtet-Eigenschaft

Position:Integer
Bestimmt bei mehreren Wertfeldern den Zugriff der Function Wert(s.u.).

Besucht:Boolean
Besucht-Markierung

Wert:string
Wert des Inhaltsfeldes,das durch Position bestimmt wird.

Methoden:

function Zielknoten(Kno:TKnoten):TKnoten
Der zu Kno entgegengesetzte Knoten bzw. der Endknoten

function KanteistSchlinge:Boolean
Anfangs-und Endknoten sind identisch.

TKnotenliste:

Methoden:

function Knoten(Index:Integer):TKnoten
Knoten der Liste mit Nummer Index (0<= Index<=Anzahl-1)

TKnoten:

Propertys:

Graph:TGraph
Falls der Knoten in einen Graph eingefügt wurde,Zeiger auf den Graph.

EingehendeKantenliste:TKantenliste
Liste der eingehenden Kanten des Knoten

AusgehendeKantenliste:TKantenliste
Liste der ausgehenden Kanten des Knoten

Pfadliste:TPfadliste
Liste,in der Pfade zum bzw. vom Knoten aus gespeichert werden können.

Position:Integer
Position des Datenfeldes,auf das Wert (s.u:) zurückgreift.

Besucht:Boolean
Besucht-Markierung des Knotens

Wert:string
Wert des Inhaltsfeldes,das durch Position bestimmt wird.

Methoden:

function Kantenzahlausgehend:Integer
Zahl der ausgehenden Kanten

function Kantenzahleingehend:Integer
Zahl der eingehenden Kanten

function Kantenzahlungerichtet:Integer
Zahl der ungerichteten Kanten (ein-bzw. ausgehend)

function Kantenzahl:Integer
Gesamtkantenzahl aller mit dem Knoten inzidenten Kanten

function AnzahlSchlingen:Integer
Anzahl der Schlingenkanten

function MinimalerPfad(Wert:TWert):TPfad
Bestimmt den minimalen Pfad der Pfadliste des Knoten bezüglich Wert.

function InhaltallerKnoten(Sk:TString):string;
Erzeugt eine Rückgabe in Form eines strings,der alle Knotenwerte gemäß Sk vom Typ TString des Graphen enthält.

TGraph:

Propertys:

Knotenliste:TKnotenliste
Liste,die alle Knoten des Graphen enthält.

Kantenliste:TKantenliste
Liste,die alle Kanten des Graphen enthält

Methoden:

function Leer:Boolean
Gibt zurück,ob der Graph ohne Knoten ist.

function AnzahlKanten: Integer
Anzahl aller Kanten des Graphen ohne Schlingenkanten

function AnzahlSchlingen:Integer
Anzahl aller Schlingenkanten des Graphen

function AnzahlKantenmitSchlingen:Integer
Anzahl aller Kanten des Graphen mit Schlingenkanten

function AnzahlKnoten:Integer
Anzahl der Knoten des Graphs

function AnzahlgerichteteKanten:Integer
Anzahl der gerichteten Kanten

function AnzahlungerichteteKanten:Integer
Anzahl der ungerichteten Kanten

fuction AnzahlKomponenten:Integer
Anzahl der Komponenten des Graphen

function Kantensumme(Wert:TWert):Extended
Summe aller Kantenwerte bezüglich Wert

function Kantenprodukt(Wert:TWert):Extended
Produkt aller Kantenwerte bezüglich Wert

function Kopie:TGraph
Kopie der Graphenstruktur mit Orginalknoten und Kanten

procedure Knotenloeschen(Kno:TKnoten)
Löscht den Knoten Kno aus dem Graph.

procedure LoescheKantenbesucht
Löscht die Besucht-Markierung aller Kanten des Graphen.

procedure LoescheKnotenbesucht
Löscht die Besucht-Markierung aller Knoten des Graphen.

TPfadliste:

Methoden:

function Pfad(Index:Integer):TPfad
Pfad der Liste mit der Nummer Index (0<=Index<=Anzahl-1)

function Kopie:TPfadliste
Erzeugt eine Kopie der Pfadliste mit den Orginalpfaden.

TPfad:

function PfadSumme (Wert:TWert): Extended
Gibt die Summe des Pfades bezüglich Wert zurück.

function Pfadprodukt (Wert:TWert): Extended
Gibt das Produkt des Pfades bezüglich Wert zurück.

TInhaltsknoten:

Propertys:

X:Integer
X-Koordinate des Knotenmittelpunkts

Y:Integer
Y-Koordinate des Knotenmittelpunkts

Radius:Integer
Radius des Knotenkreises

Farbe:TColor
Farbe des Knotens

Stil:TPenstyle
Zeichenstil des Knotens

Methoden:

procedure ZeichneKnoten(Flaeche:TCanvas)
Zeichnet einen Knoten auf der Fläche Flaeche.

procedure Knotenzeichnen(Flaeche:TCanvas;Demo:Boolean;Pausenzeit:Integer)
Zeichnet einen Knoten auf der Fläche Flaeche rot und gestrichelt,legt bei Demo=true eine Pause der Länge Pausenzeit ein,und zeichnet den Knoten danach wieder schwarz und durchgezogen.

procedure AnzeigePfadliste(Flaeche:TCanvas;Ausgabe:TLabel
var SListe:TStringList;Zeichnen:Boolean;LetzterPfad:Boolean)
Zeigt die Pfade der Pfadliste bei Zeichnen=true nacheinander gefärbt und gestrichelt auf der Fläche Flaeche an (der letzte Pfad bleibt bei LetzterPfad=true gezeichnet),gibt die Knotenfolge der Pfade im Label Ausgabe aus,und fügt sie der Liste SListe hinzu.

TInhaltskante:

Propertys:

Typ:char
Typ des Inhalts (Werts) der Kante (Integer,string,Real)

Weite:Integer
Auslenkung des Kantenmittelpunkts von der Verbindungsstrecke zwischen den Randknoten

Farbe:TColor
Farbe der Kante

Stil:TPenstyle
Zeichenstil der Kante

Methoden:

procedure ZeichneKante(Flaeche:TCanvas)
Zeichnet die Kante auf der Fläche Flaeche.

procedure Kantezeichnen(Flaeche:TCanvas;Demo:Boolean;Pausenzeit:Integer)
Zeichnet die Kante auf der Fläche Flaeche zunächst rot und gestrichelt,legt dann eine Pause (bei Demo=true) der Länge Pausenzeit ein,und zeichnet die Kante danach schwarz und durchgezogen.

TInhaltsgraph:

Propertys:

Knotenwertposition:Integer
Position des Datenfeldes zur Ausgabe mittels Wert der Knoten des Graphen

Kantenwertposition:Integer
Position des Datenfeldes zur Ausgabe mittels Wert der Kanten des Graphen

Pausenzeit:Integer
Einzulegende Pausenzeit

Demo:Boolean
Demo-Modus setzen

Zustand:Boolean
Dient als Flag zum Auslösen von Aktionen bei ereignisorientierter Programmierung.

Knotengenauigkeit:Integer
Bestimmt die Genauigkeit,mit der Zahlenergebnisse bei Knoten angezeigt werden.

Kantengenauigkeit:Integer
Bestimmt die Genauigkeit,mit der Zahlenergebnisse bei Kanten angezeigt werden.

Radius:Integer
Gemeinsamer Radius aller Knoten des Graphen

LetzterMausklickknoten:TInhaltsknoten
Speichert den letzten mit der Maus angeklickten Knoten des Graphen

Methoden:

procedure Demopause
Fügt eine Pause,deren Wert durch Pausenzeit festgelegt wird,in den Programmablauf ein.

procedure FuegeKnotenein(Kno:TInhaltsknoten)
Fügt den Knoten Kno in den Graph ein.

procedure FuegeKanteein(Kno1,Kno2:TInhaltsknoten;Gerichtet:Boolean;Ka:TInhaltskante)
Fügt die Kante Ka zwischen die Knoten Kno1 und Kno2 des Graphen ein,wobei die Eigenschaft gerichtet durch Gerichtet vorgegeben wird.

procedure EinfuegenKante(Ka:TInhaltskante)
Fügt die Kante Ka in den Graph ein.

procedure LoescheInhaltskante(Ka:TInhaltskante)
Löscht die Kante Ka aus dem Graph

procedure ZeichneGraph(Flaeche:TCanvas)
Zeichnet den Graphen auf der Fläche Flaeche.

procedure Graphzeichnen(Flaeche:TCanvas;Ausgabe:TLabel;Wert:TWert;Sliste:TStringlist Demo:Boolean;Pausenzeit:Integer;Kantengenauigkeit:Integer)
Zeichnet den Graph auf der Fläche Flaeche zunächst rot gefärbt und nach Ablauf der Pausenzeit schwarz gefärbt.Die Knoten der Kantenliste sowie ihre Wert-Summe und ihr Wert-Produkt (gemäß Wert) werden im Label Ausgabe angezeigt und der Liste SListe hinzugefügt.

procedure FaerbeGraph(F:TColor;T:TPenstyle)
Der Graph wird gemäß den Parametern gefärbt und markiert.

function FindezuKoordinatendenKnoten(var A,B:Integer;var Kno:TInhaltsknoten):Boolean
Bestimmt einen Graphknoten Kno,der sich in unmittelbarer Nähe des Punktes mit den X/Y-Koordinaten A und B befindet.

function Graphknoten(Kno:TInhaltsknoten):TInhaltsknoten
Bestimmt zu einem Knoten Kno mit X/Y-Koordinaten einen in unmittelbarer Nähe liegenden Knoten des Graphen.

function InhaltsKopiedesGraphen(Inhaltsgraphclass:TInhaltsgraphclass; Inhaltsknotenclass:TInhaltsknotenclass;Inhaltskanteclass:TInhaltskanteclass; UngerichteterGraph:Boolean):TInhaltsgraph
Erzeugt eine Kopie des Graphen mit neuen Knoten und Kanten,von einem Objekttyp,der durch die Parameter Inhaltsknotenclass und Inhaltskanteclass vorgegeben wird.Der neue Graph ist vom Typ Inhaltsgraphclass.Der Parameter UngerichteterGraph bestimmt,ob ein ungerichteter Graph erzeugt wird.

TKnotenform

Propertys

Aktiv
Bestimmt,ob der ursprüngliche Graph Graph oder der Ergebnisgraph GraphH gezeichnet wird.

Graph:TInhaltsgraph
Ursprungsgraph,der vom Anwender durch die Menüs der Entwicklungsumgebung erzeugt wird.

GraphH:TInhaltsgraph
Steht zur Anzeige als Ergebnisgraph für Algorithmen zur Verfügung.

Über diese Übersicht hinaus werden bei den einzelnen Algorithmen des Kapitel C nur gelegentlich neue Methoden benötigt,die dann bei dieser Gelegenheit bekannt gemacht werden müssen.

Bei einem Einstieg in die graphische Entwicklungsumgebung,ohne vorher die textorientierte Version zu behandeln,ist zunächst vor der eigentlichen Programmerstellung ein problemorientierter Einstieg in das jeweilige Graphenproblem notwendig,als dessen Ergebnis sich schließlich ein Verbalalgorithmus ergibt,der dann objektorientiert und mittels graphischer Anzeige als Quellcode realisiert werden kann.

Entsprechende Beschreibungen von Unterichtsreihen befinden sich im Kapitel C Unterrichtsskizzen.Dieses Kapitel enthält auch den kommentierten Quellcode des jeweiligen Algorithmus sowie geeignete Anwendungsaufgaben,um den Algorithmus zu testen oder einzuüben.Darüber hinaus sind,soweit notwendig,außerdem Beweise bzw. Erläuterungen zur Herleitung bzw. Korrektheit der Algorithmen angegeben.

Gemäß der in den vorigen Abschnitten dieses Kapitels bisher beschriebenen Unterrichtsreihen waren bei den Schülern schon Vorkenntnisse des Hamilton- und Eulerproblems (Klausuraufgabe) auf Grund der Erstellung der entsprechenden objektorientierten Textanwendungen (Konzeption CAK) bei den Schülern vorhanden,wovon ab jetzt ausgegangen werden soll.Dann fällt der Umstieg auf die neue objektorientierte Entwicklungsumgebung besonders leicht.Im Folgenden soll jetzt dargestellt werden,durch welche kleinen Änderungen am jeweiligen textorientierten Algorithmus die entsprechende graphisch orientierten Methoden erstellt werden können (vgl. dazu auch Kapitel C V).Es empfiehlt sich der Übersichtlichkeit halber für die mathematischen Anwendungen eine neue Unit UMath zu erstellen,in die der Quellcode eingefügt wird.(Die Unit UForm enthält dann notwendigerweise zusätzlich die entsprechende uses-Anweisung:uses UMath.)

Bei den Methoden zum Hamilton-Problem braucht nur weniges geändert zu werden:

Zunächst wird völlig analog zu der textorientierten Version ein neuer Objekttyp THamiltongraph jetzt von TInhaltsgraph abgeleitet:

```
THamiltongraph = class(TInhaltsgraph)
    constructor Create;
    procedure Hamilton(Stufe:Integer;
     Kno,Zielknoten:TInhaltsknoten;var Kliste:TKantenliste;Flaeche:TCanvas);
    procedure Hamiltonkreise(Flaeche:TCanvas;Ausgabe:TLabel;var SListe:TStringlist);
  end;
```

Der Constructor gestaltet sich in gleicher Weise:

```
constructor THamiltongraph.Create;
begin
  inherited Create;
end;
```

Die Methode Hamilton enthält jetzt (nur) einige kleinere Änderungen,die mit {****} markiert sind.

```
procedure THamiltongraph.Hamilton(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;Var Kli-
ste:TKantenliste;
  Flaeche:TCanvas);
var Index:Integer;
    Zkno:TInhaltsknoten;
    Ka:TInhaltskante;
begin
  if not Kno.AusgehendeKantenliste.leer then
    for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
    begin
      Ka:=TInhaltskante(Kno.AusgehendeKantenliste.Kante(Index));
      Zkno:=TInhaltsknoten(Kno.AusgehendeKantenliste.Kante(Index).Zielknoten(Kno));
      if ((Not Zkno.Besucht) and (Stufe<AnzahlKnoten+1)) or
        ((Zkno=Zielknoten)and (Stufe=AnzahlKnoten)and(Stufe>2) )
      then
      begin
        Ka.Pfadrichtung:=Zkno;
        if Demo then                              {****}
        begin
          Ka.Farbe:=clred;
          Ka.Stil:=psdot;
          Ka.ZeichneKante(Flaeche);
          Demopause;
        end;                                      {****}
        Zkno.Besucht:=true;
        Kliste.AmEndeanfuegen(Ka);
        Stufe:=Stufe+1;
        if (Zkno=Zielknoten)and (Stufe=AnzahlKnoten+1)
        then
          Zielknoten.Pfadliste.AmEndeanfuegen(Kliste.Kopie.Graph) {****}
        else
          Hamilton(Stufe,Zkno,Zielknoten,Kliste,Flaeche);
        Stufe:=Stufe-1;
        if Zkno<>Zielknoten then Zkno.Besucht:=false;
        if not Kliste.Leer then
        begin
          if Demo then                            {****}
          begin
            Ka.Farbe:=clblack;
            Ka.Stil:=pssolid;
            Ka.ZeichneKante(Flaeche);
            Demopause;
          end;                                    {****}
          Kliste.AmEndeloeschen(TObject(Ka));;
        end;
      end;
    end;
end;
```

Um im Demomodus die Anzeige der Pfade zu ermöglichen,wird (zweimal) folgende Quellcodesequenz eingefügt:

```
if Demo then
          begin
            Ka.Farbe:=clred/clblack;
            Ka.Stil:=psdot/pssolid;
            Ka.ZeichneKante(Flaeche);
            Demopause;
          end;
```

Die Sequenz benutzt die neuen Propertys Farbe und Stil einer Kante vom Typ TInhaltskante sowie die Methode ZeichneKante,um eine Kante auf der Zeichenflaeche darzustellen.Außerdem wird ein Pfad jetzt nicht sofort als Knotenfolge ausgegeben,sondern mittels

```
Zielknoten.Pfadliste.amEndeanfuegen(Kliste.Kopie.Graph)
```

in der Pfadliste des Zielknotens,der gleich dem Startknoten des Pfades ist,als Kopie abgespeichert und steht so für eine spätere Anzeige und Ausgabe in der folgenden Methode bereit.
Eine andere Möglichkeit ist es hier die Methode Graphzeichnen einzusetzen, und die Kantenliste mittels Kliste.Graph in einen Graph umzuwandeln,so dass die Anzeige der Hamiltonkreise direkt (ohne Zwischenspeichern) erfolgt.Allerdings muß dann die Ermittlung der Traveling-Salesman-Lösung anders erfolgen.

```
procedure THamiltongraph.Hamiltonkreise(Flaeche:TCanvas;Ausgabe:TLabel;var SLi-
ste:TStringlist);
var Kno:TInhaltsknoten;
    MomentaneKantenliste:TKantenliste;
    zaehl:Integer;
    T:TInhaltsgraph;
begin
  if Leer then exit;
  MomentaneKantenliste:=TKantenliste.Create;
  LoescheKnotenbesucht;
  Pfadlistenloeschen;
  Kno:=LetzterMausklickknoten;
  Kno.Besucht:=true;
  Ausgabe.Caption:='Berechnung läuft';
  Hamilton(1,Kno,Kno,MomentaneKantenliste,Flaeche);
  Kno.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
  FaerbeGraph(clblack,pssolid);
  ZeichneGraph(Flaeche);
  T:=TInhaltsgraph(Kno.MinimalerPfad(Bewertung));      {****}
  if not T.Leer then
  begin
    Ausgabe.Caption:='Traveling Salesmann Lösung: '+
    T.InhaltallerKnoten(ErzeugeKnotenstring)+' Summe: '+
      RundeZahltostring(t.Kantensumme(Bewertung),Kantengenauigkeit)
      +' Produkt: '+
      RundeZahltostring(T.Kantenprodukt(Bewertung),Kantengenauigkeit);
    SListe.Add(Ausgabe.Caption);
    T.FaerbeGraph(clred,psdot);
    T.ZeichneGraph(Flaeche);
    ShowMessage(Ausgabe.Caption);
    T.FaerbeGraph(clred,psdot);
    T.ZeichneGraph(Flaeche);
    MomentaneKantenliste.Free;
    MomentaneKantenliste:=nil;
  end;
  Ausgabe.Caption:='';
  Ausgabe.Refresh;
end;
```

In der die Methode Hamilton aufrufenden Methode Hamiltonkreise erfolgt die Anzeige der in der Pfadliste gespeicherten Kantenlisten durch die Methode ZeichnePfadliste des Startknoten,der per Mausklick durch die Methode LetzterMausklickknoten bestimmt wird.Die Methode ZeichnePfadliste erzeugt außerdem eine Stringliste SListe vom Typ TStringlist,die schon die Knoteninhalte des Pfades als strings enthält.Durch die Methode MinimalerPfad des Startknotens wird (anders als in der textorientierten Version) die Traveling-Salesman-Lösung erzeugt,deren Knotenwerte durch die Methode InhaltallerKnoten für die Ausgabe im Label Ausgabe bereitgestellt werden.Durch Kantensumme und Kantenprodukt werden die entsprechenden Summen und Produkte der Pfade bestimmt.Die Methode FaerbeGraph sorgt dafür,dass die Hamiltonkreise rot-markiert mittels ZeichneGraph gezeichnet werden.

Zum Aufruf der Methode Hamiltonkreise muß jetzt in der neuen Form Knotenformular (Unit UForm) ein neues Menü Hamiltonkreise eingefügt werden.Die folgende (gegenüber dem entsprechenden Quellcode des Programms Knotengraph im Anhang vereinfachte) Methode ist die zugehörige Ereignismethode.

```
procedure TKnotenform.HamiltonkreiseClick(Sender: TObject);
var Sliste:TStringList;
    Hamiltongraph:THamiltongraph;
begin
    if Graph.Leer then exit;
    if Graph.AnzahlKomponenten>1 then
    begin
```

```
      ShowMessage('Mehrere Komponenten!');
      exit;
    end;
    Hamiltongraph:=THamiltongraph(Graph.InhaltskopiedesGraphen(THamiltongraph,TInhaltsknoten,
      TInhaltskante,false));
    Hamiltongraph.Pfadlistenloeschen;
    Sliste:=TStringList.Create;
    Hamiltongraph.Hamiltonkreise(Paintbox.Canvas,Ausgabe1,Sliste);
    StringlistnachListbox(Sliste,Listbox);
    Hamiltongraph.Freeall;
    Hamiltongraph:=nil;
end;
```

Um den ursprünglichen Graphen nicht zu beeinflussen, wird zunächst ein neuer Graph vom Typ THamiltongraph Hamiltongraph erzeugt,und mittels der Methode InhaltskopiedesGraphen wird aus dem vorhandenen Graph TKnotenformular.Graph der neue Graph mit entsprechender Knoten-und Kantenstruktur erzeugt.Nachdem die Methode Hamiltonkreise aufgerufen worden ist,wird mittels der Methode StringlistnachListbox die Liste SListe in das Ausgabefenster kopiert.

Mit der Besprechung dieser Methoden hat man schon einen Großteil der auch für die Erstellung des Quellcodes der weiteren mathematischen Algorithmen benötigten Methoden bereitgestellt.Da sie an dieser Stelle zum ersten Mal neu eingesetzt werden,müssen die obigen Methoden unter Führung des Lehrers entwickelt werden.

In der gleichen Weise kann jetzt der Algorithmus geschlossene Eulerlinie für die Bestimmung der offenen Eulerlinie siehe Kapitel C V) mit Hilfe der neuen Entwicklungsumgebung dargestellt werden.Die Methoden kommen mit den eben erläuterten schon bekannten Methoden aus,und können nun als Schülerprojekt gestellt werden:

Lösung (gegenüber der Darstellung in C V sowie dem Quellcode von Knotengraph im Anhang vereinfacht):

```
TEulergraph   = class(TInhaltsgraph)
    constructor Create;
    procedure Euler(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;var Kliste:TKantenliste;
      Flaeche:TCanvas);
    procedure Eulerlinie(Flaeche:TCanvas;Ausgabe:TLabel;var SListe:TStringlist;
      Anfangsknoten,Endknoten:TInhaltsknoten);
  end;

constructor TEulergraph.Create;
begin
  inherited Create;
end;


procedure TEulergraph.Euler(Stufe:Integer;
Kno,Zielknoten:TInhaltsknoten;var Kliste:TKantenliste;Flaeche:TCanvas);
var Index:Integer;
    Zkno:TInhaltsknoten;
    Ka:TInhaltskante;
begin
  if not Kno.AusgehendeKantenliste.Leer then
    for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
    begin
      Ka:=TInhaltskante(Kno.AusgehendeKantenliste.Kante(Index));
      Zkno:=TInhaltsknoten(Kno.AusgehendeKantenliste.Kante(Index).Zielknoten(Kno));
      if ((not Ka.Besucht) and (Stufe<=AnzahlKantenmitSchlingen)) then
      begin
        Ka.Pfadrichtung:=Zkno;
        Ka.Besucht:=true;
        if Demo then
        begin
          Ka.Farbe:=clred;
          Ka.Stil:=psdot;
          Ka.ZeichneKante(Flaeche);
          Demopause;
        end;
        Kliste.AmEndeanfuegen(Ka);
        if Demo then Ausgabe.Caption:=Kliste.Kantenlistealsstring;
        Stufe:=Stufe+1;
        if (Zkno=Zielknoten)and (Stufe=AnzahlKantenmitSchlingen+1)  then
          Zielknoten.Pfadliste.AmEndeanfuegen(Kliste.Kopie.Graph);
```

```
        else
          Euler(Stufe,Zkno,Zielknoten,Kliste,Flaeche);
        Stufe:=Stufe-1;
        Ka.Besucht:=false;
        if Demo then
        begin
          Ka.Farbe:=clblack;
          Ka.Stil:=pssolid;
          Ka.ZeichneKante(Flaeche);
          Demopause;
        end;
        if not Kliste.Leer then
          Kliste.AmEndeloeschen(TObject(Ka));;
      end;
    end;
end;

procedure TEulergraph.Eulerlinie(Flaeche:TCanvas;Ausgabe:TLabel;var SListe:TStringlist;
  Anfangsknoten,Endknoten:TInhaltsknoten);
var MomentaneKantenliste:TKantenliste;
    Zaehl:Integer;
    T:TInhaltsgraph;
begin
  if Leer then exit;
  MomentaneKantenliste:=TKantenliste.Create;
  LoescheKantenbesucht;
  Pfadlistenloeschen;
  Ausgabe.Caption:='Berechnung läuft';
  Ausgabe.Refresh;
  Euler(1,Anfangsknoten,Endknoten,MomentaneKantenliste,Flaeche);
  Endknoten.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
  if not Endknoten.Pfadliste.leer
  then
  begin
    T:=TInhaltsgraph(Endknoten.Pfadliste.Pfad(0));
    if not T.Leer then
    begin
      Ausgabe.Caption:='Eulerlinie: '+
      T.InhaltallerKnoten(ErzeugeKnotenstring)+' Summe: '+
        RundeZahltostring(T.Kantensumme(Bewertung),Kantengenauigkeit)
        +' Produkt: '+
      RundeZahltostring(t.Kantenprodukt(Bewertung),Kantengenauigkeit);
      Ausgabe.Refresh;
      T.FaerbeGraph(clred,psdot);
      T.ZeichneGraph(Flaeche);
      ShowMessage(Ausgabe.Caption);
    end;
  end;
  Ausgabe.Caption:='';
  Ausgabe.Refresh;
end;

procedure TKnotenform.EulerliniegeschlossenClick(Sender: TObject);
var Sliste:TStringList;
    Eulergraph:TEulergraph;
begin
 if Graph.Leer then exit;
  if Graph.AnzahlKomponenten>1 then
  begin
    ShowMessage('Mehrere Komponenten!');
    exit;
  end;
  Eulergraph:=TEulergraph(Graph.InhaltskopiedesGraphen(TEulergraph,TInhaltsknoten,
    TInhaltskante,false));
  Sliste:=TStringList.Create;
  Eulergraph.ZeichneGraph(Paintbox.Canvas);
  Eulergraph.Eulerlinie(Paintbox.Canvas,Ausgabe1,Sliste,F.letzterMausklickknoten,
  Eulergraph.letzterMausklickknoten);
  StringlistnachListbox(Sliste,Listbox);
  Eulergraph.Freeall;
  Eulergraph:=nil;
end;
```

Da der wesentliche Algorithmus der textorientierten Version erhalten bleibt,läßt sich der Übergang zur graphisch orientierten Version in nur wenigen Unterrichtsstunden vollziehen.

Nach der Art dieser Algorithmen können jetzt alle weiteren mathematische Anwendungen mit graphischer Oberfläche erstellt werden.Die Beschreibung dieser Anwendungen mit Erläuterungen des zugehörigen Quellcodes findet sich

in den Kapiteln des Abschnittes C „Unterrichtsskizzen",die die Fortsetzung der Ausführungen dieses Abschnittes ist.

7) Der Einsatz des Programms Knotengraph als methodisch-didaktisches Werkzeug zur Veranschaulichung von Graphenalgorithmen

(Konzeption DWK)

Während bei der bisherigen Darstellung der Aspekt der objektorientierten Programmierung von Graphenalgorithmen einerseits mit Hilfe der Entwicklungsumgebung Knotengraph bzw. andererseits als textorientierte Consolenanwendung im Vordergrund stand, soll jetzt als dritte Möglichkeit der Einsatz des (fertigen) Programms Knotengraph als methodisch-didaktisches Werkzeug zur Veranschaulichung von Graphenalgorithmen im Mathematik- oder Informatikunterricht erörtert werden.

Genaue Beschreibungen von entsprechenden Unterrichtsreihen,die sowohl unter dem Aspekt der rein mathematischen Behandlung ohne zu Programmieren als auch als Programmierungsprojekt eingesetzt werden können,finden sich in den entsprechenden Kapiteln des Abschnittes C Unterrichtsskizzen.

Lässt man die in den Kapiteln des Abschnittes C enthaltenen Quellcodedarstellungen sowie die dazu gehörenden Erläuterungen weg,und beschränkt bzw. konzentriert sich stattdessen jeweils auf die Verbalbeschreibung der Graphenalgorithmen, eignet sich der jeweilige dort dargestellte Unterichtsgang auch für den Einsatz in einem Kurs,in dem Graphentheorie ohne Bezugnahme auf eine Programmiersprache vermittelt werden soll.

Da unter dieser Voraussetzung die durch die Erstellung und das Testen von Quellcode gewonnene dynamische Erfahrung des Ablaufs eines Graphenalgorithmus fehlt,bietet es sich als Ersatz an,den Ablauf der Algorithmen mittels des Demo-Modus des (fertigen) Programms Knotengraphs als Abfolge von Graphenzuständen zu veranschaulichen (vgl. dazu auch die zugehörigen Unterrichtspläne im Anhang).

Der Demo-Modus oder Einzelschrittmodus gestattet es,die Einzelschritte der Algorithmen der Menüs Pfade und Anwendungen (einstellbar) zeitverzögert und farblich hervorgehoben zu verfolgen.

Um die Graphenalgorithmen im Unterricht erarbeiten zu können,empfiehlt es sich an Hand geeigneter Einstiegsaufgaben die Problemstellung herauszuarbeiten und die Algorithmen sowie ihre Lösungen durch Aufruf der Algorithmen der Menüs des Programm Knotengraphs zu verfolgen.Die Kapitel des Abschnitts C Unterrichtsskizzen stellen dazu eine Fülle von Material in Form von Problemstellungen der algorithmischen Graphentheorie als auch des Operations Research sowie der zur Lösung geeigneten Verbalform der Algorithmen mit deren mathematischen Begründungen zur Verfügung.

Die in diesem Kapitel durchgängig beschriebene Unterrichtsreihe in der Jahrgangsstufe 12,die beginnend mit dem Einstiegsprojekt zur objektorientierten Programmierung (Abschnitt 2) und dem Projekt Objekt als Liste (Abschnitt 3) schließlich zum Aufbau einer Graphenstruktur mit textorientierter Ausgabe führte (Abschnitt 4),woran sich schließlich als mathematische Anwendungen das Euler-und Hamiltonproblem sowie der Tiefe-Baumdurchlauf und das Problem minimaler und aller Pfade zwischen zwei Knoten zunächst als Consolenanwendung (Abschnitt 4 und 5) und später als Graphikanwendung mittels der Entwicklungsumgebung Knotengraph (EWK) anschloss (Abschnitt 6),wurde beispielsweise schließlich unter Einsatz des (fertigen) Programms Knotengraph (DWK) als Ergänzung durch eine Sequenz mit den Themen Breiter Baumdurchlauf,Binärbäume, Erstellung des minimalen Gerüstes eines Graphen,Maximaler Netzfluss und Maximales Matching abgeschlossen.In Jahrgangsstufe 13 wurde dann noch der endliche Automat im Rahmen einer Unterrichtsreihe zur Theoretischen Informatik durch dieses Programm demonstriert.

Weil den Schülern auf Grund des vorangegangenen Unterrichts der prinzipielle Aufbau des Programms Knotengraph schon bekannt war,ergab sich hier be-

sonders einfaches Arbeiten,und die Aufmerksamkeit konnte voll auf die mathematische Problemstellung konzentriert werden.

Stehen Programmierungserfahrungen nicht zur Verfügung,da man den Einstieg in die algorithmische Graphentheorie direkt mittels des (fertigen) Programms Knotengraph sucht,bietet es sich an zunächst mit dem Begiff des Baums bzw. Suchbaums und einer Erläuterungen der Vorteile beim Suchen von Daten sowie einiger Sätze über die Eigenschaften von Bäumen gemäß dem Unterrichtsgang,der in Kapitel C I der Unterichtskizzen dargestellt ist,zu beginnen.Daran schließen sich dann die in Kapitel C II beschriebenen miteinander sehr verwandten Durchlaufstrategien in Graphen Tiefer Baum,Alle Pfade,Minimale Pfade,Kreise und Pfade zwischen zwei Knoten (C III) usw. an. Die ganze Unterichtsreihe läßt sich unter dem Titel „Bäume und ihre Eigenschaften“ zusammenfassen,wobei sich als Zusatz die Bestimmung von Pfadwahrscheinlichkeiten in Bäumen anbietet(siehe Ende von C XII).

Wendet man die Algorithmen mittels des Programms Knotengraph jeweils auf ein und denselben Graph an und benutzt den Demo-Modus,lassen sich sehr deutlich die Unterschiede im Ablauf der verschiedenen Algorithmen demonstrieren.

Es bietet sich dann an als Spezialfälle auch die Algorithmen von Kruskal und Dijkstra (C III) in den Unterrichtsgang mit einzubeziehen.Führen doch beide Algorithmen schon von einer bloßen Durchlaufstrategie hin zu einer Optimierungsaufgabe,nämlich das minimale Gerüst eines Graphen bzw. den kürzesten Pfad zwischen zwei Knoten zu suchen.

Weitere Optimierungsaufgaben lassen sich dann dem Gebiet des Operations Research entnehmen,und das Thema Optimierung läßt sich beispielsweise fortsetzen mit den aufeinander aufbauenden Themen maximaler Netzfluss (C IX) sowie maximales Matching (C X),und man kann dann schließlich zum allgemeinen Transportproblem (C XIII) und dem Simplexproblem (auf Graphen) gelangen,das auf dem minimalen Kostenproblem (C XIII) beruhend,seine Fortsetzung in den damit zusammenhängenden Algorithmen Hitchcockproblem, Optimales Matching (C XIII) und Chinesischer Briefträger (C XIV) hat.Beim Thema Optimierung bietet sich auch die Erstellung eines Netzzeitplans nach der CPM-Methode an (C VI) an.Auf diese Weise läßt sich anschaulich experimentell eine Unterrichtsreihe zum Thema diskrete,endliche Optimierungsverfahren im Mathematikunterricht gestalten.

Eine andere Möglichkeit ist es Themen wie die Eulerbeziehung,Planarität und Färben von Graphen,dargestellt als miteinander verknüpfte Problemstellungen in Kapitel C IV (vgl. dazu auch den entsprechenden Unterrichtsplan im Anhang) oder die graphische Darstellung von Relationen (C VIII) zu behandeln,sowie das Programm zur Unterstützung eines Kursus zum Thema Wahrscheinlichkeitsrechnung, speziell Markovketten (C XII) oder zur Lösung von linearen Gleichungssystemen (C IX) einzusetzen.

Durch die Möglichkeit mit Hilfe des Programms Knotengraph relativ schnell auch umfangreichere Graphen erzeugen und auch wieder abändern und den Ablauf eines Algorithmus im Demo-Modus verfolgen zu können,lassen sich auch komplexere Verfahren auf Graphen anschaulich erläutern.Dadurch können algorithmische Graphenverfahren für Schüler motivierend dargestellt werden,weil jeder Schüler selbst unter Verwendung von verschiedenen Graphen oder unterschiedlicher Parameter mit dem Algorithmus experimentieren kann. Auf diese Weise läßt sich das fertige Programm Knotengraph als Werkzeug zur Entdeckung und Veranschaulichung von Graphenalgorithmen und von Grapheneigenschaften benutzen (vgl. dazu den entsprechenden Unterrichtsplan zum Thema Fluss- und Kostenprobleme im Anhang).

In der oben genannten Unterrichtsreihe zum Thema Baum läßt sich das (fertige) Programm Knotengraph als didaktisch-methodisches Werkzeug beispielsweise zur Veranschaulichung folgender Sachverhalte einsetzen (vgl. für detaillertere Informationen auch das Kapitel C I der Unterrichtsskizzen):

a)Selbständiges experimentelles Entdecken des Zusammenhangs zwischen der Knoten-und Kantenzahl eines vollständigen Graphen,des Zusammenhangs zwischen der Kantenzahl und der Summe der Knotengrade in ungerichteten Graphen sowie einer Beziehung für die Anzahl der Knoten ungeraden Grades durch die Schüler.

b)Selbstständiges experimentelles Entdecken eines Zusammenhangs zwischen der Kantenzahl und Knotenzahl eines Baumes sowie des Zusammenhangs zwischen der Kantenzahl bei einem Graphen und dessen Gerüstbaums durch die Schüler.Auf Grund der gewonnenen Ergebnisse läßt sich die Definition eines Baumes erarbeiten.

C)Graphische Veranschaulichung (mit der Möglichkeit der Anzeige von Einzelschritten) von Durchlaufstrategien wie tiefer und breiter Baumdurchlauf in beliebigen Graphen, als auch darauf aufbauend das Suchen aller Pfade,das Suchen minimaler Pfade sowie das Suchen von Kreisen als auch die graphische Veranschaulichung der Algorithmen (mit der Möglichkeit der Anzeige von Einzelschritten) nach Dijkstra (alle Pfade und minimaler Pfad zwischen zwei Knoten) und Kruskal (minimales Gerüst/Baum).

Dadurch können die entsprechenden Algorithmen je nach Unterrichtskonzeption vom Schüler entweder neu entdeckt oder aber,falls den Schülern schon bekannt, anschaulich nachvollzogen werden.

d)Durch Erzeugen von entsprechenden Graphen rasches Programmunterstütztes und dadurch motivierendes Lösen von Anwendungs- und Übungsaufgaben zum Thema Bäume und Pfade,wie z.B. von Umschüttungsaufgaben, Routingproblemen sowie von topologischen Problemen oder Optimierungsproblemen wie z.B. das Verlegen von (Telefon-) Leitungen.

Das Verstehen der Algorithmen fällt den Schülern insbesondere dann nicht schwer,wenn schon fundamentale Erfahrungen mit Graphenalgorithmen auf Grund der eigenen objektorientierten Programmierung,wie in den vorigen Abschnitten beschrieben,vorhanden sind.Sollten diese Kenntnisse z.B. bei der Behandlung des Themas in einem Mathematikkurs ohne Programmierungskenntnisse fehlen,muß insbesondere für die Entdeckung und Erläuterung der Algorithmen sicher mehr Zeit zur Verfügung gestellt werden.

Zu dem oben genannten Themenkomplex Titel „Bäume und ihre Eigenschaften“, zu dessen Behandlung ca. 10 Unterrichtsstunden nötig waren,wurde folgende Klausuraufgabe gestellt,die in den letzten beiden Teilen e) und f) Programmierkenntnisse voraussetzt.Werden diese Teilaufgaben durch die nachfolgend dargestellten alternativen Teilaufgaben e) bis g) ersetzt,ist die so veränderte Aufgabe zu diesem Thema auch für einen reinen Mathematikkurs Algorithmische Graphentheorie ohne Programmierungskenntnisse geeignet.

Aufgabe:

Drei Krüge A,B und C fassen 3 l,2 l und 1 l.Nur A ist anfangs mit 3 l gefüllt.B und C sind leer.Zustand:(3/0/0) Durch bloßes Umschütten durch die durch die Krüge vorgegebenen Flüssigkeitsmengen in Litern (ohne sonstiges Maß für die Flüssigkeitsmengen) soll erreicht werden,dass am Schluß in B 2 l und in C 1 l vorhanden ist.Der Behälter A soll leer sein. Zustand: (0/2/1)

a)Wie muß jeweils umgeschüttet werden,damit der Zustand (0/2/1) vom Zustand (3/0/0) erreicht wird? Beschreiben Sie die Umfüllzustände durch die Kennzeichnung (a,b,c),und geben Sie eine Kette von Umfüllzuständen bis zum gewünschten Endzustand an:d.h. verlangt sind die Zwischenzustände von (3/0/0) nach (0/2/1).

Geben Sie insgesamt noch zwei weitere mögliche Umfüllmöglichkeiten als Umfüllkette an.

Lösen sie diese Aufgaben zunächst durch gezieltes Probieren ohne Verwendung eines Graphen.

Im folgenden soll jetzt mit Hilfe eines Graphen eine Übersicht über alle möglichen Lösungen gegeben werden:

b)Erstellen Sie dazu eine Tabelle,die alle zulässigen Füllungen der Krüge,die durch Umschüttungen entstehen können,enthält.Zustandsbschreibung mittels: (a/b/c) (dabei sind a,b,c die Füllzahlen der Krüge)

Nummerieren Sie die einzelnen Füllungsmöglichkeiten als Zustände fortlaufend durch.Geben Sie in einer weiteren Tabellenspalte an,welche Zustände jeweils vom aktuellen Zustand durch Umfüllen erreicht werden können.

Tabelle:

(Der Übergang vom Zustand 1) ist als Beispiel schon vorgegeben.Die übrigen leeren Zellen sind auszufüllen.Tabelle ins Heft übertragen!Die Zahlen (Zustände) der Zeilen der Spalte Übergang zu sind aufsteigend zu ordnen,also in Zeile 1: 2),4) statt 4),2)))

Zustand:	Krug A:	Krug B:	Krug C:	Übergang zu:
1)	3	0	0	2),4)
2)	2	0	1	
3)	2	1	0	
4)	1	2	0	
5)	1	1	1	
6)	0	2	1	

c)Zeichnen Sie zu dem Umfüllproblem einen Graphen,in dem die Füllungszustände durch Knoten und das Umfüllen,d.h. der Übergang von einem Zustand zu anderen möglichen Zuständen durch gerichtete Kanten dargestellt wird.Die Bewertung der Kanten ist dabei die Flüssigkeitsmenge in l,die umgeschüttet wird.Eine mögliche Lösung des Umfüllproblems ist dann ein Pfad vom Knoten (3/0/0) zum Knoten (0/2/1).

d)Bestimmen Sie vom Knoten (3/0/0) aus alle tiefen Baumpfade jeweils in Postorder- und in Preorder- Reihenfolge.(Kantenfolge der ausgehenden Kanten von einem Knoten/Zustand sei gemäß der aufsteigenden Nummerierung der Zielknoten/Zustände in der Tabelle von Aufgabe b) gewählt,d.h. bei Zustand 1 ist die Kantenreihenfolge die Kante nach Zustand 2 (2/0/1) und danach die Kante nach Zustand 4 (1(2/0).)
Wieso läßt sich mit Hilfe des Tiefen-Baum-Algorithmus das oben gestellte Umfüllproblem lösen?Definieren und erläutern Sie den Durchlauf nach Preorder,Postorder und Inorder (letzteres für einen Binärbaum) eines Baumes.Definieren Sie den Begriff des Baums.

Falls Programmierungskenntnisse vorhanden sind:

e)Schreiben Sie eine Methode procedure TPfadknoten.ErzeugeTiefeBaumPfade mit TPfadknoten = class(TInhaltsknoten),die alle tiefen Baumpfade in Preorder-Reihenfolge vom (die Methode besitzenden) Pfadknoten aus erzeugt und in seiner Pfadliste als Graphen-(Pfade) (Umwandlung einer Kantenliste in einen Graph mittels der Methode TKantenliste.Graph) speichert.Was muß geändert werden,damit die Baumpfade in Postorder-Folge erzeugt werden?.
(Setzen Sie dazu und für die folgende Teil-Aufgabe die in den Units UList,UGraph und UInhGrph enthaltenden Methoden voraus.)

f)Ändern Sie die Methode ErzeugeTiefeBaumpfade so um,dass Sie **alle** Pfade und den **minimalen** Pfad bezüglich der oben vorgegeben Kantenbewertung zwischen zwei Knoten bestimmt (Kanten-Bewertung Wert vom Typ TWert gleich Anzahl Liter,die zwischen zwei Zuständen d.h. Knoten umgeschüttet werden).Die Durchlauf erfolge jetzt gemäß der Preorder-Reihenfolge. Die Methode erhält dann den Namen procedure TPfadKnoten.ErzeugeAllePfadeundMinimalenPfad (ZKno:PfadKnoten;var Minlist:TKantenliste).Der Ausgangsknoten ist der die Methode besitzende Knoten vom Typ TPfadknoten.Der Endknoten ist der Knoten ZKno (Zielknoten) vom Typ TPfadknoten.Die zwischen den Knoten gefundenen Pfade werden von der Methode als Graphen wiederum in der Pfadliste des Ausgangsknotens gespeichert.Der minimale Pfad wird als Kantenliste in Form der

Kantenliste Minlist zurückgegeben.Es wird eine Function Bewertung vom Typ TWert vorausgesetzt,die die Kantenbewertung zurückgibt.

Hinweise:

1)Benutzen Sie zur Auswahl des minimalen Pfades die Methode function TListe.WertSummederElemente(Wert:TWert):Extended aus der Unit UList,die die Wertsumme der Elemente einer (z.B. Kanten-) Liste bezüglich der Bewertung Wert bestimmt.

2)Der Methode AllePfadeundminimalerPfad wird als Kantenliste Minlist die Kantenliste des Graphen als Kopie beim Aufruf übergeben.

Geben Sie für den obigen Graphen des Umfüllproblems den minimalen Pfad (bezüglich obiger Kanten-Bewertung) zwischen dem Knoten zum Zustand (3/0/0) und dem Knoten zum Zustand (0/2/1) als Folge von (Knoten-)Zuständen der Form (a/b/c) an.

Alternativ statt den obigen Aufgabe e) und f) (falls keine Programmierungskenntnisse vorhanden sind):

e)Geben Sie eine verbale Beschreibung des Algorithmus von Dijkstra an.Suchen Sie nach dem Algorithmus von Dijkstra den minimalen Pfad (bezüglich obiger Kantenbewertung) vom Anfangszustand (3/0/0) zum Endzustand (0/2/1).
Dokumentieren Sie das Vorgehen durch Notierung aller (auch versuchsweise) durchlaufenden Pfade bzw. des entsprechenden Baums.

f)Bestimmen Sie ein minimales Gerüst des Graphen nach dem Algorithmus von Kruskal.Welche Bedeutung hat die Lösung für das Umfüllproblem?

g)Ermitteln Sie,ob der Graph einen HamiltonKreis enthält,und bestimmen Sie gegebenenfalls diesen Kreis durch Angabe der Knotenfolge.Welche Bedeutung hat bzw. hätte dieser Kreis für das Umfüllproblem?

Die Lösungen dieser Umfüllaufgabe befinden sich im Anhang.

Der Schwierigkeitsgrad der Teilaufgaben ist in die Kategorien Wissen und Transfer einzuordnen,da die grundlegenden Verfahren im Unterricht besprochen wurden.

Wie oben erwähnt,könnte eine Weiterführung der Unterrichtsreihe (aber auch ein Neubeginn oder Neueinstieg,da die Themen auch voneinander unabhängig zu behandeln sind) jedes der im folgenden genannten Themen behandeln:maximaler Fluss, maximales Matching, Probleme des Operations Research wie minimale Kosten, optimales Matching, Transportprobleme, Briefträgerproblem, oder aber das Aufstellen eines Netzwerkzeitplanes,Darstellung und Eigenschaften einer Relation,Darstellung und Simulation eines endlichen Automaten (im Informatikunterricht),das anschauliche Lösen von Gleichungssystemen, und die Reduzierung von Signalflussgraphen sowie die Ermittlung von Wahrscheinlichkeiten durch die Darstellung von (absorbierenden oder stationären) Markovketten als Graphen.

Unter dem Gesichtspunkt der rein mathematischen Behandlung soll jetzt für die verschiedenen genannten Themen die Vorteile des Einsatzes des (fertigen) Programms Knotengraphs als didaktisch-methodisches Werkzeug im Überblick beschrieben werden.(Vergleiche dazu auch die Bemerkungen in den einzelnen Kapiteln des Abschnittes C)

<u>Netzwerkzeitplan (C IV):</u>

Veranschaulichung der CPM-Methode sowie der den Zeitplan beschreibenden Größen wie Pufferzeit,frühstmögliche Anfangszeit usw. sowie Darstellung des kritischen Pfades

Programmgestütztes Lösen von realistischen Anwendungsaufgaben

Endlicher Automat (C VII):

Visuelle Darstellung der Steuerung eines endlichen Automats

Simulation der Lösung von Problemen der technischen oder theoretischen Informatik

Graph als Relation (C VIII):

Graphische Darstellung einer Relation sowie anschauliches Erzeugen der reflexiven und transitiven Hülle.Dadurch wird die selbstständige Entdeckung oder gelenkte Erarbeitung von Begriffen wie Transitivität, Reflexivität, Symmetrie, Antisymmetrie, Äquivalenz im Unterricht ermöglicht.

Visualisierung der Methode des toplogischen Sortieren zur Erzeugung einer globalen Ordnung (Fortsetzung der Tiefensuche)

Maximaler Netzfluss (C IX):

Anschauliche Demonstration des Algorithmus von Ford-Fulkerson

Lösen von einfachen Optimierungsproblemen wie Transportprobleme

Maximales Matching (C X):

Veranschaulichung und Motivation der Aussagen des Heiratssatzes von Hall sowie Lösung von entsprechenden Anwendungsaufgaben

Demonstration des Algorithmus „Erweiternden Weg suchen“ zur Bestimmung eines maximalen Matchings

Gleichungssystem (C XI):

Anschauliche Lösung eines Gleichungssystems nach dem Algorithmus von Mason

Durch Zurückführung auf ein Gleichungssystem programmgestützt den Fluss in Signalflussgraphen bestimmen.

Markovkette (absorbierend und stationär) und Graph reduzieren (C XII):

Die Anwendung der Mittelwertregeln bei Markovketten veranschaulichen.

Methoden der Reduzierung von Markovketten oder Signalflussgraphen anschaulich demonstrieren.

Den Lernprozess Erstellung einer Markovkette zu einem gegebenen realen Problem (Modellierung) besser dadurch zu motivieren,dass die Lösung anschließend mit Programmhilfe ermittelt werden kann.

Veranschaulichung der Aussagen der Hauptsätze zu absorbierenden und stationären Markovketten

Simulation eines Spiels zur Bestimmung der Lösung der Übergangswahrscheinlichkeiten von Markovketten

Minimale Kosten, Transportprobleme, Optimales Matching, Chinesisches Briefträgerproblem (C XIII und C XIV):

Die Lösungsstrategie von Aufgaben des Operations Research anschaulich verfolgen können.(z.B. Simplex-Algorithmus auf Graphen)

Realistische Aufgabenstellungen des Operations Research programmgestützt lösen zu können.Dadurch entsteht Motivation zum Aufstellen von Graphenmodellen zu realen Aufgabenstellungen:

Verständnis für Optimierungsprobleme gewinnen.

Motivation zur Beschäftigung mit außermathematischen Problemen

Allgemein:Anschauliches Lösen von Aufgaben des Operations Research.Themen aus diesem Bereich wurden bisher im Unterricht der Schule fast überhaupt nicht behandelt und werden durch das Programm Knotengraph einer motivierenden Lösung zugänglich.

Die genannten Themenbereiche einschließlich der schon vorher erörterten Probleme (Baum-Pfadprobleme und verwandte Aufgaben, Eulerlinie, Hamiltonkreis, Färbungen eines Graphen und Planarität) (C II bis C V) decken weite Teile des in Hochschullehrbüchern der Graphentheorie dargestellten Problemkreise ab.Daher ist das (fertige) Programm Knotengraph ein universelles Werkzeug,um Problem der Graphentheorie im Schulunterricht anschaulich und motivierend zu behandeln

Nachfolgend wird noch eine in der Jahrgangsstufe 13 gestellte Aufgabe zu den Themen endlicher Automat,Markovkette und Graph reduzieren vorgestellt,die auch geeignet ist als schriftliche oder (auch in Teilen) als mündliche Abituraufgabe im Fach Informatik zum Gebiet Spracherkennung (Theoretische Informatik) eingesetzt zu werden.Zu Ihrer Lösung wird nur die Konzeption DWK,nicht jedoch Kenntnisse gemäß der Konzeptionen EWK oder CAK vorausgesetzt.

(Die Aufgabe wurden vom Verfasser dieser Arbeit als mündliche Abituraufgabe in den Teilen b und c gestellt.)

Aufgabe:

Ein Würfelspiel hat folgende Gewinnregeln:Es wird mit einem Würfel solange gewürfelt bis Gewinn oder Verlust eintritt. Verlust tritt dann ein,sobald eine der Zahlen 1, 2 oder 3 gewürfelt werden.Wenn eine der Zahlen 4, 5 oder 6 fallen,darf weiter gewürfelt werden.Das Spiel ist ebenfalls verloren,wenn unmittelbar nach der Zahl 4 die Zahl 5 geworfen wird.Im übrigen ist das Spiel gewonnen,wenn die Summe von zwei direkt aufeinanderfolgenden Würfen größer als 9 ist.

a)Definieren Sie den Begriff endlicher,determinierter Automat.Welche Bedeutung haben Eingabealphabet und Übergangsfunktion?Was versteht man unter einem reduzierten Automaten?Wie ist die von einem Automaten erkannte Sprache definiert?

b)Geben Sie jeweils zum oben beschriebenen Würfelspiel drei verschiedene Zahlenfolgen an,die einerseits zum Gewinn und anderseits zum Verlust führen.Das (obige) Würfelspiel soll durch einen Automaten dargestellt werden.Beschreiben Sie die von dem Automaten erkannte Sprache durch Angabe von Beispielen.

Stellen Sie den Automaten durch einen Graphen dar,und kennzeichnen Sie dabei Anfangs-und Endzustände in bekannter Weise.

c)Entwerfen Sie zu dem Automaten einen Graphen,dessen Kantenbewertung die Übergangswahrscheinlichkeiten zwischen den verschiedenen Zuständen des Automaten bedeutet als absorbierende Markovkette,und bestimmen Sie nach dem Algorithmus **Reduktion eines Signalflussgraphen/einer Markovkette** durch fortgesetztes Löschen der inneren Knoten einen reduzierten Graphen,der nur noch die Knoten des Anfangs-und der Endzustände enthält.Bestimmen Sie an Hand dieses neuen Graphen die Wahrscheinlichkeit bei diesem Spiel zu gewinnen bzw. zu verlieren.
(Zeichnen Sie zur Dokumentation des Algorithmus jeweils den nach Löschen eines Knoten neu entstehenden Graphen mit seiner Kantenbewertung auf.)

d)Bestimmen Sie zu dem Automaten den **reduzierten Automaten** (bezüglich der vom Automaten erkannten Sprache) mit 4 Zuständen, und zeichnen Sie auch zu diesem Automaten einen Graphen.Geben Sie zu diesem reduzierten Automaten eine linksreguläre Grammatik an,durch die sich die Wörter der Sprache erzeugen lassen.

Die Lösungen dieser Würfelspielaufgabe befinden sich im Anhang.

Die Aufgabenteile c) und d) gehören zum Anforderungsbereich Transfer und erfordern Anwendung von im Unterricht besprochenen Verfahren.Das Entwerfen des Graphen selbst erfordert,da schon an ähnlichen Beispielen geübt,ebenfalls Transfer und im kleinem Umfang eigenständiges Denken,während Aufgabe a) lediglich Wissen abfragt.(Die Lösungen von Teil c und d können mit dem Programm Knotengraph simuliert werden.)

8.Erfahrungen aus den duchgeführten Unterrichtsreihen und die Zusammenstellung von Unterrichtssequenzen

Im Anhang F IV befinden sich jeweils detailierte Unterrichtspläne mit didaktisch-methodischen Erläuterungen zu möglichen Einstiegstunden der in diesem Kapitel B III vorgestellten verschiedenen Konzeptionen DWK,EWK oder CAK sowie als deren Voraussetzungen zu den Einstiegsprojekten Einführung in das Programmieren mit Objekten und Aufbau einer Objektliste.

Diese Arbeit stellt eine Vielzahl von verschiedenen Unterrichtssequenzen zum Thema Graphentheorie vor,die sich einer der genannten Konzeptionen zuordnen lassen.Aus der Vielzahl der möglichen Unterrichtssequenzen muss zur Gestaltung einer Unterrichtsreihe eine geeignete Auswahl getroffen werden.

Bei einem Einstieg gemäß den Konzeptionen EWK und CAK her,ist es günstig zunächst die unter Abschnitt 2) und 3) dieses Kapitels beschrieben Einstiegsprojekte durchführen.Wie die Unterrichtserfahrung gezeigt hat,ist es nicht unbedingt nötig beide Projekte komplett,wie oben beschrieben durchzuführen,falls dazu die Zeit fehlen sollte.

Bei dem Graphik-Projekt kann man bei Zeitknappheit nur die Objektstruktur bis zum Kantentyp entwickeln,d.h. lediglich die Typen TFigur,TPunkt,TKnoten,TInhaltsknoten und TKante behandeln,um schon alle wichtigen Aspekte objektorientierten Programmierens und die unbedingt erforderlichen Grundkenntnisse für die spätere Entwicklung oder Anwendung der Graphenstruktur vermitteln zu können.

Bezüglich der Objekt-Liste wurde in Abschnitt 3) schon erwähnt,dass man sich auf die Quellcodeerstellung von beispielhaften Methoden wie Anfügen und Löschen von Elementen am Anfang und Ende der Liste sowie die Methoden ErstesElement und LetztesElement sowie Leer und Anzahl beschränken kann.Wichtig ist es,bei diesem Projekt die Datenstruktur und das Prinzip des objektorientierten Hinzufügen von Elementen mit konkretem,vom Anwender frei wählbarem Datentyp zu der Liste gründlich zu besprechen.

Danach kann die Besprechung der Datenstrukturen Keller und Schlange als Anwendung der vorliegenden Listenstruktur eventuell entfallen,und man hat stattdessen die Möglichkeit eine der Konzeption CAK oder EWK sofort weiter zu verfolgen.

Möchte man das Hauptgewicht auf das Programmieren von mathematischen Graphenalgorithmen (EWK) und nicht so sehr auf das Erstellen der Datenstruktur des Graphen (CAK) legen,hat es sich im Unterricht trotzdem als nützlich erwiesen zum Kennenlernen der prinzipiellen Datenstruktur des Graphen der graphikorientierten Entwicklungsumgebung die textorientierte und deshalb einfachere Version von Knotengraph (CAK),d.h. die entsprechenden Units den Schülern aber ohne das textorientierte Hauptprogramm zunächst fertig vorzugeben.

Durch das Erstellen und Experimentieren mit eigenen Programmen,die mittels der zur Verfügung stehenden Methoden der vorgegebenen Objekte,Graphen auf-und abbauen sowie verändern,können Schülern die nötige Erfahrung im Um-

gang mit den neuen Objektmethoden sammeln,die in gleicher oder erweiterter Form in der Entwicklungsumgebung EWK vorhanden sind.Auf diese Weise gestaltet sich der Übergang zur graphikorientierten Programmsystem bedeutend einfacher.

Vom Schwierigkeitsgrad her und wegen des relativ kurzen Quellcodeumfangs bietet sich als einfachste Unterrichtsreihe die Behandlung von Pfadproblemen an,beginnend mit dem Tiefen Baumdurchlauf,dessen Algorithmus sich in vielfältiger Form zu den verwandten Algorithmen Kreise,Anzahl Zielknoten,Alle Pfade,Minimale Pfade abändern lässt,bis zum Abstand von zwei Knoten,alle Pfade zwischen zwei Knoten und dem minimalen Gerüst eines Graphen.Zusätzlich in diese Reihe paßt die Behandlung der Algorithmen geschlossene und offene Eulerlinie,Hamiltonkreis.

Der Umfang einer solchen Unterrichtsreihe kann durch Auswahl aus dem eben genannten Katalogs vom Lehrer beliebig zusammengestellt werden.Es kann auch nur ein einziger Algorithmus exemplarisch behandelt werden und die übrigen werden an Hand der Fertigversion von Knotengraph (DWK) mit Hilfe des Demo-Modus veranschaulicht.

Die übrigen dem Menü Anwendungen zugeordneten Algorithmen haben sich vom Umfang und von der Schwierigkeit her als komplexer erwiesen,so dass es sich als sinnvoll erwiesen hat,in einer Unterrichtsreihe nur eines oder bestenfalls zwei miteinander zusammenhängende Graphenprobleme dieser Art als Programmierungsprojekt zu behandeln.

Für den Grundkursbereich geeignet,sind hier besonders die Anwendungen Netzwerkzeitplan,Färbbarkeit,Endlicher Automat,Graph als Relation,Maximaler Netzfluss und Maximales Matching sowie Gleichungssystem.(Bis auf den letzten Algorithmus befinden sich alle Objekte und die entsprechenden Methoden in der Unit UMath1 des Quellcodelistings der Fertigversion (DWK) von Knotengraph.Der Quellcode zu Gleichungssystem ist in der Unit UMath2.)

Die übrigen Anwendungen absorbierende und stationäre Markovketten,Graph reduzieren sowie Minimale Kosten,Transportproblem,Optimales Matching und Chinesischer Briefträger sind vom Umfang her größer,so dass sich die Programmierung der entsprechenden Algorithmen meistens nur in einem Leistungskurs anbietet,oder aber stattdessen die Fertigversion von Knotengraph benutzt werden sollte (Konzeption DWK),um diese Algorithmen mittels des Demo-Modus zu veranschaulichen und ohne Quellcodeerzeugung im Grundkurs zu besprechen.

Die im letzten Absatz genannten Algorithmen können nämlich als Fertigversion eine gute Ergänzung zu den im Grundkurs durchführbaren im vorletzten Absatz genannten zu programmierenden Graphenanwendungen darstellen,z.B. ist der Algorithmus Chinesischer Briefträger die konsequente Erweiterung des Eulerlinienproblems oder aber das Maximalflussproblem wird durch den Algorithmen Minimale Kosten und Transportproblem und das Maximale Matchingsproblem durch den Algorithmus Optimales Matching sinnvoll ausgeweitet und ergänzt.Der Algorithmus Graph reduzieren baut schließlich auf dem Algorithmus Gleichungssystem auf,und Signalflussprobleme,können mit ihm bequemer als in der Gleichungssystemform gelöst werden.

Will man dagegen das Hauptgewicht auf die Entwicklung der objektorientierten Datenstruktur des Graphen legen (Konzeption CAK),wird man den unter Punkt 4 beschriebenen Aufbau der textorientierten Version von Knotengraph verfolgen.Auch hier ist es nicht unbedingt notwendig die kompletten Units von den Schülern erstellen zu lassen,sondern man kann auch hier wieder exemplarisch vorgehen.

Möglichkeiten dazu für die Unit UGraphC wurden weiter oben (Abschnitt 4) schon genannt.In der Unit UInhgrphC kann man sich auf die Erstellung folgender Methoden durch die Schüler als beispielhafte Methoden beschränken:

function Graphknoten,procedure Knotenerzeugenundeinfuegen,procedure FuegeKanteein,procedure Kanteerzeugenundeinfuegen,procedure ZeigeGraph

1. Aus dem Kapitel 5 sollten danach wahlweise ein,maximal aber zwei mathematische Graphenalgorithmen ausgesucht werden,um die Funktionsweise der Struktur des Graphen demonstrieren zu können.

Die neu entwickelte Datenstruktur der Objekt-Graphen der Konzeptionen CAK,EWK und DWK stand von vorneherein nicht in der hier vorgestellten Form fest,sondern ergab sich erst nach und nach als Resultat aus den Unterrichtserfahrungen.Ausführliche Erläuterungen dazu finden sich im Anhang F I.

Ebenfalls im Anhang (F V) findet sich die Auswertung von Fragen zur Unterrichtsreihe an die Schüler.Als Resultat zeigt sich,dass die Beschäftigung mit algorithmischer Graphentheorie unter Verwendung von objektorientierten Datenstrukturen von den Schülern durchweg positiv beurteilt wird:

Herausgestellt wird die andere Art Mathematik zu betreiben,nämlich größtenteils ohne Formeln und stattdessen mit Hilfe von Algorithmen,die interessanter und anschaulicher ist,weil sie mehr selbstständiges eigenes Handeln ermöglicht.Durch den Demo-Modus können die Algorithmen visualisiert werden.Im „normalen“ Mathematikunterricht waren Graphen als Strukturierungsmittel oder gar Algorithmen auf Graphen den Schüler während der Schulzeit nicht begegnet.Die Beschäftigung mit Graphen hält die Mehrzahl der Schüler für interessanter als die Lerninhalte des konventionellen Mathematikunterrichts.

Mittels Graphen können nach Schülerantworten die mathematischen Beziehungen zwischen Objekten besser abgebildet werden.Auch im Mathematikunterricht sollten Graphen eingesetzt werden.Ein Vorteil wäre,dass man die (mathematischen) Eigenschaften besser versteht und behält.Ein Teil der Schüler plädiert allerdings auch dafür,Graphen hauptsächlich im Informatikunterricht einzusetzen,weil sich dort mehr Zeit und Möglichkeiten für die Programmierung bietet.

Graphen werden auf Grund ihrer Struktur als Hilfsmittel gesehen,um Probleme besser sichtbar zu machen und auf Grund dessen schneller zu einer Lösung zu kommen.Das objektorientierte Programmieren wird von den meisten Schülern nicht schwerer empfunden als ein imperativer Programmierstil.Die meisten Schüler sind der Meinung,auch nach Abschluss der Unterrichtsreihe selbstständig neue unbekannte Graphenprobleme mit dem erworbenen Rüstzeug lösen zu können.Durch die Kopplung an geeignete Objekte werden die Graphenalgorithmen als leichter empfunden.

Die objektorientierte Vorgabe der graphischen Oberfläche wird mehrheitlich als sehr vorteilhaft eingestuft,allerdings plädiert eine große Zahl von Schülern (ca. 50%) auch dafür zunächst mit einer textorientierten Version des Programms Knotengraph zu starten,um die Struktur des Graphen von Grund auf besser verstehen zu können.

Die Entwicklung der Graphenstruktur aus der Listenstruktur wird von den meisten Schüler als leicht oder mittelschwer eingeschätzt,wenn gleich der Umgang mit Listen,wie aus anderen Antworten auf Fragen hervorgeht,das Thema ist,bei dem Schüler am ehesten Probleme hatten.Dagegen scheint die Programmierung von Algorithmen wie der Tiefensuche und davon abgeleitete Algorithmen,das Suchen von Euler-und Hamiltonlinien und allgemein das Backtrackingverfahren von keinem besonderen Schwierigkeitsgrad gewesen zu sein.Die meisten Schüler trauen sich zu,diese Verfahren auch noch nach Abschluss der Unterrichtsreihe zu beherrschen und sind der Meinung,dass allgemein die in den Unterrichtsreihen vermittelten Kenntnisse auch noch nach dem Abitur für sie von Bedeutung und wichtig sind.

9.Zusammenfassung:

Die Ausführungen dieses Kapitels zeigen,dass das Programmsystem Knotengraph methodisch-didaktisch unter drei Gesichtspunkten eingesetzt werden kann:

1)Als fertiges Programm zur Demonstration von Graphenalgorithmen mit Hilfe des Demo-Modus oder Einzelschrittmodus.Auf diese Weise können experimentell

Eigenschaften von Graphen erkannt und der Ablauf von Algorithmen nachvollzogen bzw. neu entdeckt werden.
(Geeignet zum Einsatz im Mathematikunterricht oder an Stellen im Informatikunterricht,in dem Algorithmen nur verbal ohne Programmierung von Quellcode entdeckt oder beschrieben werden sollen.)
Konzeption:DWK (siehe Einleitung)

2)Als objektorientierte Entwicklungsumgebung für die Erstellung von Algorithmen der Graphentheorie,die schon alle Eigenschaften zur Verwaltung von Graphen einschließlich einer Benutzeroberfläche zur Verfügung stellt,und in der nur noch der eigentlich neue gewünschte Algorithmus hinzugefügt werden muß.Da alle grundlegenden Methoden schon vorhanden sind,kann man sich ausschließlich auf die Umsetzung des mathematischen Verfahrens direkt in den entsprechenden Quellcode konzentrieren.

(Geeignet zum Einsatz im Mathematik- und Informatikunterricht bei Vorhandensein von entsprechenden Kenntnissen objektorientierten Programmierens)
Konzeption:EWK (siehe Einleitung)

3)Als (nachzuvollziehendes) Muster für den objektorientierten Aufbau eines Graphen aus einer Listenstruktur zur Durchführung eines Programmierungsprojektes,das es zum Schluss gestattet,ausgewählte wichtige mathematische Algorithmen zu programmieren und auszuführen.Hierbei empfiehlt es sich,auf eine graphische Darstellung des Graphen zu verzichten und textorientierte Ausgaben zu verwenden,um den Umfang des Programms nicht zu groß oder unübersichtlich werden zu lassen.

(Geeignet zum Einsatz im Informatikunterricht und zum Erlernen objektorientierten Programmierens)
Konzeption:CAK (siehe Einleitung)

Optimal ist es,eine Unterichtsreihe zu konzipieren,die wie im vorigen beschrieben alle drei Aspekte in der Reihenfolge 3,2 und 1 behandelt,da dadurch ein vollkommener Einblick in die Verfahrensweisen der algorithmischen Graphentheorie erreicht werden kann.

C Skizzen von Unterrichtsreihen

In diesem dritten Teil C der vorliegenden Arbeit sollen Unterrichtsreihen zu verschiedenen Themen der Graphentheorie skizziert werden,wie sie entweder mit Hilfe des fertigen Programms Knotengraphs oder aber dessen Entwicklungsumgebung durchgeführt werden können (Konzeptionen DWK oder EWK,siehe Einleitung).

Die erste Option wird man wählen,wenn es darum geht eine Unterrichtsreihe zu ausgewählten Themen der Graphentheorie ohne Benutzung einer Programmiersprache unter umgangssprachlicher Beschreibung der Algorithmen z.B. im Mathematikunterricht zu konzipieren und die zweite Möglichkeit bei der Umsetzung eines geeigneten objektorientierten Programmierungsprojekt im Informatikunterricht (oder im Mathematikunterichts bei Vorhandensein entsprechender Programmierkenntnisse der Schüler),bei dem auch der Aspekt der Entwicklung der entsprechenden Datenstrukturen sowie deren Codierung im Vordergrund steht.

Jede Unterrichtsskizze besteht dabei aus ein oder mehreren Einführungsaufaben,an Hand derer das Lösungsverfahren von den Schülern problemorientiert erarbeitet werden kann,einer Verbalbeschreibung des Lösungsalgorithmus sowie dessen Umsetzung und Codierung in eine objektorientierte Datenstruktur mittels der Programmiersprache Delphi aufbauend auf den vorgegebenen Möglichkeiten der Entwicklungsumgebung von Knotengraph (EWK),sowie einer Reihe von Anwendungsaufgaben jeweils mit Lösungen,die der Übung,Demonstration und Vertiefung dienen sollen.Zur Begründung des Lösungsverfahrens werden,wenn nötig,die entsprechenden mathematischen Sätze angeführt,zu denen (meistens) schulgerechte Beweisskizzen angegeben werden.

Mit Hilfe der Einführungsaufgabe(n) werden zunächst Kern und Struktur der graphentheoretischen Aufgabenstellung sichtbar gemacht,wobei oft von Problemstellungen der realen Erfahrungswelt ausgegangen wird und mittels eines Modellierungsprozesses die wesentlichen Beziehungen und Zusammenhänge auf ein Graphenmodell abgebildet und reduziert werden.

Daran kann sich dann die Definition geeigneter Begriffe anschließen.

Entscheidet man sich für den Einsatz des Programm Knotengraphs als fertiges Werkzeug DWK z.B. im Mathematikunterricht,wird man den Demomodus benutzen,um den Ablauf des Lösungsalgorithmus zu veranschaulichen bzw. um den Verbalalgorithmus zu erarbeiten.

Beim Einsatz der Entwicklungsumgebung EWK als Ausgangspunkt für ein Programmierungsprojekt verfolgt man dagegen eine andere Intention,nämlich die Umsetzung der Zusammenhänge des Graphenmodells in eine objektorientierte Struktur sowie die Erarbeitung und Anwendung allgemeiner Lösungsverfahren wie z.B. des rekursiven Backtrackingprinzips.Dann wird man den Demomodus erst nach der Erstellung des fertigen Programmcodes benutzen,um den prognostizierten Ablauf des algorithmischen Verfahrens zu bestätigen.In diesem Fall wird man zunächst versuchen die Einführungsaufgabe manuell,d.h. mittels eines auf Papier oder an die Tafel gezeichneten Graphen zu lösen und dabei ein allgemeines Verfahren sowie geeignete Datenstrukturen zu entwikkeln.

Eine Möglichkeit der objektorientierten Codierung (,wie sie auch jeweils in dem fertigen Programm Knotengraph (DWK) realisiert ist,) zur Orientierung bzw. als Vorgabebeispiel für ein neu zu beginnendes Programmierungsprojekt, schließt sich an die Verbalbeschreibung des grundlegenden Algorithmus in den einzelnen Kapitel an.Man kann es in dieser Weise übernehmen oder aber natürlich auch nach eigenen Wünschen entsprechend modifizieren.

Dabei wird in den Kapitel C II bis C XII jeweils fast der komplette Quellcode des Projekt wiedergegeben,wobei die einzelnen Methoden um den Kern des Algorithmus besser herauszustellen,lediglich um die

Anweisungen,die den Demomodus und Abbruchmodus betreffen,gekürzt worden sind.

(Der wirklich vollständige Quellcode kann dem Quelltextlisting im Anhang entnommen werden.)

Das Kapitel C I enthält keinen Quellcode und die entsprechenden Aufgaben können schon mit Hilfe der Entwicklungsumgebung des Programms Knotengraph (insbesondere dem Menü Eigenschaften) gelöst werden.Die dort dargestellten Aufgaben dienen zur Einführung in die Begriffe der Graphentheorie sowie zur Einübung der Bedienung der Entwicklungsumgebung.

Der Quellcode der Projektthemen der Kapitel C XII (Markovketten) als auch C XIII/C IV (Minimale Kosten, Transportproblem, Optimales Matching, Chinesischer Briefträger) ist jeweils relativ umfangreich,so dass die Erstellung eines solchen Programmierungsprojekts nur dann möglich ist,wenn genug Zeit,z.B. in einem Informatikleistungskurs zur Verfügung steht. Ansonsten bietet sich bei der Thematik dieser Kapitel die Anwendung des Programms Knotengraph als fertiges Werkzeug im Mathematikunterricht bei der Wahrscheinlichkeitsrechnung (Kapitel C XII) oder aber als Fortführung und Abrundung entsprechender Programmierungsprojekte der Kapitel C IX (Maximaler Netzfluß) und C X (Maximales Matching) im Informatik- oder Mathematikunterricht zum Thema Operations Research an (DWK).

Der Quellcode ist hier nur überblicksmäßig angegeben,aber natürlich wieder komplett im Quelltextlisting des Anhangs (als Quelltext des fertigen Programms Knotengraph) enthalten.

Das im Kapitel C XI dargestellte Verfahren nach Mason zur Lösung eines linearen Gleichungssystem mit Hilfe von Graphen ist die Grundlage für die Methode der Reduktion eines Markov-oder Signalflußgraphen in Kapitel C XII. Wegen der Anschaulichkeit und der Bedeutung des Themas für den Mathematikunterricht (lineare Algebra) ist es aber auch sinnvoll,sich nur auf die Behandlung des Mason-Verfahrens zu beschränken,ohne auf Markov- oder Signalflußgraphen einzugehen.

Kapitel C X kann als Fortsetzung von Kapitel C IX aufgefaßt werden,so dass eine Unterrichtsreihe zum Thema Maximaler Netzfluß auch das Thema Maximales Matching behandeln oder aber zumindest einen Ausblick auf die Ermittlung eines maximalen Matching im Spezialfall bipartiter Graph geben sollte.

Das Thema des Kapitels C VII (Endlicher Automat) kann eine Unterichtsreihe theoretische Informatik zum Thema Automatentheorie,Spracherkennung und Grammatik von Sprachen sowie Berechenbarkeit ergänzen und veranschaulichen.

Die Kapitel C II und C III behandeln Pfade in Graphen und werden durch die Inhalte des Kapitel C V Euler- und Hamiltonlinien ergänzt.Die dargestellten Unterrichtsreihen sind aber jeweils auch unabhängig voneinander im Unterricht einzusetzen.

Die Kapitel C IV (Eulerbeziehung,ebene Graphen und Färbbarkeit) sowie C VIII (Graphen als Relationen) und C VI (Graph als Netzplan) beinhalten jeweils die Erörterung und Lösung interessanter Einzelthemen der Graphentheorie und können unabhängig von den anderen Kapiteln behandelt werden.Das Erstellen von Netzplänen gehört auch wiederum zum Gebiet Operations Research.

Zusammenfassend gesagt,eignen sich gerade die in den Kapitel C II bis C XI dargestellten Graphenprobleme besonders als Aufgabenstellung eines Programmierungsprojekts (z.B. in Grundkursen Informatik).

Bezüglich der Schwierigkeit und des Programmumfangs relativ einfache Projekte sind insbesondere die Programmierung von Pfadproblemen der Kapitel C II,C III sowie C V sowie das Färbbarkeitsproblem in Kapitel C IV,die auch von grundsätzlicher didaktischer Bedeutung sind,da sie einerseits die Behandlung der schon im herkömmmlichen Informatikunterricht enthaltenden Baumalgorithmen ersetzen und/oder ausweiten sowie unter einem neuen

Aspekt darstellen können,als auch gestatten,so wichtige Algorithmen wie den Tiefendurchlauf und das Backtrackingverfahren an Hand immer neuer ähnlicher Beispiele einzuüben.

Die Graphenprobleme der Kapitel C XII bis C XIV sind,wie schon erwähnt, eher als Ergänzung der Programmierungsprojekte der vorigen Kapitel unter Einsatz des (fertigen) Programms Knotengraphs (und nur in besonderen Fällen als Programmierungsprojekt s.o.) gedacht.

Außerdem eignen sich in dieser Weise (insbesondere unter Verwendung des Demomodus) alle Unterichtsreihen der Kapitel C I bis C XIV als Thema für einen Mathematikunterricht,der ausgewählte Probleme der Graphentheorie behandelt.

Detailierte Unterrichtspläne zu Einführungsstunden in verschiedene Unterrichtssequenzen der Konzeptionen DWK und EWK mit didaktisch-methodischen Bemerkungen finden sich im Anhang F IV.

Die Beispielgraphen zu den einzelnen Aufgaben sind auf der Installations-CD vorhanden,werden beim Setup unter der Bezeichnung G001.gra bis G100.gra in ein Verzeichnis (Beisp) der Festplatte installiert und brauchen nicht selber erstellt zu werden.Auf diese Weise läßt sich die Lösung aller im folgenden aufgeführten Anwendungsaufgaben unmittelbar nachvollziehen.

C I Knoten,Kanten,Bäume,Wege und Komponenten

Dieses Kapitel ist stellt eine Einführung in die grundlegenden (als Kapitelüberschrift genannten) Begriffe der Graphentheorie dar und erläutert ihre Bedeutung an Hand von Beispielen und Aufgaben.Die Aufgaben sollten dabei teilweise per Hand aber größtenteils auch mit Hilfe des Programms Knotengraph gelöst werden.Dabei ist stets (im Gegensatz zu den nachfolgenden Kapiteln) das (fertige) Programm Knotengraph als **Demonstrations-Werkzeug** einzusetzen, und es sind keine Programmierungsaufgaben zu lösen.Die Aufgaben können deshalb außer zur Verdeutlichung der grundlegenden Begriffe der Graphentheorie auch dazu dienen,den Umgang mit der Benutzeroberfläche Knotengraph zu üben.Die Aufgaben brauchen nicht in der hier dargestellten Reihenfolge behandelt zu werden,sondern man kann auch nur eine einzelnen Aufgabe herausgreifen,um sie zwecks Veranschaulichung und Klärung eines oder mehrerer Begriffe an den Beginn eines Programmierungsprojekts zu stellen.Außerdem können die dargestellten Aufgaben gut zur Einführung in die Graphentheorie in einem rein mathematisch orientierten Kurs, ohne die Algorithmen programmieren zu wollen,eingesetzt werden.

C Definition I.1:

Ein Graph G ist ein Tripel (K,A,R) mit einer nicht leeren Menge von Knoten K,einer Menge von Kanten A und einer Relation $R \subseteq AxKxK$,wobei jedes Element aus A in der Relation R vorkommt.

Wenn $(a,k_1,k_2) \in R$ gilt,bezeichnet man k_1 und k_2 als die Knoten der Kante a.Die Kante a ist dann inzident mit den Knoten k_1 und k_2. Wenn $k_1=k_2$ gilt,heißt a Schlinge.

Einer Kante a kann eine Durchlaufrichtung zugeordnet werde.Sie wird dann als gerichtete Kante bezeichnet,sonst als ungerichtete Kante.Durch die Durchlaufrichtung wird einer der beiden Knoten k_i zum Ausgangsknoten und der andere zum Zielknoten.

Ein Graph G heißt ungerichtet,wenn seine Kanten nur aus ungerichteten Kanten bestehen,und gerichtet,wenn seine Kanten aus gerichteten Kanten bestehen.

Zwei Kanten a_1 und a_2 mit $(a_1,k_1,k_2) \in R$ und $(a_2,k_1,k_2) \in R$ heißen Parallelkanten.

Ein Graph heißt schlicht,wenn er keine Parallelkanten oder Schlingen als Kanten enthält.

Ein Knoten heißt isoliert,wenn er keine mit ihm inzidenten Kanten besitzt.

Unter dem Grad eines Knoten versteht man bei einem ungerichteten Graph die Anzahl der mit dem Knoten inzidenten Kanten und bei einem gerichteten Graph die Anzahl der in der Richtung von dem Knoten auslaufenden gerichteten inzidenten Kanten vermindert um die Anzahl der in den Knoten einlaufenden gerichteten inzidenten Kanten.

Ein Knoten heißt gerade bzw. ungerade,wenn der Knotengrad des Knotens gerade oder ungerade ist.

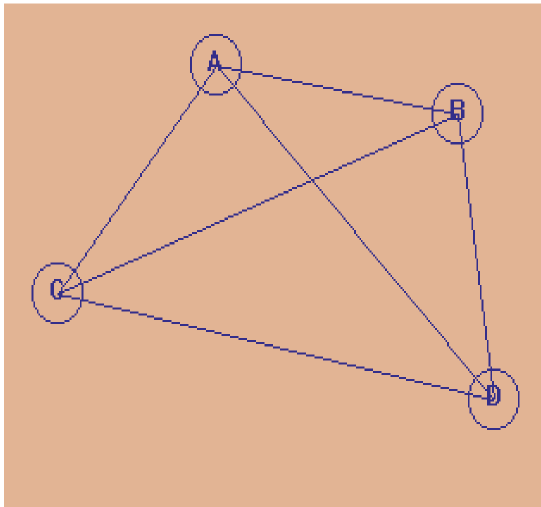
Ein schlichter Graph heißt vollständig,wenn mit je zwei verschiedenen Knoten jeweils genau eine Verbindungskante inzident ist.

(Normalerweise wird im Unterricht zunächst keine formale Definition eines Graphen benötigt,da die Bedeutung von Knoten,Kanten usw. schon anschaulich sofort klar ist.Eine Präzisierung der Begriffe sollte erst bei der Durchführung eines darauf aufbauenden Beweises erfolgen)

C Aufgabe I.1 (Einstiegsproblem):

Erzeuge beliebige vollständige Graphen (z.B: mit 2,3,4,5 Knoten).Ermittle mit Hilfe des Menüs Eigenschaften/Anzahl Kanten und Knoten des Programms

Knotengraph jeweils die Kanten- und Knotenzahl und jeweils die Summe der Grade der Knoten.



G001.gra

Vollständiger Graph g mit Knotenzahl 4

Knotenzahl:	Kantenzahl:	Summe Knotengrad:
k(g)=k	a(g)=a	s(g)=s
2	1	2
3	3	6
4	6	12
5	10	20

Vermutete Formel: $a = \frac{k(k-1)}{2}$

Beweis:

Von jedem Knoten führen genau a-1 Kanten zu den übrigen Knoten des Graphen,weil der Graph schlicht und vollständig ist.

Dies ergibt k(k-1) Kanten,wobei jede Kante doppelt gezählt wird.

Aus der letzten Bemerkung ergibt sich für jeden Graphen:

C Satz I.1:

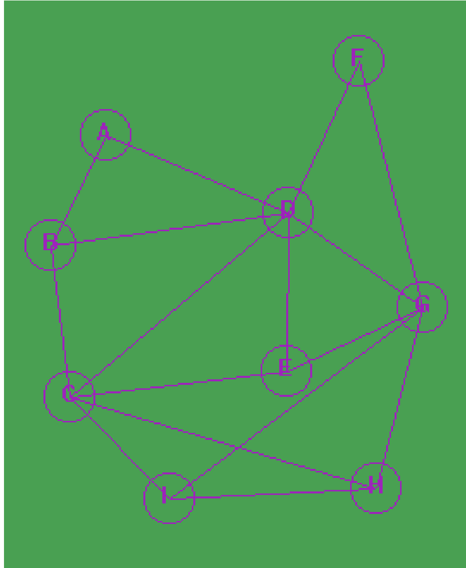
In jedem ungerichteten Graphen ist die Summe der Knotengrade gleich der doppelten Kantenzahl.

C Aufgabe I.2a:

Bei einem Treffen begrüßen sich diejenigen Personen per Handschlag,die sich gegenseitig kennen.

Die Personen werden durch Knoten,die Handschläge durch Kanten dargestellt.

Zeichne einige Begrüßungsgraphen und ermittle,wieviel Personen jeweils einer ungeraden Zahl von Personen die Hand reicht.



G002.gra

B,C,E,G,H und I begrüßen eine ungerade Personenzahl.Das sind 6 (gerade Zahl) Personen.

Lösung (Ausgabe bei Anwahl des Menüs Eigenschaften/Knoten zeigen):

A Koordinaten: X=207 Y=138 Knotengrad (gerichtet/ungerichtet): 0/2 Typ:

B Koordinaten: X=175 Y=202 Knotengrad (gerichtet/ungerichtet): 0/3 Typ:

C Koordinaten: X=186 Y=290 Knotengrad (gerichtet/ungerichtet): 0/5 Typ:

D Koordinaten: X=313 Y=183 Knotengrad (gerichtet/ungerichtet): 0/6 Typ:

E Koordinaten: X=312 Y=275 Knotengrad (gerichtet/ungerichtet): 0/3 Typ:

F Koordinaten: X=354 Y=95 Knotengrad (gerichtet/ungerichtet): 0/2 Typ:

G Koordinaten: X=391 Y=238 Knotengrad (gerichtet/ungerichtet): 0/5 Typ:

H Koordinaten: X=364 Y=343 Knotengrad (gerichtet/ungerichtet): 0/3 Typ:

I Koordinaten: X=244 Y=349 Knotengrad (gerichtet/ungerichtet): 0/3 Typ:

C Satz I.2:

In einem Graphen ist die Anzahl der ungeraden Knoten eine gerade Zahl.

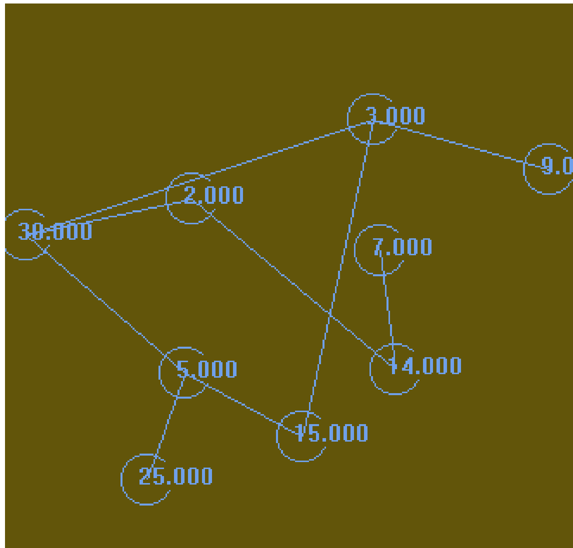
Beweis:

Die Summe der Knotengrade ist nach I.1 eine gerade Zahl.Nimmt man die geraden Knoten aus dieser Summe heraus,so ergibt sich,dass die Summe der Knotengrade der ungeraden Knoten eine gerade Zahl ist.Damit eine Summe von ungeraden Zahlen gerade ist,muß die Anzahl der Summanden gerade sein.

Weiteres Anwendungsbeispiel zu Satz I.2:

C Aufgabe I.2b:

Gegeben ist die Menge M={2;3;5;7;9;14;15;25;30}.Stelle die Relation „ist teilbar durch" bzw. „ist Vielfaches von" durch eine Kante zwischen den Zahlen als Knoten dar.Wieviele Zahlen haben eine ungerade Zahl von Teilern bzw. Vielfachen?



G003.gra

Teilbarkeits-/Vielfachengraph

Die Zahlen 3;5;7;9;25;30 (Anzahl 6 d.h. gerade Zahl) haben eine ungerade Zahl von Teilern.

<u>Lösung (Ausgabe bei Anwahl des Menüs Eigenschaften/Knoten zeigen):</u>

2 Koordinaten: X=200 Y=113 Knotengrad (gerichtet/ungerichtet): 0/2 Typ:

3 Koordinaten: X=305 Y=67 Knotengrad (gerichtet/ungerichtet): 0/3 Typ:

5 Koordinaten: X=196 Y=214 Knotengrad (gerichtet/ungerichtet): 0/3 Typ:

7 Koordinaten: X=309 Y=143 Knotengrad (gerichtet/ungerichtet): 0/1 Typ:

14 Koordinaten: X=318 Y=212 Knotengrad (gerichtet/ungerichtet): 0/2 Typ:

15 Koordinaten: X=264 Y=251 Knotengrad (gerichtet/ungerichtet): 0/2 Typ:

9 Koordinaten: X=407 Y=96 Knotengrad (gerichtet/ungerichtet): 0/1 Typ:

25 Koordinaten: X=174 Y=276 Knotengrad (gerichtet/ungerichtet): 0/1 Typ:

30 Koordinaten: X=104 Y=134 Knotengrad (gerichtet/ungerichtet): 0/3 Typ:

<u>C Definition I.2:</u>

Es seien x und y zwei Knoten des Graphen G.Außerdem seien n Kanten a_i mit $1 \le i \le n$ und n Knoten k_i mit $1 \le i \le n$ vorgegeben. Die Kante a_i sei jeweils inzident mit k_{i-1} und k_i für $2 \le i \le n-1$.Die Kante a_1 sei inzident mit x und k_1.Die Kante a_n sei inzident mit k_n und y.

Dann heißt $(a_1,a_2..,a_n)$ ein x mit y verbindender Kantenzug oder Walk.Der Walk heißt Weg bzw. Pfad,wenn alle Knoten der Mengen $\{x;k_1;..;k_n\}$ und $\{k_1;..;k_n;y\}$ jeweils paarweise verschieden sind.(Bez. auch: (x,k1..,kn,y))

Der Walk oder Weg ist gerichtet bzw. ungerichtet,je nachdem die Kanten gerichtet bzw. ungerichtet sind.Ein Weg heißt Kreis,wenn x=y ist.

Falls gerichtet,kann der Walk oder Weg von x nach y durchlaufen werden.

Ein Graph heißt zusammenhängend,wenn zwischen je zwei beliebigen Knoten des Graphen ein Kantenzug existiert.

Ein Teilgraph eines Graphen $G=(K,A,R)$ ist ein Graph $G'=(K',A',R')$ mit $K'\subseteq K;A'\subseteq A;R'\subseteq R$.

Ein maximaler Teilgraph von G bezüglich der Anzahl der Knotenmenge K',mit der Eigenschaft,dass zwischen jedem Knoten in K' zu jedem anderen Knoten ein diese Knoten verbindender Kantenzug existiert,heißt Komponente von G.

Eine Kante heißt Brücke,wenn beim Löschen der Kante aus dem Graphen,der Graph in eine größere Komponentenzahl zerfällt.

C Aufgabe I.2c:

Ermittle jeweils alle Brücken im Graph der Aufgabe C I.2b G003.gra.Hat der Graph Kreise?Wenn ja mit welcher kleinsten und größten Knotenzahl?

Lösung Brücken (Ausgabe bei Anwahl des Menüs Eigenschaften/Anzahl Brücken):

2 30

2 14

3 9

5 25

7 14

Gesamt: 5 Brücken

Lösung Kreise (Ausgabe bei Anwahl des Menüs Eigenschaften/Kreise sowie Pfade/Alle Kreise bei Wahl des Startknoten 3):

Ungerichteter Graph:

Kreis mit kleinster Kantenzahl(>2):

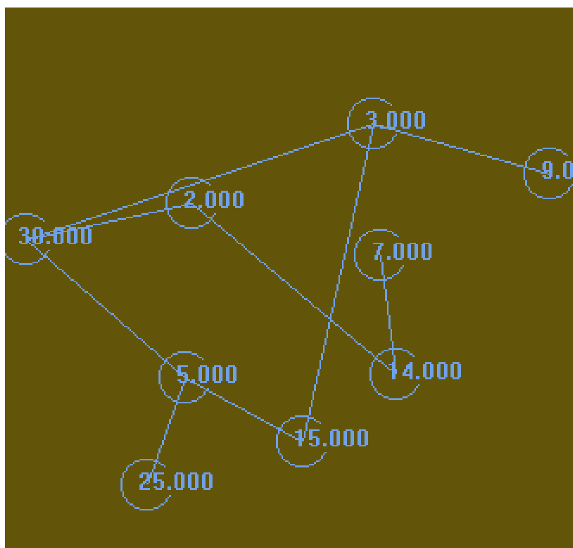
Kantenzahl: 4

Kreis mit größter Kantenzahl(>2):

Kantenzahl: 4

3 15 5 30 3 Summe: 4.000 Produkt: 1.000

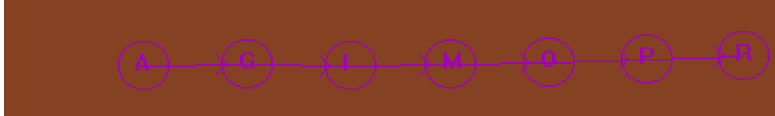
3 30 5 15 3 Summe: 4.000 Produkt: 1.000



G003.gra

C Aufgabe I.3 (Einstiegsproblem zu Bäumen,Gerüsten,Durchlaufverfahren):

Gegeben ist die Menge M={A;G;I;M;O;P;R},deren Elemente (Buchstaben) jeweils für Namen stehen sollen.Die Namen sollen auf Karteikarten notiert sein,die geordnet hintereinander stehen.Die Ordnung kann durch folgenden Graph (geordnete lineare Liste) dargestellt werden:



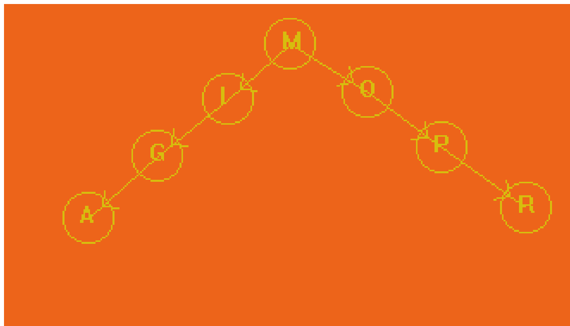
Um zu entscheiden,ob ein bestimmtes Listenelement in der Liste enthalten ist,wird die Liste von links nach rechts durchsucht.

Bei jedem Element wird dabei ein Vergleich durchgeführt.

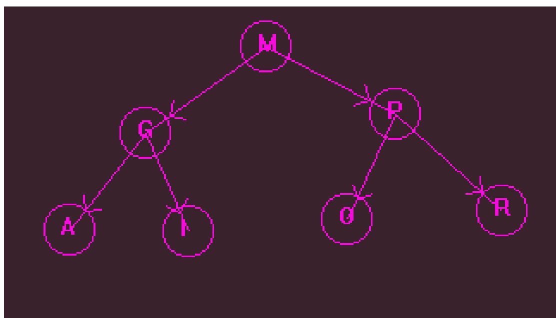
a)Wieviele Vergleiche sind durchschnittlich zum Finden eines bestimmten Elements notwendig,wenn die Suche beim Knoten A begonnen wird.?

Die Anordnung wird verändert und durch die folgenden Graphen dargestellt.Wieviel durchschnittliche Vergleiche sind jetzt notwendig,wenn jeweils die Suche beim Knoten M begonnen wird,und die Ordnung der Elemente im linken und rechten Teilgraphen durch eine entsprechende Abfrage berücksichtigt wird?

b)



c)



G0004a.gra Inorder

Lösung:

	Knoten:	Anzahl:	Anzahl Vergleiche:	Gesamtvergleiche:
a)	A	1	1	1
	G	1	2	2
	I	1	3	3
	M	1	4	4
	O	1	5	5
	P	1	6	6
	R	1	7	7
b)	M	1	1	1
	I,O	2	2	4
	G,P	2	3	6
	A,R	2	4	8
c)	M	1	1	1
	G,P	2	2	4
	A,I,O,R	4	3	12

Durchschnittliche Vergleichszahl:

a) (1+2+3+4+5+6+7):7 = 4
b) (1+4+6+8):7= 2,71
c) (1+4+12):7= 2,43

Ergebnis:

Der bei c) verwendete Graph (Suchbaum) hat den Vorteil,dass er die kleinste durchschnittliche Anzahl von Vergleichen zum Finden eines bestimmten Elements benötigt.

Zusatzaufgabe:

Problemorientierter Einstieg zu den Durchlaufordnungen in einem Baum:

d)Lade jeweils die Graphen G004a.gra bis G004c.gra in das Programm Knotengraph und überprüfe mittels der Menüs Pfade/AllePfade/Inorder (G0004a.gra) und Pfade/TieferBaum/Postorder (G004b.gra) oder /Preorder (G004c.gra) die Reihenfolge des Suchlaufs unter Einschaltung des Demo-Modus.(Diese Aufgabe kann zur problemorientierten Erarbeitung der Durchlaufordnungen durch Vergleich der Verfahren dienen.)

Beobachtung bzw. Anzeige im Ausgabefenster:

M G A Summe: 2.000 Produkt: 1.000
M G Summe: 1.000 Produkt: 1.000
M G I Summe: 2.000 Produkt: 1.000
M Summe: 0.000 Produkt: 0.000
M P O Summe: 2.000 Produkt: 1.000
M P Summe: 1.000 Produkt: 1.000
M P R Summe: 2.000 Produkt: 1.000

R I A Summe: 2.000 Produkt: 1.000
R I G Summe: 2.000 Produkt: 1.000
R I Summe: 1.000 Produkt: 1.000
R P M Summe: 2.000 Produkt: 1.000
R P O Summe: 2.000 Produkt: 1.000
R P Summe: 1.000 Produkt: 1.000
R Summe: 0.000 Produkt: 0.000

A Summe: 0.000 Produkt: 0.000
A G Summe: 1.000 Produkt: 1.000
A G I Summe: 2.000 Produkt: 1.000
A G M Summe: 2.000 Produkt: 1.000
A O Summe: 1.000 Produkt: 1.000
A O P Summe: 2.000 Produkt: 1.000
A O R Summe: 2.000 Produkt: 1.000

d)Lösungen und Ergebnisse der Beobachtungen:

Der Baum ist in Inorder-Suchreihenfolge aufgebaut,d.h. ein Suchvorgang,der bei der Wurzel M beginnt,besucht jeweils den linken Kinderknoten vor dem Elternknoten und danach den rechten Kinderknoten.

Dabei enthält jeweils der linke Teilbaum die kleineren Knotenwerte (Buchstaben),so dass ein entsprechender Suchalgorithmus gerade die alphabetische Reihenfolge liefert.

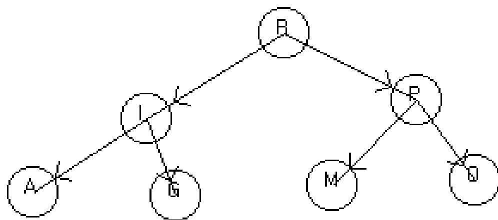
Ein Preorder-Suchlauf besucht gerade die Elternknoten vor den Kinderknoten und ein Postorder-Durchlauf die Kinderknoten vor den Elternknoten.

Die Inorder-Reihenfolge ist nur speziell bei Binärbäumen und nicht bei allgemeinen Graphenbäumen sinnvoll,da hier mehr als zwei Kinderknoten existieren können.

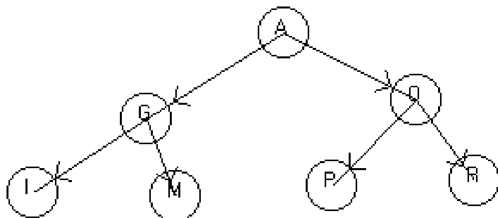
Dagegen behalten die Postorder-und Preorderfolge auch für Nicht-Binärbäume (also für Gerüste (s.u.) in allgemeinen Graphen) ihre Bedeutung.

Der Baum mit derselben alphabetischen Ausgabe wie oben in Postorder- und Preorder-Suchlauffolge:

(Das Ergebnis der obigen Berechnung der durchschnittlich notwendigen Vergleichszahl bleibt dabei unverändert.)



G004b.gra Postorder



G004c.gra Preorder

Die Einstiegsaufgabe gibt Anlass zu folgender Definition:

C Definition I.3:

Ein zusammenhängender Graph ohne Kreise heißt Baum.

Knoten mit dem Grad 1 heißen die Randknoten des Baums.

Ein Baum heißt gerichtet,wenn er ein gerichteter Graph ist.

Ein gerichteter Baum heißt Wurzelbaum,wenn gilt:

Es gibt genau einen Knoten,die Wurzel,von dem nur Kanten ausgehen.Alle anderen Knoten besitzen genau eine einlaufende Kante.Die Knoten,die keine ausgehenden Kanten besitzen,heißen Blätter (dies sind die Randknoten),die anderen Knoten heißen innere Knoten.Die Richtung der Kanten ist stets die Richtung von der Wurzel zum den Blättern.

Haben alle inneren Knoten des Baums und die Wurzel stets genau zwei ausgehende Kanten,heißt der Baum Binärbaum.

Sind die Knoteninhalte so geordnet,dass im linken Teilbaum (Teilgraph zu dem die linke ausgehende Kante führt) eines Binärbaums die Knoteninhalte kleiner bzw. kleiner-gleich (nach einer vorgegebenen Ordnung) als der Wurzelinhalt und der Wurzelinhalt wiederum kleiner ist als die Inhalte im rechten Teilbaum (Teilgraph zu dem die rechte ausgehende Kante führt) sind,so heißt der Binärbaum (gemäß Inorder) geordnet.

Sind jeweils bei die Knoteninhalte bei einem Wurzelbaum (nach einer vorgegebenen Ordnung) so geordnet,dass jeweils die Inhalte der Knoten eines Teilbaums kleiner/größer (bzw. kleiner-gleich/größer-gleich) als der Inhalt von dessen Wurzel ist,heißt der Baum nach Preorder/Postorder geordnet.

C Aufgabe I.4:

Erstelle mit dem Programm Knotengraph verschiedene Bäume (ungerichtete Bäume,Wurzelbäume usw.) und lasse jeweils die Knoten- und Kantenzahl anzeigen.

Ergebnis z.B. für die obigen Bäume G004a.gra bis G004c.gra:

Knotenzahl:7   Kantenzahl:6

C Satz I.3:

In einem Baum ist stets k-a=1.

(k:Knotenzahl,a:Kantenzahl)

Beweis:

durch vollständige Induktion:

Induktionsanfang: Ein Baum mit 2 Knoten hat genau eine Verbindungskante.

Induktionsschritt:

Entfernt man aus dem vorgegebenen Graph einen Randknoten,so verringert sich definitionsgemäß die Kantenzahl um Eins und die Knotenzahl um Eins. In dem verbleibenden Baum gilt k'-a'=1 nach Induktionsvoraussetzung.Da k=k'+1 und a=a'+1 gilt als auch k-a=1.

Jeder Baum muß stets mindestens einen Randknoten besitzen,da sonst ein Weg (ohne Rücksicht auf die Richtung der Kanten) bei einem minimalen Knotengrad von 2 unbegrenzt fortgesetzt werden könnte.

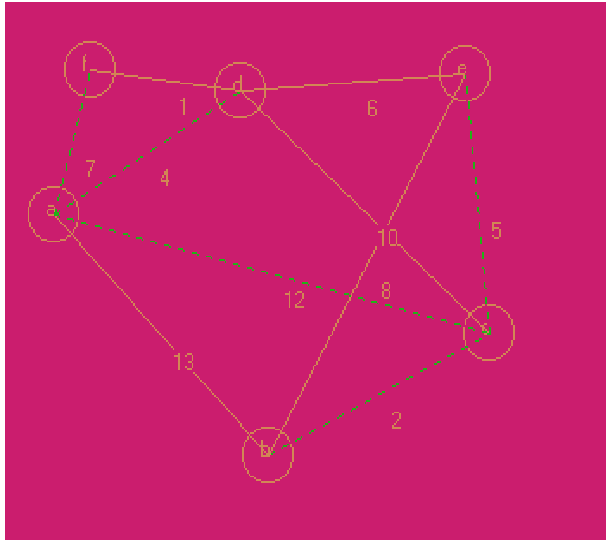
C Aufgabe I.5:

Erstelle mit dem Programm Knotengraph mindestens 3 verschiedene zusammenhängende Graphen und erstelle durch das Menü Pfade/Minimales Gerüst des Graphen einen Baum innerhalb des Graphen.

Bestimme jeweils die Anzahl der Kanten,die zum Graphen,aber nicht zum Baum gehören.

Versuche jeweils auch andere Bäume als Teilgraphen zu finden,die alle Knoten des ursprünglichen Graphen enthalten und stelle die Knoten- und Kantenzahl fest.

Lösung:



G0005.gra

Ergebnis:

Im erstellten Graph sind die zu entfernenden Kanten: 1;6;8;10;13, also 5 Kanten.Knoten:6 Kanten:10

Tabelle:

Anzahl Knoten k:	Anzahl Kanten a:	Anzahl Lösch-Kanten l:
6	10	5
8	12	3
4	6	1

Ergebnis der Tabelle: l=a-k+1

C Satz I.4:

Jeder zusammenhängende schlichte Graph,der kein Baum ist,kann durch Entfernen von Kanten auf einen Baum (Gerüst des Graphen genannt) reduziert werden.Es müssen dazu stets genau l=a-k+1 Kanten entfernt werden.

Beweis:

Nach Satz I.3 hat ein Baum mit k Knoten k-1 Kanten.Also müssen a-(k-1) = a-k+1 = l Kanten gelöscht werden.

Bemerkung:

Ob ein Graph ein Baum ist,kann mittels des Menüs Eigenschaften/Kreise durch Überprüfung der Existenz von Kreisen bestimmt werden.

C Definition I.4:

Jeder Baum,der aus einem zusammenhängenden Graphen durch Löschen von f=a-k+1 Kanten entsteht und daher alle Knoten des Graphen enthält,heißt ein Gerüst des Graphen.

Es ergeben sich nun die Fragen,wie man einen Baum in einem vorgegebenen Graphen konstruieren und einen Baum durchsuchen kann.Diese Fragen werden im nächsten Abschnitt behandelt.

C II Suchverfahren in Graphen,Baum-und Pfadalgorithmen,Kreise,Gerüste

Dieses Kapitel und das nachfolgende stellt eine große Zahl von Pfad,Baum- und Suchalgorithmen in Graphen dar.Ausgangspunkt ist dabei der Algorithmus Erzeugen eines TiefenBaums in einem Graphen.Durch kleinere Änderungen dieses Grundalgorithmus lassen sich dann leicht die Verfahren Erzeugen Aller Pfade zu den übrigen Knoten des Graphen von einem Knoten als Startknoten aus,Erzeugen aller Kreise von einem Startknoten aus und Erzeugen aller minimaler Pfade zu den übrigen Knoten des Graphen von einem Startknoten aus bestimmen.Außerdem liefert der TiefeBaumdurchlauf sofort auch die Anzahl der von einem Startknoten erreichbaren Zielknoten.

Der TiefeBaumdurchlauf gibt Anlaß auch die andere Methode einen Baum zu durchlaufen,nämlich den BreitenBaumdurchlauf zu besprechen.Sowohl Tiefer- als auch BreiterBaumdurchlauf erzeugen innerhalb eines beliebigen Graphen von einem Startknoten als Wurzel aus Bäume,d.h. Gerüste des Graphen.Die Frage nach einem minimalen Gerüst bei einem Kanten-bewerteten Graphen beantwortet dann schließlich der Algorithmus von Kruskal.

So ergibt sich zwanglos von der inneren Problematik des Stoffes her ein geschlossener Unterrichtsgang,der z.B. an die Stelle der bisher üblichen (ausschließlichen) Behandlung von Binärbäumen im Informatikunterricht treten könnte.Dabei lassen sich die bisher hauptsächlich betrachteten Algorithmen Inorder-Durchlauf und Binäres Suchen als Spezialfälle des Suchens aller Pfade in einem Baum bzw. des minimalen Pfades zwischen zwei Knoten problemlos einbeziehen (siehe am Ende dieses Kapitels).

Bei Zeitknappheit könnte dabei eventuell der BreiteBaumdurchlauf entfallen.Bei einem größeren bemessenen zeitlichen Rahmen könnte zum Vergleich mit dem auf der Tiefensuche beruhende Algorithmus Minimale Pfade das wesentlich effizientere Verfahren zur Bestimmung von minimalen Pfaden von einem Startknoten aus von Dijkstra besprochen werden und sich dann der Algorithmus Minimaler Pfad (nach Dijkstra) zwischen zwei Knoten anschließen.Diese Algorithmen sind das Thema des nächsten Kapitels.

Der Quellcode der Algorithmen dieses Kapitels ist in der Unit UPfad zu finden,die die im Unterricht (in Auswahl) zu besprechenden Methoden demonstrieren soll.Einige Methoden befinden sich noch einmal zusätzlich in der Unit UGraph,weil sie von anderen Methoden der Unit UInhgrph zu deren Ausführung benötigt werden.

Ein Unterrichtsgang nach der Mathode DWK wird die im folgenden beschriebenen Quellcodes der Algorithmen auslassen und den Demo-Modus des Programms Knotengraph benutzen,um die Algorithmen an Hand von geeigneten Graphen z.B. an Hand der Aufgaben C II.1 bis C II.5 und ähnlichen Aufgaben problemorientiert zu erarbeiten und darzustellen.Danach sollte das Verfahren gemäß dem in den einzelnen Abschnitten jeweils vorgegebenen Muster der verbalen Beschreibung allgemein erläutert,systematisiert und danach mittels weiterer Aufgaben von den Schülern eingeübt werden.Außer den in diesen Aufgaben genannten Graphen können fast alle der 100 Graphen G001.gra bis G100.gra des Übungsverzeichnisses als Beispiele benutzt werden.

Grundlage der Suchverfahren in Graphen ist ein (vorzugebender) Markierungsalgorithmus,der folgende Gestalt hat:

Verbale Beschreibung:

a)Markiere den Startknoten.

b)Solange noch Kanten von markierten zu unmarkierten Knoten existieren,wähle eine solche Kante und markiere den unmarkierten Knoten.

Die Suchverfahren unterscheiden sich in der Weise,wie die Kanten ausgewählt werden.Da kein markierter Knoten zweimal besucht wird,entsteht auf diese Weise bei einem zusammenhängenden Graph ein Suchbaum.Bei mehreren Komponenten entstehen mehrere Bäume d.h. ein Wald.

C Aufgabe II.1a (Einstiegsproblem):

Erstelle mittels des Programms Knotengraph Menü Pfade/Tiefer Baum zu dem Baum von Aufgabe I.3 c) (G004a.gra) mit dem Knoten M als Startknoten die tiefen Baumpfade in Postorder- und Preorder-Folge.Bestimme mit Hilfe des Demomodus sowie der Ergebnisse im Ausgabefenster in welcher Reihenfolge die Knoten besucht werden.Versuche den Algorithmus allgemein und detailiert zu beschreiben.(Falls die Aufgabe C I 3 des vorigen Kapitels behandelt wurde,kann bei der Konzeption EWK diese Aufgabe auch zur manuellen Bearbeitung,d.h. ohne Einsatz des Programms Knotengraph gestellt werden,wobei die Knotenfolge schriftlich notiert werden sollte.)

Lösung:

Postorder:

M G A Summe: 2.000 Produkt: 1.000

M G I Summe: 2.000 Produkt: 1.000

M G Summe: 1.000 Produkt: 1.000

M P O Summe: 2.000 Produkt: 1.000

M P R Summe: 2.000 Produkt: 1.000

M P Summe: 1.000 Produkt: 1.000

M Summe: 0.000 Produkt: 0.000

Preorder:

M Summe: 0.000 Produkt: 0.000

M G Summe: 1.000 Produkt: 1.000

M G A Summe: 2.000 Produkt: 1.000

M G I Summe: 2.000 Produkt: 1.000

M P Summe: 1.000 Produkt: 1.000

M P O Summe: 2.000 Produkt: 1.000

M P R Summe: 2.000 Produkt: 1.000

Beobachtung und problemorientierte Erarbeitung: Insbesondere durch den Vergleich der beiden Verfahren können Schüler selbständig erkennen,dass bei Postorder die Teilbäume vor deren Wurzelknoten und bei Postorder gerade die Wurzeln vor den Teilbäumen besucht werden.Als weiteres Beispiel kann noch Aufgabe II 1 b (s.u.) benutzt werden,um das Verfahren an Hand eines Graphen,der kein Baum ist,zu veranschaulichen.Das Verfahren wird anschließend verbal systematisiert:

Die Tiefensuche:

Verbale Beschreibung des Algorithmus:

Bei der Tiefensuche wird versucht,die am weitesten vom Startknoten entfernten Knoten zuerst zu besuchen.Daher wird,sobald ein Knoten besucht und markiert wurde,direkt nach der (in der Kantenfolge) ersten von diesem Knoten wieder auslaufenden Kante gesucht,um einen noch weiter entfernt

befindlichen Knoten als Zielknoten dieser Kante aufsuchen und markieren zu können.Erst wenn sich keine Kanten mehr finden lassen,die von einem besuchten Knoten ausgehen (Randknoten) oder alle möglichen Zielknoten dieses Knotens schon markiert wurden,werden die (in der Kantenfolge) nächsten Kanten (und Zielknoten) des als letztes markierten Knoten berücksichtigt.

Bei der Preorder-Reihenfolge werden gemäß dem oben genannten Markierungsalgorithmus jeweils die Inhalte der in der zeitlichen Reihenfolge zuerst besuchten und markierten Knoten vor den Knoten(-Inhalten) des danach markierten Teilbaums (Teilgraphs) ausgegeben.

Bei der Postorder-Reihenfolge werden jeweils die Inhalte der Knoten jedes Teilbaums vor dem Knoten(-Inhalt) des Knotens,von dem der Teilbaum ausgeht,ausgegeben.(Die Inorder-Reihenfolge hat nur bei Graphen,die als Binärbäume aufgefaßt werden können,einen Sinn.Ein entsprechende Methode wird am Ende dieses Kapitels beschrieben.)

Algorithmus Alle tiefen Baumpfade von einem Knoten aus erzeugen

(**Unit UPfad und UGraph)**

Der folgende Algorithmus erzeugt von einem Startknoten ausgehend alle Pfade (unter Berücksichtigung der Kantenrichtungen bei einem gerichteten Graphen) eines tiefen Baumdurchlaufs wahlweise in Preorder oder Postorder-Reihenfolge,wobei bei einem ungerichteten Graph stets ein Gerüst entsteht,bei einem gerichteten Graphen jedoch nicht alle Knoten vom Startknoten aus durch Wege erreichbar sein müssen.

```
procedure TPfadknoten.ErzeugeTiefeBaumPfade(Preorder:Boolean);
var Index:Integer;
    MomentaneKantenliste:TKantenliste;
    P:TGraph;

  procedure GehezuNachbarknoten(Kno:TKnoten;Ka:TKante);
  var Ob:TObject;
      Index:Integer;
  begin
    if not Ka.Zielknoten(Kno).Besucht then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kno);                                   3)
      MomentaneKantenliste.AmEndeanfuegen(Ka);                               4)
      if Preorder then Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);      5)
      Ka.Zielknoten(Kno).Besucht:=true;                                      6)
      if not Ka.Zielknoten(Kno).AusgehendeKantenliste.Leer then              7)
        for Index:=0 to Ka.Zielknoten(Kno).AusgehendeKantenliste.Anzahl-1 do
          GehezuNachbarknoten(Ka.Zielknoten(Kno),Ka.Zielknoten(Kno).AusgehendeKantenliste.
          Kante(Index));
      if not Preorder then                                                   8)
        Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
      MomentaneKantenliste.AmEndeloeschen(Ob);                               9)
    end;
  end;

begin
  MomentaneKantenliste:=TKantenliste.Create;                                 1)
  Graph.Pfadlistenloeschen;
  Graph.LoescheKnotenbesucht;
  Besucht:=true;
  if not AusgehendeKantenliste.Leer then
  for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
    GehezuNachbarknoten(self,AusgehendeKantenliste.Kante(Index));            2)
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
  P:=TGraph.Create;
  P.Knotenliste.AmEndeanfuegen(self);
  if Preorder                                                               10)
```

```
then
    Pfadliste.AmAnfanganfuegen(P)
  else
    Pfadliste.AmEndeanfuegen(P);
end;
```

Erläuterung des Algorithmus:

Nachdem bei 1) als Initialisierung alle Pfadlisten gelöscht sowie alle Besucht-Markierungen zurückgesetzt wurden,der Startknoten als besucht markiert wurde und eine leere Kantenliste als Zwischenspeicher des momentanen Pfades initialisiert wurde,wird für alle vom Startknoten ausgehenden Kanten die rekursive Procedure GehezuallenNachbarknoten mit der momentanen Kante und dem Startknoten als Parameter bei 2) aufgerufen.

In dieser Procedure wird zunächst bei 3) die Pfadrichtung in der die Kante durchlaufen wurde auf den Zielknoten hin gesetzt,was insbesondere für ungerichtete Graphen von Bedeutung ist.Bei 4) wird die Kante in den momentanen Pfad durch Anfügen an die MomentanePfadliste aufgenommen.Vor dem erneuten rekursiven Aufruf der Procedure bei 7) mit dem neuen Knoten als Ausgangsknoten und der nächsten Kante,wird bei 6) der neue Knoten (d.h. der Zielknoten der Kante) als besucht markiert,und falls die Preorder-Suchfolge gewählt wurde,wird der komplette bisherige Suchpfad bei 5) als Graphkopie am Ende an die Pfadliste des Startknotens angefügt.Bei der Postorder-Suchfolge geschieht dies bei Rückkehr aus der Rekursion an der Stelle 8),so daß dieser Pfad erst nach den Pfaden zu den Knoten des Teilbaums eingefügt wird.

Bei 9) wird auf dem Rückweg die letzte Kante aus dem Pfad wieder entfernt.

Bei 10) wird schließlich noch ein Graph,der nur aus dem Startknoten besteht,in die Pfadliste eingefügt.

So erstellt der Algorithmus eine Pfadliste aus sämtlichen tiefen Baumpfaden als Graphen geordnet in der Reihenfolge Preorder oder Postorder im Startknoten.

Die Ausgabe der Pfade kann dann gemäß der Methode TPfadgraph. AlleTiefenBaumPfadevoneinemKnotenzeigen in der Unit UPfad folgendermaßen durchgeführt werden:

```
procedure TPfadgraph.AlletiefenBaumpfadevoneinemKnotenbestimmen(X,Y:Integer;
  Ausgabe:TLabel;var SListe:TStringList;Flaeche:TCanvas);
var Kno:TPfadknoten;
begin
  if not Leer  then
  begin
    if FindezuKoordinatendenKnoten(X,Y,TInhaltsknoten(Kno))=false              11)
    then
      Kno:=TPfadknoten(Anfangsknoten);
    if MessageDlg('Durchlaufordnung Postorder? (ansonsten Preorder!)',          12)
      mtConfirmation, [mbYes, mbNo], 0) = mrYes
    then                                                                        13)
      Kno.ErzeugetiefeBaumpfade(false)
    else
      Kno.ErzeugetiefeBaumpfade(true);
    if Kno.Pfadliste.Leer
    then
      ShowMessage('Keine Pfade')
    else
      Kno.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);                   14)
  end;
  Pfadlistenloeschen;
end;
```

Bei 11) wird zunächst ein in der Nähe der Koordinaten X und Y eines Mausklickpunkts befindlicher Knoten Kno ausgesucht oder falls nicht vorhanden der Anfangsknoten der Graph-Knotenliste ausgewählt.Nach Auswahl von Preorder bzw. Postorder bei 12) werden bei 13) die entsprechenden Baumpfade des Knotens erzeugt.

Durch 14) wird die gesamte Pfadliste des Knoten durchlaufen und die Pfade der Pfadliste rot markiert auf der Objektfläche Flaeche (Paintbox.Canvas)

gezeichnet.Die Knotenfolgen der Pfade werden unter Berücksichtigung von Pfadrichtung,Pfadsumme und Pfadprodukt bestimmt und im Label Ausgabe (Ausgabe1) angezeigt.Außerdem werden der Stringliste SListe die Einträge aus Knotenfolge,Pfadsumme und Pfadprodukt der Pfade als Listeneintrag hinzugefügt.

Falls die Pfade nicht als Pfadlisten gespeichert sondern die Pfade direkt gezeichnet werden sollen,können die Zeilen 5) bzw. 9) durch die Methode Graphzeichen ersetzt werden,die einen Graph(pfad) zunächst markiert zeichnet,eine Demopause einlegt und dann die Markierung des Graphen(-pfads) wieder entfernt:
z.B:

```
if {not} preorder
then
  TPfadgraph(MomentaneKantenliste.Graph).Graphzeichnen(Flaeche,Ausgabe,Bewertung,Sliste,
  TPfadgraph(self.Graph).Demo,TPfadgraph(self.Graph).Pausenzeit,
  TPfadgraph(self.Graph).Kantengenauigkeit);
```

Dadurch wird auch die Methode
AlletiefenBaumpfadevoneinemKnotenbestimmen (etwas) vereinfacht.

Die beiden vereinfachten Methoden sind im folgenden dargestellt:

(Unit UPfad)

```
procedure TPfadknoten.ErzeugeTiefeBaumPfadeeinfach(Preorder:Boolean;Flaeche:TCanvas;
Ausgabe:TLabel;var SListe:TStringlist);
var Index:Integer;
    MomentaneKantenliste:TKantenliste;


  procedure GehezuNachbarknoten(Kno:TKnoten;Ka:TKante);
  label Endproc;
  var Ob:TObject;
      Index:Integer;
  begin
    if Graph.Abbruch then goto Endproc;
    if not Ka.Zielknoten(Kno).Besucht then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kno);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
      if preorder
      then
        TPfadgraph(MomentaneKantenliste.Graph).Graphzeichnen(Flaeche,Ausgabe,Bewertung,Sliste,
        TPfadgraph(Graph).Demo,TPfadgraph(Graph).Pausenzeit,
        TPfadgraph(Graph).Kantengenauigkeit);
      Ka.Zielknoten(Kno).Besucht:=true;
      if not Ka.Zielknoten(Kno).AusgehendeKantenliste.Leer then
        for Index:=0 to Ka.Zielknoten(Kno).AusgehendeKantenliste.Anzahl-1 do
          GehezuNachbarknoten(Ka.Zielknoten(Kno),Ka.Zielknoten(Kno).AusgehendeKantenliste.
          Kante(Index));
      if not preorder
      then
        TPfadgraph(MomentaneKantenliste.Graph).Graphzeichnen(Flaeche,Ausgabe,Bewertung,Sliste,
        TPfadgraph(Graph).Demo,TPfadgraph(Graph).Pausenzeit,
        TPfadgraph(Graph).Kantengenauigkeit);
      MomentaneKantenliste.AmEndeloeschen(Ob);
    end;
    Endproc:
  end;

begin
  if Preorder then
  begin
    Knotenzeichnen(Flaeche,TPfadgraph(Graph).Demo,TPfadgraph(Graph).Pausenzeit);
    SListe.Add(' '+Wert);
  end;
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.LoescheKnotenbesucht;
  Besucht:=true;
  if not AusgehendeKantenliste.Leer then
  for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
    GehezuNachbarknoten(AusgehendeKantenliste.Kante(Index));
  MomentaneKantenliste.Free;
  MomentaneKnatenliste:=nil;
  if not Preorder then
```

```
  begin
    Knotenzeichnen(Flaeche,TPfadgraph(self.Graph).Demo,TPfadgraph(self.Graph).Pausenzeit);
    SListe.Add(' '+self.Wert);
  end;
end;

procedure TPfadgraph.AlletiefenBaumpfadevoneinemKnotenbestimmeneinfach(X,Y:Integer;
  Ausgabe:TLabel;var SListe:TStringList;Flaeche:TCanvas);
var Kno:TPfadknoten;
begin
  if not Leer  then
  begin
    if FindezuKoordinatendenKnoten(X,Y,TInhaltsknoten(Kno))=false
    then
      Kno:=TPfadknoten(Anfangsknoten);
    if MessageDlg('Durchlaufordnung Postorder? (ansonsten Preorder!)',
      mtConfirmation, [mbYes, mbNo], 0) = mrYes
    then
      Kno.ErzeugetiefeBaumpfadeeinfach(false,Flaeche,Ausgabe,SListe)
    else
      Kno.ErzeugetiefeBaumpfadeeinfach(true,Flaeche,Ausgabe,SListe);
  end;
end;
```

Diese Vereinfachung der Methode durch sofortiges Zeichnen kann in gleicher Weise auch noch angewendet werden bei den Algorithmen Alle Pfade erzeugen,Alle Kreise erzeugen,Alle minimalen Pfade erzeugen und Alle weiten Baumpfade erzeugen.Der Nachteil liegt einerseits darin,dass die Pfade einer evtl. weiteren Anwendung nicht mehr zur Verfügung stehen,und dass der letzte erzeugte Pfad nicht mehr als Bild auf der Zeichenoberfläche nach Ende des Algorithmus erhalten bleibt.

Im Menü Tiefer Baum wird die Methode AlletiefenBaumpfadevoneinemKnoten bestimmen durch die folgenden Zeilen mittels des Letzten Mausklickknotens als Startknoten aufgerufen:

```
GraphH:=Graph.InhaltskopiedesGraphen(TPfadgraph,TPfadknoten,TInhaltskante,false);
with GraphH.LetzterMausklickknoten do
TPfad-
Graph(GraphH).AlletiefenBaumpfadevoneinemKnotenbestimmen(X,Y,Bewertung,Ausgabe1,true,Sliste,Pa
intbox.Canvas);
```

Der Aufruf der übrigen in diesem und den folgenden Kapitel enthaltenden Methoden erfolgt in der gleichen Weise.Deshalb wird darauf im weiteren nicht mehr näher eingegangen.

Zur Vorbereitung des Algorithmus sollte das Verfahren mehrfach zuvor per Hand an Hand geeigneter Graphen geübt werden.Zur Kontrolle können dann die Pfade mit Hilfe des Menüs Pfade/Tiefer Baum und dem Ausgabefenster des Programms Knotengraph ausgegeben werden (entweder nach Programmierung des Verfahrens mittels der Enwicklungsumgebung oder durch Benutzung des fertigen Programms).

Bemerkung:

Auf der Tiefensuche beruht auch das topologische Sortieren in Paragraph VIII.

<u>C Aufgabe II.1b:</u>

Erstelle (per Hand) zu den Bäumen von Aufgabe I.2 a) und I.5 für jeweils alle Knoten des Graphen als Startknoten die tiefen Baumpfade in Preorder- und Postorder-Folge (Startknoten A bzw. a).

Kontrolliere jeweils die Ergebnisse mit Hilfe von Knotengraph.(durch Programmierung des Verfahrens mittels der Entwicklungsumgebung bzw. bei Benutzung des fertigen Programms mit dem Menü Pfade/Tiefer Baum.Bei Verwendung der letzteren Option kann diese Aufgabe auch zur problemorientierten Erarbeitung des Algorithmus mittels des Demomodus dienen.)

Lösung für I.2 a),I.5 (G002.gra,G0005.gra nur für Knoten A bzw. a) :

G0002.gra (Startknoten A):

Preorder:

A Summe: 0.000 Produkt: 0.000

A B Summe: 1.000 Produkt: 1.000

A B D Summe: 2.000 Produkt: 1.000

A B D C Summe: 3.000 Produkt: 1.000

A B D C H Summe: 4.000 Produkt: 1.000

A B D C H G Summe: 5.000 Produkt: 1.000

A B D C H G E Summe: 6.000 Produkt: 1.000

A B D C H G F Summe: 6.000 Produkt: 1.000

A B D C H G I Summe: 6.000 Produkt: 1.000

Postorder (Startknoten A):

A B D C H G E Summe: 6.000 Produkt: 1.000

A B D C H G F Summe: 6.000 Produkt: 1.000

A B D C H G I Summe: 6.000 Produkt: 1.000

A B D C H G Summe: 5.000 Produkt: 1.000

A B D C H Summe: 4.000 Produkt: 1.000

A B D C Summe: 3.000 Produkt: 1.000

A B D Summe: 2.000 Produkt: 1.000

A B Summe: 1.000 Produkt: 1.000

A Summe: 0.000 Produkt: 0.000

G0005.gra (Startknoten a):

Preorder(Startknoten a):

a Summe: 0.0 Produkt: 0.0
a d Summe: 1.0 Produkt: 1.0
a d c Summe: 2.0 Produkt: 1.0
a d c b Summe: 3.0 Produkt: 1.0
a d c b e Summe: 4.0 Produkt: 1.0
a d f Summe: 2.0 Produkt: 1.0

Postorder:(Startknoten a)
a d c b e Summe: 4.0 Produkt: 1.0
a d c b Summe: 3.0 Produkt: 1.0
a d c Summe: 2.0 Produkt: 1.0
a d f Summe: 2.0 Produkt: 1.0
a d Summe: 1.0 Produkt: 1.0
a Summe: 0.0 Produkt: 0.0

Algorithmus Anzahl der Zielknoten von einem Knoten aus bestimmen

(Unit UPfad und UGraph)

```
function TPfadknoten.AnzahlPfadZielknoten: Integer;
begin
  Graph.Pfadlistenloeschen;
  ErzeugeTiefeBaumpfade(true);
  AnzahlPfadZielknoten:=Pfadliste.Anzahl-1;
end;
```

Verbale Beschreibung und Erläuterung des Algorithmus:

Die Tiefensuche kann schließlich auch noch dazu benutzt werden,um die Anzahl der vom Startknoten auf Pfaden bzw. Wegen erreichbaren Zielknoten zu bestimmen,was insbesondere für gerichtete Graphen von Interesse ist.Die Anzahl der Pfade in der Pfadliste des Startknotens entspricht nämlich einfach der Anzahl der Zielknoten.

C Aufgabe II.1c:

Bestimme zu dem Baum von Aufgabe I.3 c) mit dem Knoten M als Startknoten die Anzahl der Zielknoten mit Hilfe des Menüs Pfade/Anzahl Zielknoten von Programm Knotengraph (bzw. mit einem selbst erstellten Algorithmus mittels der Entwicklungsumgebung).

Lösung: Anzahl: 6

Algorithmus Alle Pfade von einem Knoten aus erzeugen

(Unit UPfad und UGraph)

Erläuterung des Algorithmus:

Wenn man in der Procedure GehezuallenNachbarknoten nach dem rekursiven Aufruf dieser Procedure durch sich selbst bei 7) die Besucht-Markierung des Zielknotens durch die zusätzliche Zeile Ka.Zielknoten(Kno).Besucht:= false wieder löscht und außerdem z.B. nur die Preorder-Suche zuläßt,indem man die Anweisungen zu 8) für die Postorder-Suche entfernt,erhält man einen Algorithmus,der **alle** Pfade eines Graphen durchläuft,und die Pfade in der Pfadliste des Startknotens als auch die zu den einzelnen Knoten führenden Pfade in den Pfadlisten der Knoten abspeichert.Es ist hier sinnvoll die Anweisung 10) zu entfernen,da nur die Pfade und nicht noch zusätzlich der Startknoten ausgegeben werden soll.

Der Algorithmus ist im folgenden wiedergegeben.Die zusätzlich eingefügte Anweisung ist mit (*****) markiert.

```
procedure TPfadknoten.ErzeugeallePfade;
var Index:Integer;
    MomentaneKantenliste:TKantenliste;

  procedure GehezuallenNachbarknoten(Kno:TKnoten;Ka:TKante);
  var Ob:TObject;
      Index:Integer;
  begin
    if not Ka.Zielknoten(Kno).besucht then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kno);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
      Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);        5)
      Ka.Zielknoten(Kno).Besucht:=true;
      if not Ka.Zielknoten(Kno).AusgehendeKantenliste.Leer then
        for Index:= 0 to Ka.Zielknoten(Kno).AusgehendeKantenliste.Anzahl-1 do
          GehezuallenNachbarknoten(Ka.Zielknoten(Kno),Ka.Zielknoten(Kno).
```

```
          AusgehendeKantenliste.Kante(Index));
        MomentaneKantenliste.AmEndeloeschen(Ob);
        Ka.Zielknoten(Kno).Besucht:=false;                                (*****)
      end;
    end;

begin
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.Pfadlistenloeschen;
  Graph.LoescheKnotenbesucht;
  Besucht:=true;
  if not AusgehendeKantenliste.Leer then
    for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
      GehezuallenNachbarknoten(self,AusgehendeKantenliste.Kante(Index));
  MomentaneKantenliste.Free;
  MomentaneKnatenliste:=nil;
end;
```

__Verbale Beschreibung:__

Der TiefeBaum-Durchlauf wird in Preorder-Folge durchgeführt und die Besucht-Markierung der besuchten Knoten werden auf dem Rückweg wieder gelöscht.Bei manuellem Vorgehen wird **jeder** Pfad beim Erreichen eines neuen Knoten über eine Auswahlkante schriftlich notiert.

Die Ausgabe der Pfade kann dann wie oben bei der Ausgabe der tiefen Baumpfade erfolgen: (dargestellt in der Methode AllePfadevonKnotenzeigen in der Unit UPfad.)

```
procedure TPfadgraph.AllePfadevoneinemKnotenbestimmen(X,Y:Integer;
  Ausgabe:TLabel;var SListe:TStringList;Flaeche:TCanvas);
var Kno:TPfadknoten;
begin
  if not Leer  then
  begin
    if FindezuKoordinatendenKnoten(X,Y,TInhaltsknoten(Kno))=false then
      Kno:=TPfadknoten(Anfangsknoten);
      Kno.ErzeugeallePfade;
    if Kno.Pfadliste.Leer
    then
      ShowMessage('Keine Pfade')
    else
      Kno.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
  end;
  self.Pfadlistenloeschen;
end;
```

Es gilt wieder:

Falls die Pfade nicht als Pfadlisten gespeichert sondern die Pfade direkt gezeichnet werden sollen,kann die Zeile 5) noch zusätzlich wieder durch Anweisungen zur Zeichnung der momentanen Kantenliste ersetzt werden:

z.B:

```
TPfadgraph(MomentaneKantenliste.Graph).Graphzeichnen(Flaeche,Ausgabe,Bewertung,Sliste,
  TPfadgraph(Graph).Demo,TPfadgraph(Graph).Pausenzeit,
  TPfadgraph(self.Graph).Kantengenauigkeit);
```

oder als Einzelbefehlsfolge:

```
TObject(T):=MomentaneKantenliste.Graph.Kopie;
T.FaerbeGraph(clred,psdot);
T.ZeichneGraph(Flaeche);
...
```

(Die entsprechenden Parameter müssen natürlich dazu der Methode ErzeugeAllePfade hinzugefügt werden:vgl. die Methode ErzeugeTiefeBaumpfadeeinfach)

Algorithmus Alle Pfade und minimalen Pfad zwischen zwei Knoten erzeugen

(Unit UPfad)

Verbale Beschreibung:

Verändert man die Methode TPfadknoten.ErzeugeAllePfade bzw. den entsprechenden Algorithmus in der Weise,dass nur die Pfade der Momentanen-Kantenliste in die Pfadliste des Startknotens eingefügt (bzw. bei manuellem Vorgehen notiert) werden,die als Endknoten einen vorgegebenen Zielknoten ZKno des Graphen haben,erhält man ein Verfahren zur Bestimmung aller Pfade zwischen zwei Knoten.Wird dann noch zusätzlich bei der Einfügung dieser Pfade der jeweils kleinste Pfad (bezüglich der Summe der Kanten) als Kantenliste Minlist gespeichert (bzw. bei manuellem Vorgehen ausgewählt), erhält man ein Verfahren zur Bestimmung des Abstandes zweier Knoten:

C Aufgabe II.1d:

Bestimme zu dem Graph G001.gra von Aufgabe C I 1 (per Hand) alle Pfade vom Knoten A aus.Kontrolliere das Ergebnis mit Hilfe von Knotengraph.(durch Programmierung des Verfahrens mittels der Entwicklungsumgebung bzw. durch Benutzung des fertigen Programms mittels des Menüs Pfade/Alle Pfade.Bei Verwendung der letzteren Option kann diese Aufgabe auch zur problemorientierten Erarbeitung des Algorithmus mittels des Demomodus dienen.)

Lösung:

A B Summe: 1.000 Produkt: 1.000
A B D Summe: 2.000 Produkt: 1.000
A B D C Summe: 3.000 Produkt: 1.000
A B C Summe: 2.000 Produkt: 1.000
A B C D Summe: 3.000 Produkt: 1.000
A C Summe: 1.000 Produkt: 1.000
A C B Summe: 2.000 Produkt: 1.000
A C B D Summe: 3.000 Produkt: 1.000
A C D Summe: 2.000 Produkt: 1.000
A C D B Summe: 3.000 Produkt: 1.000
A D Summe: 1.000 Produkt: 1.000
A D B Summe: 2.000 Produkt: 1.000
A D B C Summe: 3.000 Produkt: 1.000
A D C Summe: 2.000 Produkt: 1.000
A D C B Summe: 3.000 Produkt: 1.000

Erläuterung des Algorithmus:

Die eben beschrieben Änderungen sind wieder mittels (*) und (**) gekennzeichnet:

```
procedure TPfadKnoten.ErzeugeAllePfadeundMinimalenPfad(ZKno:TPfadKnoten;
var Minlist:TKantenliste);
var Index:Integer;
    MomentaneKantenliste:TKantenliste;

  procedure GehezuallenNachbarknoten(Kno:TKnoten;Ka:TKante);
  var Ob:TObject;
      Index:Integer;
  begin
    if not Ka.Zielknoten(Kno).Besucht then
```

```
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kno);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
      if Ka.Zielknoten(Kno)= ZKno then                                          (*)
      begin
        Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
        if (MomentaneKantenliste.WertsummederElemente(Bewertung)<              (**)
          Minlist.WertsummederElemente(Bewertung)) then
            Minlist:=MomentaneKantenliste.Kopie;
      end;
      Ka.Zielknoten(Kno).Besucht:=true;
      if not Ka.Zielknoten(Kno).AusgehendeKantenliste.Leer then
        for Index:= 0 to Ka.Zielknoten(Kno).AusgehendeKantenliste.Anzahl-1 do
          GehezuallenNachbarknoten(Ka.Zielknoten(Kno),Ka.Zielknoten(Kno).
          AusgehendeKantenliste.Kante(Index));
      MomentaneKantenliste.AmEndeloeschen(Ob);
      Ka.Zielknoten(Kno).Besucht:=false;
    end;
  end;

begin
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.Pfadlistenloeschen;
  Graph.LoescheKnotenbesucht;
  Besucht:=true;
  if not AusgehendeKantenliste.Leer then
    for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
      GehezuallenNachbarknoten(self,AusgehendeKantenliste.Kante(Index));
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
end;
```

Die Methode wird aufgerufen durch die folgende Methode,wobei als Minlist die gesamte (maximal große) Kantenliste des Graphen übergeben wird:

```
function TPfadgraph.AllePfadeundminimalenPfadzwischenzweiKnotenbestimmen(X,Y:Integer;
  Ausgabe:TLabel;var SListe:TStringList;Flaeche:TCanvas):Boolean;
var Kno1,Kno2:TPfadknoten;
    Gefunden:Boolean;
    Minlist:TKantenliste;
begin
  result:=false;
  if self.ZweiKnotenauswaehlen(X,Y,TInhaltsknoten(Kno1),TInhaltsknoten(Kno2),Gefunden)
    and Gefunden
  then
  begin
    Ausgabe.Caption:='Berechnung läuft..
    Minlist:=Kantenliste.Kopie;
    Kno1.ErzeugeAllePfadeundMinimalenPfad(Kno2,Minlist);
    result:=true;
    if Kno1.Pfadliste.Leer then
    begin
      Ausgabe.Caption:='';
      Ausgabe.Refresh;
      ShowMessage('Keine Pfade zwischen den Knoten');
    end
    else
      Kno.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
   if not Minlist.leer then
        begin
          TInhaltsgraph(Minlist.UGraph).FaerbeGraph(clred,psdot);
          TInhaltsgraph(Minlist.UGraph).ZeichneGraph(Flaeche);
          Ausgabe.caption:='Minimaler
          Pfad:'+Minlist.UGraph.InhaltallerKnoten(ErzeugeKnotenstring)
          +' Summe: '+
          RundeZahltostring(Minlist.UGraph.Kantensumme(Bewertung),Kantengenauigkeit);
          Messagebeep(0);
          Pause(2000);
          Ausgabe.Refresh;
        end;
    SListe.Add(Ausgabe.Caption);
    Ausgabe.Caption:='';
    Ausgabe.Refresh;
  end
  else
    result:=false;
end;
```

Der Algorithmus hat den Vorteil,dass er durch leichte Änderungen aus dem Verfahren TPfadknoten.ErzeugeAllepfade zu gewinnen ist,jedoch den Nachteil

durch Untersuchung aller Pfade nicht besonders effizient zu sein.Ein besseres Verfahren ist die Benutzung des Algorithmus nach Dijkstra.Eine entsprechende Methode wird im Abschnitt III Alle Pfade sowie minimaler Pfad zwischen zwei Knoten beschrieben.Diese Verfahren werden dann auch durch die entsprechenden Algorithmen des Programms Knotengraph benutzt,während die eben beschriebene Methoden zwar in der Unit UPfad vorhanden sind,aber nicht weiter angewendet werden.

Sie sind aber z.B. als Klausuraufgabe als Erweiterung des Algorithmus Alle-Pfade bestimmen sehr geeignet.

C Aufgabe II.1e:

Bestimme zu dem Graph G001.gra von Aufgabe C I 1 per Hand alle Pfade und den minimalen Pfad zwischen den Knoten A und D.Kontrolliere das Ergebnis mit Hilfe von Knotengraph. (durch Programmierung des Verfahrens mittels der Entwicklungsumgebung bzw. durch Benutzung des fertigen Programms mittels des Menüs Pfade/Alle Pfade zwischen zwei Knoten und Abstand von zwei Knoten,ersteres mit und letzteres ohne Demomodus.Diese Aufgabe kann auch zur problemorientierten Erarbeitung des Algorithmus Alle Pfade zwischen zwei Knoten mittels des Demo-Modus dienen.Geeignet sind auch die Graphen G002.gra und G005.gra)

Lösung:

Alle Pfade:

A B D Summe: 2.000 Produkt: 1.000
A B C D Summe: 3.000 Produkt: 1.000
A C B D Summe: 3.000 Produkt: 1.000
A C D Summe: 2.000 Produkt: 1.000
A D Summe: 1.000 Produkt: 1.000

Minimaler Pfad:

A D Summe: 1.000

Durch weitere kleinere Veränderungen des Quellcodes der Methode TPfadknoten.ErzeugeallePfade können nun auch alle durch den Knoten verlaufenden Kreise ermittelt werden:

Algorithmus Alle Kreise durch Knoten erzeugen

(Unit UPfad und UGraph)

```
procedure TPfadknoten.ErzeugeKreise;
var Index:Integer;
    MomentaneKantenliste:TKantenliste;


  procedure GehezudenNachbarknoten(Kno:TKnoten;Ka:TKante);
  var Ob:TObject;
      Index:Integer;

  begin
    if not Ka.KanteistSchlinge then
      if not Ka.Zielknoten(Kno).besucht                                            (*)
      then
      begin
        Ka.Pfadrichtung:=Ka.Zielknoten(Kno);
        MomentaneKantenliste.AmEndeanfuegen(Ka);
        Ka.Zielknoten(Kno).Besucht:=true;
        if not Ka.Zielknoten(Kno).AusgehendeKantenliste.Leer then
          for Index:=0 to Ka.Zielknoten(Kno).AusgehendeKantenliste.Anzahl-1 do
```

```
          GehezudenNachbarknoten(Ka.Zielknoten(Kno),Ka.Zielknoten(Kno).
          AusgehendeKantenliste.Kante(Index));
        MomentaneKantenliste.AmEndeloeschen(Ob);
        Ka.Zielknoten(Kno).Besucht:=false;
      end
      else                                                                (**)
      if (Ka.Zielknoten(Kno)=self) and
      (Ka<>MomentaneKantenliste.Kante(MomentaneKantenliste.Letztes)) then
      begin
        Ka.Pfadrichtung:=Ka.Zielknoten(Kno);
        MomentaneKantenliste.AmEndeanfuegen(Ka);
        Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);      (***)
        MomentaneKantenliste.AmEndeloeschen(Ob);
      end;
  end;


begin
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.Pfadlistenloeschen;
  Graph.LoescheKnotenbesucht;
  Besucht:=true;
  if not AusgehendeKantenliste.Leer then
    for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
      GehezudenNachbarknoten(self,AusgehendeKantenliste.Kante(Index));
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
end;
```

Erläuterung des Algorithmus:

Der Unterschied zum Algorithmus ErzeugeallePfade besteht darin,dass jetzt die Bedingung

if not Ka.Zielknoten(Kno).Istbesucht bei (*)

außer dem then-Teil einen zusätzlichen else-Teil bei (**)

erhält,in dem jeweils zusätzlich abgefragt,ob der Zielknoten gleich dem Startknoten ist und ob der Pfad,der zum Zielknoten führt nicht nur aus einer Kante besteht.Wenn diese Bedingung erfüllt ist,wird jetzt der MomentanePfad,der dann ein Kreis ist,d.h.

die MomentaneKantenliste in die Pfadliste des Startknoten bei (***)

eingefügt.

Diese Anweisung kann natürlich wieder,wenn die Pfade nicht gespeichert werden sollen, wie oben durch folgende Anweisungen ersetzt werden:

z.B.:

```
TPfadgraph(MomentaneKantenliste.Graph).Graphzeichnen(Flaeche,Ausgabe,Bewertung,Sliste,
  TPfadgraph(self.Graph).Demo,TPfadgraph(self.Graph).Pausenzeit,
  TPfadgraph(self.Graph).Kantengenauigkeit);
```

bzw. als Einzelanweisungen:

```
TObject(T):=MomentaneKantenliste.Graph.Kopie;
T.FaerbeGraph(clred,psdot);
T.ZeichneGraph(Flaeche);
...
...
```

(Die entsprechenden Parameter müssen natürlich dazu der Methode ErzeugeKreise hinzugefügt werden: vgl. Methode ErzeugeTiefeBaumpfadeeinfach)

Verbale Beschreibung:

Der Algorithmus Erzeuge alle Pfade wird duchgeführt,wobei nach der Bestimmung des jeweiligen Zielknotens abgefragt wird,ob der Zielknoten der Startknoten ist und schon ein Pfad der Länge größer als zwei Kanten durchlaufen wurde.Bei manueller Durchführung werden dann nur diese Pfade notiert.

Die Ausgabe der Pfade bei Quellcodeerstellung kann dann analog wie oben bei der Ausgabe des Algorithmus AllePfade mittels der Methode TPfadgraph.-AlleKreisevoneinemKnotenbestimmen in der Unit UPfad erfolgen.

Diese Methode ist bis auf den Aufruf Kno.ErzeugeallePfade,der durch Kno.ErzeugeKreise ersetzt wird,identisch mit der oben angegebenen Methode und wird hier deshalb nicht noch einmal aufgeführt.

Eine vereinfachte Version der Methode ErzeugeKreise ist die Function TKnoten.KnotenistKreisknoten in der Unit UGraph,die lediglich testet,ob durch einen Knoten ein Kreis verläuft und nach dem ersten Kreis,der durch den Knoten verläuft,sucht.Wenn dieser gefunden ist,bricht der Algorithmus ab.Wenn kein Kreis gefunden wurde wird false zurückgegeben sonst true.

Algorithmus KnotenistKreisknoten

(Unit UPfad und UGraph)

```
function TKnoten.KnotenistKreisknoten:Boolean;
label Endproc;
var Index:Integer;
    MomentaneKantenliste:TKantenliste;
    Gefunden:Boolean;

  procedure GehezudenNachbarknoten(Kno:TKnoten;Ka:TKante);
  label Ende;
  var Ob:TObject;
      Index:Integer;
  begin
    if Gefunden then goto Ende;                                        (*****)
    if not Ka.KanteistSchlinge then
      if not Ka.Zielknoten(Kno).Besucht  then
      begin
        Ka.Pfadrichtung:=Ka.Zielknoten(Kno);
        MomentaneKantenliste.AmEndeanfuegen(Ka);
        Ka.Zielknoten(Kno).Besucht:=true;
        if not Ka.Zielknoten(Kno).AusgehendeKantenliste.Leer then
          for Index:=0 to Ka.Zielknoten(Kno).AusgehendeKantenliste.Anzahl-1 do
            GehezudenNachbarknoten(Ka.Zielknoten(Kno),Ka.Zielknoten(Kno).
            AusgehendeKantenliste.Kante(Index));
        MomentaneKantenliste.AmEndeloeschen(Ob);
        Ka.Zielknoten(Kno).Besucht:=false;
      end
      else
        if (Ka.Zielknoten(Kno)=self) and
        (Ka<>MomentaneKantenliste.Kante(MomentaneKantenliste.Letztes))
        then
          Gefunden:=true;
    Ende:
  end;

begin
  Gefunden:=false;
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.Pfadlistenloeschen;
  Graph.LoescheKnotenbesucht;
  Besucht:=true;
  if not {Startknoten.}AusgehendeKantenliste.Leer then
  for Index:=0 to {Startknoten.}AusgehendeKantenliste.Anzahl-1 do
  begin
    GehezudenNachbarknoten(self,{Startknoten.}AusgehendeKantenliste.Kante(Index));
    if Gefunden then goto Endproc;
  end;
  Endproc:
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
  KnotenistKreisknoten:=Gefunden;
end;
```

Erläuterung des Algorithmus:

Durch die Zeile (****) wird der Ablauf vorzeitig beendet,wenn ein Kreis gefunden wurde.

Schließlich kann durch Untersuchung aller Knoten des Graphen noch getestet werden,ob insgesamt im Graphen ein Kreis vorhanden ist:

Algorithmus GraphhatKreise

(Unit UPfad und UGraph)

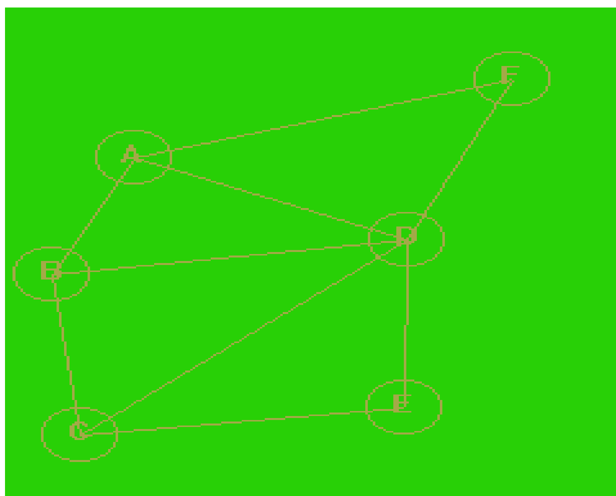
```
function TGraph.GraphhatKreise:Boolean;
label Endproc;
var Index:Integer;
    Gefunden:Boolean;
begin
  Gefunden:=false;
  if not Knotenliste.Leer then
  for Index:=0 to Knotenliste.Anzahl-1 do
  begin
    Gefunden:=Knotenliste.Knoten(Index).KnotenistKreisknoten;
    if Gefunden then goto Endproc;
  end;
  Endproc:
  GraphhatKreise:=Gefunden;
end;
```

Verbale Beschreibung (GraphhatKreise und Kreisknoten):

Wenn irgendein Knoten Startknoten eines Kreises ist,ist der Knoten ein Kreisknoten,und der Graph enthält einen Kreis.

C* Aufgabe II.2:

Suche in dem folgenden Graphen nach dem oben beschriebenen Algorithmus (per Hand) alle Kreise,die durch den Knoten A verlaufen,an.Kontrolliere das Ergebnis mit Hilfe von Knotengraph.(durch Programmierung des Verfahrens mit der Entwicklungsumgebung bzw. durch Benutzung des fertigen Programms Knotengraph mittels des Menüs Pfade/Alle Kreise.Bei Verwendung der letzteren Option kann diese Aufgabe auch zur problemorientierten Erarbeitung der Algorithmen mittels des Demomodus dienen.)



Lösung:

A B D A Summe: 3.000 Produkt: 1.000
A B D F A Summe: 4.000 Produkt: 1.000
A B C E D A Summe: 5.000 Produkt: 1.000
A B C E D F A Summe: 6.000 Produkt: 1.000
A B C D A Summe: 4.000 Produkt: 1.000
A B C D F A Summe: 5.000 Produkt: 1.000

A D B A Summe: 3.000 Produkt: 1.000
A D C B A Summe: 4.000 Produkt: 1.000
A D E C B A Summe: 5.000 Produkt: 1.000
A D F A Summe: 3.000 Produkt: 1.000
A F D A Summe: 3.000 Produkt: 1.000
A F D B A Summe: 4.000 Produkt: 1.000
A F D C B A Summe: 5.000 Produkt: 1.000
A F D E C B A Summe: 6.000 Produkt: 1.000

Schließlich kann die Methode TKnoten.ErzeugeallePfade auch noch zusätzlich so verändert werden,dass alle minimalen Pfade die von einem Knoten aus zu den anderen Knoten des Graphen verlaufen,bestimmt werden können:

(Eine andere,schnellere Implementation nach Dijkstra wird im nächsten Kapitel beschrieben.)

Algorithmus Alle minimalen Pfade von einem Knoten erzeugen

(Unit UPfad und UGraph)

```
procedure TPfadKnoten.ErzeugeminimalePfade;
var Index,Index1:Integer;
    HilfKantenliste:TKantenliste;
    MomentaneKantenliste:TKantenliste;


  procedure GehezuallenNachbarn(Kno:TKnoten;Ka:TKante);
  var Ob:TObject;
      Index:Integer;
  begin
    if not Ka.Zielknoten(Kno).Besucht then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kno);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
      Ka.Zielknoten(Kno).Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.UGraph); (****)
      Ka.Zielknoten(Kno).Besucht:=true;
      if not Ka.Zielknoten(Kno).AusgehendeKantenliste.Leer then
        for Index:=0 to Ka.Zielknoten(Kno).AusgehendeKantenliste.Anzahl-1 do
          GehezuallenNachbarn(Ka.Zielknoten(Kno),Ka.Zielknoten(Kno).
           AusgehendeKantenliste.Kante(Index));
      MomentaneKantenliste.AmEndeloeschen(Ob);
      Ka.Zielknoten(Kno).Besucht:=false;
    end;
  end;

begin
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.Pfadlistenloeschen;
  Graph.LoescheKnotenbesucht;
  Besucht:=true;
  if not AusgehendeKantenliste.Leer then
    for Index1:=0 to AusgehendeKantenliste.Anzahl-1 do
      GehezuallenNachbarn(self,AusgehendeKantenliste.Kante(Index1));
  LoeschePfad;
  if not Graph.Knotenliste.Leer then                                          1)
  for Index:= 0 to Graph.Knotenliste.Anzahl-1 do
  begin
    if  Graph.Knotenliste.Knoten(Index)<>self then
      if not Graph.Knotenliste.Knoten(Index).Pfadliste.Leer then
      begin
        Hilfkantenliste:=TGraph(Graph.Knotenliste.Knoten(Index).
        MinimalerPfad(Bewertung)).Kantenliste;                                2)
        Pfadliste.AmEndeanfuegen(HilfKantenliste.Kopie.Graph);                3)
        Hilfkantenliste.Free;
        Hilfkantenliste:=nil;
      end;
  end;
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
end;


function KleinererPfad(X1,X2:TObject;Wert:Twert):Boolean;
var Kali1,Kali2:TKantenliste;
begin
```

```
  Kali1:=TGraph(X1).Kantenliste.Kopie;
  Kali2:=TGraph(X2).Kantenliste.Kopie;
  KleinererPfad:=Kali1.WertsummederElemente(Wert)<
  Kali2.WertsummederElemente(Wert);                                    5)
end;

function TKnoten.MinimalerPfad(Wert:TWert):TPfad;
var Hilfliste:TPfadliste;
begin
  Hilfliste:=Pfadliste.Kopie;
  if not Hilfliste.Leer
  then
  begin
    Hilfliste.Sortieren(KleinererPfad,Wert);                           4)
    MinimalerPfad:=Hilfliste.Pfad(0);                                  6)
  end
  else
    MinimalerPfad:=TPfad(TGraph.Create);                               6)
end;
```

Erläuterung des Algorithmus:

Die Procedure GehezuallenNachbarn ist identisch mit der Procedure GehezuallenNachbarknoten bis auf die Tatsache,daß in ihr die Pfade bei der Marke (****) in den Zielknoten und nicht im Startknoten gespeichert werden.

Änderungen ergeben sich dann auch im 2.Teil der aufrufenden Methode ErzeugeminimalePfade ab Markierung 1).Bei 2) wird hier zunächst nacheinander für alle Knoten des Graphen,deren Pfadliste nicht leer sind und die nicht der Startknoten sind,einer Variablen Hilfskantenliste jeweils der minimale Pfad (d.h. der Pfad mit kleinsten Pfadbewertung) der Pfadlisten der einzelnen Knoten zugeordnet,um danach diese Pfade bei 3) alle in die anfänglich leere Pfadliste des Startknotens einzufügen.

Auf diese Weise enthält die Pfadliste des Startknotens alle minimalen Pfade.

In der Function MinimalerPfad wird bei 4) eine Kopie der Pfadliste der einzelnen Knoten nach dem Vergleichskriterium KleinererPfad sortiert und diese Funktion vergleicht bei 5) zwei Kantenlisten eines ihr als Parameter übergebenden Graphen nach dem Kriterium der kleinsten Wertsumme der in ihr enthaltenen Kantenwerte.

Die Function MinimalerPfad gibt dann bei 6) entweder den kleinsten Pfad,der sich in der 0.Position der Pfadliste befindet zurück oder,falls die Pfadliste leer ist,einen leeren Graphen.

Verbale Beschreibung:

Der Algorithmus Erzeuge Alle Pfade (vom Startknoten aus) wird durchgeführt und die Pfade zu den einzelnen Knoten des Graphen werden in den einzelnen Knoten gespeichert (manuell: schriftlich zu jedem Knoten gehörig notiert) und dort der Pfadlänge nach aufsteigend sortiert.Anschließend werden die jeweils kleinsten Pfade (zu den einzelnen Knoten) zusammengestellt und ausgegeben.

Die Anzeige und Ausgabe der minimalen Pfade kann dann wieder wie schon mehrmals weiter oben beschrieben und erläutert (z.B. bei der Anzeige der tiefen Baumpfade) durch `Kno.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true)` durchgeführt werden:

Das Verfahren von Dijkstra zur Erzeugung minimaler Pfade arbeitet wesenlich effizienter,kann aber nicht mehr (nach didaktischen Gesichtspunkten) aus dem TiefenBaumDurchlaufalgorithmus abgeleitet werden.Ihm sollte,wenn es um die Schnelligkeit des Algorithmus geht der Vorzug gegeben werden.Es wird im nächsten Kapitel III beschrieben und sollte als Vergleichsalgorithmus mit größerer Effizienz zum eben geschilderten Verfahren ,wenn genug Zeit zur Verfügung steht, im Unterricht unbedingt besprochen werden.Dieser Algorith-

mus kommt deshalb auch zur Bestimmung der minimalen Pfade von einem Knoten aus und des minimalen Pfades zwischen zwei Knoten in den entsprechenden Algorithmen des Menüs Pfade im Programm Knotengraph zum Einsatz(bzw. der Ford-Alg.).

C Aufgabe II.3: (Übungsaufgabe)

Gegeben ist der folgende Graph.Bestimme (zur Übung per Hand)

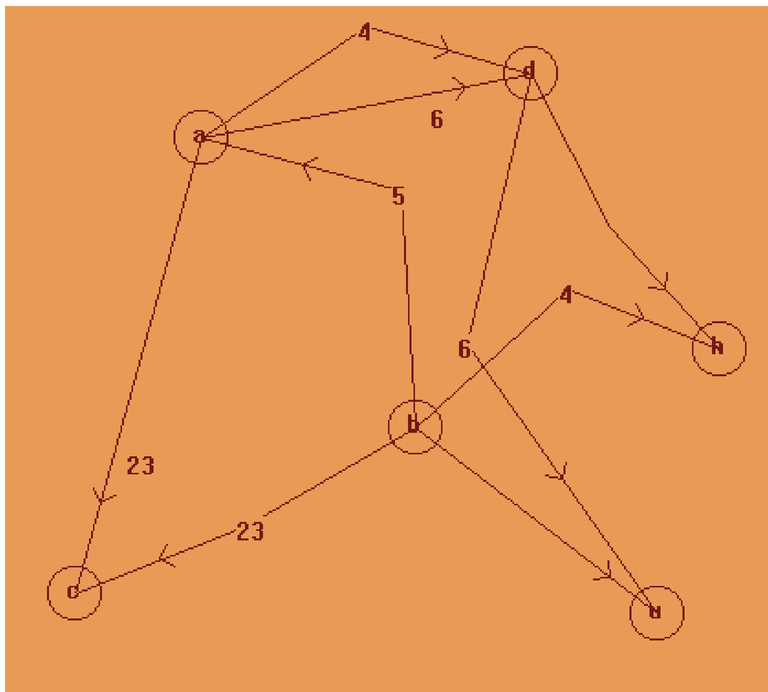
a)vom Knoten a aus alle Pfade.

b)vom Knoten a aus alle minimalen Pfade.

c)vom Knoten a aus die Anzahl der Zielknoten.

d)Löse die Aufgaben a) bis c) auch für alle übrigen Knoten als Startknoten.

Kontrolliere das Ergebnis mit Hilfe von Knotengraph.(durch Programmierung des Verfahrens mit der Entwicklungsumgebung bzw. durch Benutzung des fertigen Programms mittels des Menüs Pfade/Alle Pfade,Minimale Pfade,Anzahl Zielknoten.)



G006.gra

Lösung (nur Knoten a):

a)

a c Summe: 1 Produkt: 1
a d Summe: 4 Produkt: 4
a d h Summe: 5 Produkt: 4
a d u Summe: 10 Produkt: 24
a d Summe: 6 Produkt: 6
a d h Summe: 7 Produkt: 6
a d u Summe: 12 Produkt: 36

b)

a c Summe: 1 Produkt: 1
a d Summe: 4 Produkt: 4
a d h Summe: 5 Produkt: 4

a d u Summe: 10 Produkt: 24

c)Anzahl der Zielknoten: 4

Die Breitensuche

Verbale Beschreibung:

Bei der Breitensuche wird versucht,die am wenigsten vom Startknoten entfernten Knoten zuerst zu besuchen.Daher werden,sobald ein Knoten besucht und markiert wurde,direkt alle Zielknoten der von diesem Knoten wieder auslaufenden Kanten markiert und nacheinander ausgegeben,um danach wiederum die Zielknoten dieser Zielknoten aufzusuchen,zu markieren und auszugeben usw..Um diese letztgenannten Zielknoten erreichen zu können,werden alle besuchten Knoten jeweils in einer Schlange zwischengespeichert,bis die von ihnen ausgehenden Zielknoten wiederum alle besucht und markiert und evtl. in der Schlange ebenfalls gespeichert sind.In der Schlange wird jeweils mit dem in der Reihenfolge nächsten Knoten die im letzten Satz beschriebenen Operation durchgeführt,und der Knoten wird anschließend aus der Schlange entfernt.Bei manueller Durchführung und umfangreicheren Graphen sollte die Schlange schiftlich notiert werden,und das Einfügen bzw. Entfernen der Knoten ist durch wiederholtes Aufschreiben der zu verändernden Schlange zu simulieren.Anschaulich läuft das Verfahren auf ein Schichtenweises Durchlaufen des Graphen hinaus.

Der folgende Algorithmus erzeugt von einem Startknoten ausgehend alle Pfade (unter Berücksichtigung der Kantenrichtungen bei einem gerichteten Graphen) eines breiten Baumdurchlaufs,wobei bei einem ungerichtete Graph stets ein Gerüst des Graphen entsteht,bei einem gerichteten Graphen jedoch nicht alle Knoten vom Startknoten aus durch Wege erreichbar sein müssen.

Algorithmus Alle weiten Baumpfade von einem Knoten aus erzeugen

(Unit UPfad und UGraph)

```
procedure TPfadknoten.ErzeugeWeiteBaumPfade;
var Ob1,Ob2:TObject;
    Index:Integer;
    Kantenliste:TKantenliste;
    Knotenliste:TKnotenliste;
    MomentaneKantenliste:TKantenliste;
    P:TGraph;

  procedure SpeichereNachbarknoten(Kno:TKnoten;Ka:TKante);
  var Hilfliste:TKantenliste;
      Index:Integer;
  begin
    if not Ka.Zielknoten(Kno).besucht then
    begin
      if  (Ka.Quellknoten(Kno)=self) or Ka.Quellknoten(Kno).Pfadliste.Leer
      then
        MomentaneKantenliste:=TKantenliste.Create
      else
        MomentaneKantenliste:=TGraph(Ka.Quellknoten(Kno).Pfadliste.Pfad(0)).
        Kantenliste.Kopie;                                                    4)
      MomentaneKantenliste.AmEndeanfuegen(Ka);
      Ka.Zielknoten(Kno).Pfadliste.AmAnfanganfuegen(MomentaneKantenliste.Kopie.UGraph);
      Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.UGraph);            5)
      Ka.Zielknoten(Kno).Besucht:=true;                                       6)
      Kantenliste.AmAnfanganfuegen(Ka);                                       7)
      Knotenliste.AmAnfanganfuegen(Ka.Zielknoten(Kno));
    end;
end;

begin
  Graph.Pfadlistenloeschen;
  Graph.LoescheKnotenbesucht;
  Besucht:=false;
  Kantenliste:=TKantenliste.Create;                                           1)
  Knotenliste:=TKnotenliste.Create;
  Besucht:=true;
```

```
  P:=TGraph.Create;
  P.Knotenliste.AmEndeanfuegen(self);
  Pfadliste.AmAnfanganfuegen(P);                                              2)
  if not AusgehendeKantenliste.Leer then
    for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
      SpeichereNachbarknoten(self,AusgehendeKantenliste.Kante(Index));        3)
  while not Knotenliste.Leer do
  begin
    Kantenliste.AmEndeloeschen(Ob1);                                          8)
    Knotenliste.AmEndeloeschen(Ob2);
    TKante(Ob1).Pfadrichtung:=TKante(Ob1).Zielknoten(TKnoten(Ob2));           9)
    if not TKnoten(Ob2).AusgehendeKantenliste.Leer then
      for Index:=0 to TKnoten(Ob2).AusgehendeKantenliste.Anzahl-1 do
        SpeichereNachbarknoten(TKante(Ob1).Zielknoten(TKnoten(Ob2)),         10)
    TKnoten(Ob2).AusgehendeKantenliste.Kante(Index));
  end;
  if not AusgehendeKantenliste.Leer then
  begin
    MomentaneKantenliste.Free;
    MomentaneKantenliste:=nil;
  end;
  Kantenliste.Free;
  Kantenliste:=nil;
  Knotenliste.Free;
  Knotenliste:=nil;
end;
```

<u>Erläuterung des Algorithmus:</u>

Nachdem als Initialisierung alle Pfadlisten gelöscht sowie alle Besucht-Markierungen gelöscht und der Startknoten als besucht markiert wurde,wird bei 1) eine Knotenliste und eine Kantenliste erzeugt.

Die Knotenliste wird als Schlange zur Zwischenspeicherung der schon besuchten Knoten benutzt,die Kantenliste speichert zu jedem Knoten der Knotenliste die letzte zu ihm führende Kante.Dies ist zum Setzen der Pfadrichtung bei ungerichteten Kanten von Bedeutung.

In die Pfadliste des Startknoten (self) wird bei 2) zunächst ein nur aus dem Startknoten bestehender Graph eingefügt.Vom Startknoten (self) aus werden dann bei 3) für jede ausgehende Kante und für den zu dieser Kante gehörenden Zielknoten (als die Parameter der Procedure) die Procedure SpeichereNachbarknoten aufgerufen.

In dieser Procedure wird bei 4),falls der jeweilige Zielknoten der Kante noch nicht besucht worden ist (nicht als besucht markiert wurde),entweder eine neue leere MomentaneKantenliste angelegt - bei leerer Pfadliste des Ausgangsknoten - oder aber die MomentaneKantenliste auf den Pfad(0),der in der Pfadliste des Ausgangsknoten gespeichert war,gesetzt.Dieses ist aber der Pfad,der vom Startknoten zum Ausgangsknoten führt.

Zu dieser MomentaneKantenliste wird bei 5) die Kante zum Zielknoten hinzugefügt und die so erweiterte MomentaneKantenliste wird als Pfad(0) in der Pfadliste des Zielknoten gespeichert,so dass damit der Pfad zum Zielknoten in dessen Pfadliste gespeichert wird.Auf diese Weise wird in jedem Knoten des Graphen,der Pfad,der zu ihm führt als (nulltes) Element der Pfadliste gespeichert.

In der Pfadliste des Startknotens wird schließlich bei 5) der momentane Pfad,der zu dem momentan besuchten Knoten führt,am Ende angefügt.

Der Zielknoten wird als markiert besucht (6),und der Knoten und die Kante,die zu ihm führte, werden bei 7) in der Knoten-bzw. Kantenliste am Anfang (Schlange) zwischengespeichert.

Solange wie die Knotenliste noch nicht leer ist,werden bei 8) anschließend aus Knoten-und Kantenliste die Elemente am Ende aus der Liste entfernt (Schlange),die Pfadrichtung bei 9) gemäß der Durchlaufrichtung gesetzt und schließlich bei 10) die Procedure SpeichereNachbarknoten mit der von diesem Knoten ausgehenden Kanten sowie deren Zielknoten erneut aufgerufen.

So erstellt der Algorithmus eine geordnete Pfadliste aus sämtlichen weiten Baumpfad-Graphen im Startknoten und erzeugt jeweils noch die zu den einzel-

nen Knoten führenden Pfade als ein Pfad der Pfadliste in den einzelnen Knoten.

Die Ausgabe der Pfade kann dann (siehe Methode TPfadgraph.AlleweitenBaumpfadevoneinemKnotenbestimmen in der Unit UPfad) in gleicher Weise wie beispielsweise weiter oben bei der Ausgabe der tiefen Baumpfade beschrieben erfolgen.

Falls es nur darum gehen sollte alle Knoten eines weiten Baumes zu anzuzeigen ohne sie zwischenzuspeichern,kann der Algorithmus weitgehend dadurch vereinfacht werden,dass alle Zeilen,die Operationen mit der Kantenliste und der Momentanen Kantenliste betreffen,entfernt werden.Das Speichern der MomentanenKantenliste als Pfad der Pfadliste des Startknotens müßte dann durch die Ausgabe der jeweiligen Knoteninhalte ersetzt werden.

Zur Vorbereitung des Algorithmus sollte das Verfahren mehrfach zuvor per Hand an Hand geeigneter Graphen geübt werden.Zur Kontrolle können dann die Pfade mit Hilfe des Menüs Pfade/Weiter Baum unter Benutzung des Ausgabefenster des Programms Knotengraphs (entweder nach Programmierung des Verfahrens durch die Schüler mittels der Entwicklungsumgebung oder durch Benutzung des fertigen Programms) ausgegeben werden.

C Aufgabe II.4:

Löse Anwendungsaufgabe II.1 für weite Baumpfade und kontrolliere das Ergebnis mit Hilfe von Knotengraph. (durch Programmierung des Verfahrens mit der Entwicklungsumgebung bzw. durch Benutzung des fertigen Programms mittels des Menüs Pfade/Weiter Baum.Bei Verwendung der letzteren Option kann diese Aufgabe auch zur problemorientierten Erarbeitung des Algorithmus mittels des Demo-Modus dienen.)

Beobachtung und problemorientierte Erareitung: Zur problemorientierten Erarbeitung des Verfahrens sollte die beiden Verfahren tiefer und weiter Baumdurchlauf an Hand des Graphen der Aufgabe II.1 mittels des Demo-Modus des Progamm Knotengraphs von den Schülern miteinander verglichen werden.Wie die unten stehende Lösung zeigt,wird der Pfad a d beim breiten Baumdurchlauf erst nach Ausgabe aller andern Pfade mit zwei Elementen weiter zu a d e ergänzt.Dies gibt Anlaß eine Schlange zur Zwischenspeicherung der Pfade a d,a f,a b,a c zu benutzen.Durch geeignete weitere Ergänzung des Graphen kann dann von den Schülern beobachtet werden,dass auch die übrigen Pfade in der Reihenfolge dieser Schlange erweitert werden.Von den Schülern sollte zumindestens das Schichtenweise Durchlaufen des Graphen bemerkt werden.

Lösung:

Ergebnis für den Graph G005.gra von Aufgabe I.5 (Startknoten a):

a Summe: 0.0 Produkt: 0.0
a d Summe: 1.0 Produkt: 1.0
a f Summe: 1.0 Produkt: 1.0
a b Summe: 1.0 Produkt: 1.0
a c Summe: 1.0 Produkt: 1.0
a d e Summe: 2.0 Produkt: 1.0

Mit Hilfe der Algorithmen zur Erzeugung der tiefen und der weiten Baumpfade,lassen sich jeweils verschiedenste Gerüste eines vorgegebenen Graphen erzeugen.Falls die Kanten des Graphen bewertet sein sollten,stellt sich die Frage,welcher dieser Bäume ein Baum mit der kleinsten Kantensumme ist.

C Definition II.1:

Ein Gerüst mit der kleinsten Kantensumme innerhalb eines bewerteten Graphen heißt minimal spannender Baum.

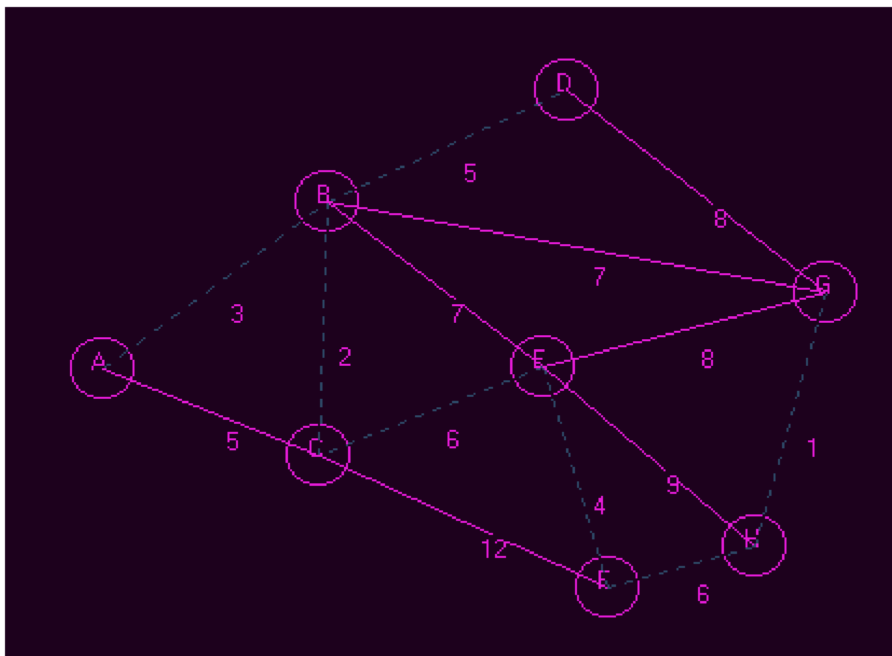
C Aufgabe II.5: (Einstiegsproblem)

In einem Stadtviertel sollen Telefonleitungen gelegt werden.Die Häuser werden durch die Knoten und die Straßen durch die Kanten des folgenden Graphs symbolisiert.Jedes Haus soll dabei mit jedem verbunden werden.Es genügt zwischen zwei Häusern eine Verbindungsleitung.Wie müssen die Leitungen gelegt werden,damit die Gesamtleitungslänge minimal ist?

a)Löse die Aufgabe durch Probieren.

b)Löse die Aufgabe mittels des Programms Knotengraph Menü Pfade/Minmales Gerüst des Graphen im Demo-Modus und versuche das Prinzip des Algorithmus zu erkennen.

Lösung:



G007.gra

a)**Kanten: 1 2 3 4 5 6 6 Summe: 27**

Markierter Graph: minimal spannender Baum mit Kantensumme:27

b)**Beobachtung und problemorientierte Erarbeitung:**Die Kanten werden in der Reihenfolge aufsteigender Werte ausgewählt,angefangen mit dem kleinsten Wert.Die noch nicht rot markierten Kanten werden zwar alle untersucht,wie sich an der grünen Markierung zeigt,sie werden jedoch nicht ausgewählt.Wie ein Blick auf den Graph zeigt,würde sich bei Auswahl einer dieser Kanten,der ausgewählte Graph Kreise erhalten und seine Eigenschaft als Baum verlieren.

Das Verfahren läßt sich zu folgendem Algorithmus präzisieren:

Algorithmus von Kruskal

Verbale Beschreibung:

Das Prinzip des Algorithmus besteht in folgenden Schritten:

1)Suche unter den Kanten des gegebenen Graphen G_1 eine Kante a mit minimalen Wert aus,wobei a keine Schlinge ist.Erzeuge aus a einen neuen Graphen G_2.Lösche die Kante a im Graphen G_1.

2)Wähle im Restgraphen G_1 eine weitere minimale Kante a' aus und füge sie in den Graphen G_2 genau dann ein,wenn der um a' erweiterte Graph G_2 keine Kreise enthält.Lösche die Kante a' im Graph G_1.

Wiederhole Schritt 2) solange,bis der Graph G_1 leer ist.Dann ist G_2 ein minimal spannender Baum von G_1.

C Satz II.1:

Es sei G ein schlichter,zusammenhängender und ungerichteter Graph,in dem alle Kanten eine verschiedene nichtnegative Kantenbewertung haben.Dann bestimmt der Algorithmus von Kruskal den (eindeutig bestimmten) minimal spannenden Baum.

Beweis:

Der Algorithmus von Kruskal bestimmt nach Konstruktion sicher ein Gerüst des Graphen G,da der durch die Kanten erzeugte Untergraph keine Kreise aber alle Knoten enthält.

$M_1=\{a_1,a_2,...a_n\}$ sei die nach dem Kruskal Algorithmus ermittelte Kantenmenge,wobei die Kanten a_1 bis a_n dem Kantenwert entsprechend aufsteigend geordnet sind. $M_2=\{j_1,j_2,...j_n\}$ sei das minimal spannende Gerüst (mit kleinster Kantenbewertung) von G.Die Kanten j_1 bis j_n seien ebenfalls dem Wert nach aufsteigend geordnet.

Annahme:$M_1 \neq M_2$

Dann gibt es einen kleinsten Index c mit $j_c \neq a_c$.

In den Graph M_2 wird die Kante a_c eingefügt.Da M_2 ein Gerüst ist,entsteht in dem neuen Graph M_3 mit n+1 Kanten, der die Kante a_c zusätzlich enthält,ein Kreis,der die Kante a_c einschließt.In dem Kreis können nicht alle Kanten aus M_1 sein,da M_1 ein Gerüst ist und daher keinen Kreis enthält und deshalb enthält der Kreis mindestens eine Kante aus M_2.Eine solche Kante in dem Kreis sei $j_s \neq a_c$. Diese Kante läßt sich nun wieder aus dem Kreis entfernen,so daß der nun so entstehende Graph M_3 wiederum ein Gerüst ist.

Also ist M_3 ein Gerüst,das statt der Kante j_s die Kante a_c enthält.Da j_s nicht aus aus M_1 ist und a_c die erste Kante von M_1 ist,die nicht zu M_2 gehört,ist der Wert der Kante a_c kleiner als der von j_s .(denn M_1 enthält ja gerade nach Konstruktion der Reihenfolge nach die Kanten mit dem kleinsten Kantenwert, und a_c ist dann die nächstgrößere Kante, und alle Kanten sind verschieden bewertet.)Also ist die Gesamtkantenlänge von M_3 kleiner als die von M_2.

Dies ist ein Widerspruch zu der Tatsache,dass M_2 das (kleinste) minimal spannende Gerüst ist.

Also ist $M_1=M_2$.

Daher liefert der Algorithmus von Kruskal schon das (kleinste) minimal spannende Gerüst.Obwohl das Suchen eines Gerüstes kein Pfadalgorithmus ist,wird der Quelltext trotzdem aus didaktischen Gründen wegen des Zusammenhangs mit den Baumalgorithmen in der Unit Pfad platziert und wird auch unter dem Menü Pfade aufgerufen.Die folgende Methode Kruskal arbeitet nach dem Algorithmus von Kruskal:

Quellcode:

(Unit UPfad)

```
procedure TPfadgraph.Kruskal(Ausgabe:TLabel;var SListe:TStringList;Flaeche:TCanvas);
var T:TInhaltsgraph;
    Knoa,Knoe:TInhaltsknoten;
    ListederKanten:TKantenliste;
    Ka,Kb:TInhaltskante;
    Ob:TObject;
begin
  if not Leer  then
  begin
    T:=TInhaltsgraph.Create;
    ListederKanten:=Kantenliste.Kopie;                                          1)
    ListederKanten.Sortieren(Groesser,Bewertung);                               2)
    while not ListederKanten.Leer do
    begin
      Ka:=TInhaltskante(ListederKanten.Kante(0));                               3)
      Knoa:=TInhaltsknoten.Create;                                                 4)
      Knoe:=TInhaltsknoten.Create;
      Knoa.X:=TInhaltsknoten(Ka.Anfangsknoten).X;
      Knoa.Y:=TInhaltsknoten(Ka.Anfangsknoten).Y;
      Knoe.X:=TInhaltsknoten(Ka.Endknoten).X;
      Knoe.Y:=TInhaltsknoten(Ka.Endknoten).Y;
      Knoa.Wert:=Ka.Anfangsknoten.Wert;
      Knoe.Wert:=Ka.Endknoten.Wert;
      Kb:=TInhaltskante.Create;
      Kb.Wert:=Ka.Wert;
      Kb.Anfangsknoten:=Knoa;
      Kb.Endknoten:=Knoe;
      Kb.Weite:=Ka.Weite;
      Kb.Typ:=Ka.Typ;
      Kb.Gerichtet:=false;
      if not Ka.KanteistSchlinge
      then
          T.EinfuegenKante(Kb);                                                 5)
      if T.GraphhatKreise
      then
        T.LoescheInhaltskante(Kb)                                               6)
      else
      begin
        Ausgabe.Caption:='GerüstKante: '+Ka.Wert;                               7)
        if not Ka.KanteistSchlinge then
        begin
          Ka.Farbe:=clred;
          Ka.Stil:=psdot;
          Ka.ZeichneKante(Flaeche);
        end;
      end;
      if not ListederKanten.Leer then ListederKanten.AmAnfangLoeschen(Ob);      8)
      if not ListederKanten.Leer then  ListederKanten.Sortieren(Groesser,Bewertung);     9)
    end;
    Ausgabe.Caption:='Kanten: '+                                               10)
    T.InhaltallerKantenoderKnoten(ErzeugeKantenstring)+
    ' Summe: '+
    RundeZahltostring(T.Kantensumme(Bewertung),Kantengenauigkeit);
    SListe.Add(Ausgabe.Caption);                                               11)
    Ausgabe.Refresh;
    T.Freeall;
    T:=nil;
  end;
end;
```

<u>Erläuterung des Algorithmus:</u>

Es wird zunächst bei 1) eine Kopie der Kantenliste des Graphen in der Variablen ListederKanten gespeichert.Diese Kantenliste wird bei 2) gemäß der Größe der Kantenwerte aufsteigend sortiert.

Es wird dann die Kante mit dem kleinsten Kantenwert bei 3) auf der Variablen Ka abgespeichert.

Danach wird bei 4) eine Kopie Kb der Kante Ka erzeugt,die von Ka alle wichtigen Daten,wie Kanteninhalt, Kantenweite,Inhalte und Koordinaten von Anfangs-und Endknoten sowie die Information,ob es sich um eine gerichtete Kante handelt oder nicht,enthält.Diese Kante wird bei 5) in den neu erzeugten Graph T,der den minimal spannenden Baum aufnimmt,eingefügt.

Falls T durch das Einfügen von der Kantenkopie einen Kreis enthält,wird die Kante bei 6) aus T wieder gelöscht.Andernfalls wird die Kante bei 7) rot markiert gezeichnet und im Label Ausgabe ausgegeben.

Bei 8) wird dann die Kante Ka aus der ListederKanten gelöscht.Schließlich wird bei 9) die Liste erneut sortiert,so daß nunmehr die nächstgrößere Kante an erster Stelle dieser Liste steht.

Die Programmabschnitte 3) bis 9) werden solange wiederholt,bis die ListederKanten leer ist.

Bei 10) und 11) wird schließlich noch eine (String-)Liste der Kanteninhalte der in T eingefügten Kanten erstellt und im Label Ausgabe ausgegeben als auch als Element der Stringliste SListe gespeichert.(Diese wird für das Ausgabefenster benötigt.)

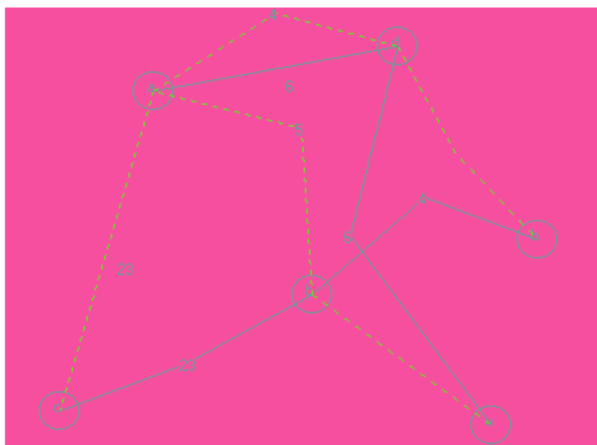
Der Abschnitt 10) bis 11) ist eventuell entbehrlich.

Insgesamt erzeugt der Algorithmus einen minimal spannenden Baum zum vorgegebenen Graphen im Graph T und zeichnet ihn rot markiert.

C Aufgabe II.6:

a)Wende den Algorithmus von Kruskal auf den Graphen von Aufgabe II.3 (Graph **G006.gra**) an.(Je nach Konzeption per Hand als Übung,durch eigenen Quellcode oder mittels des fertigen Programms Knotengraph.Außerdem können die Aufgaben a und b auch noch als Einstiegsaufgaben mittels des Demo-Modus dienen.)

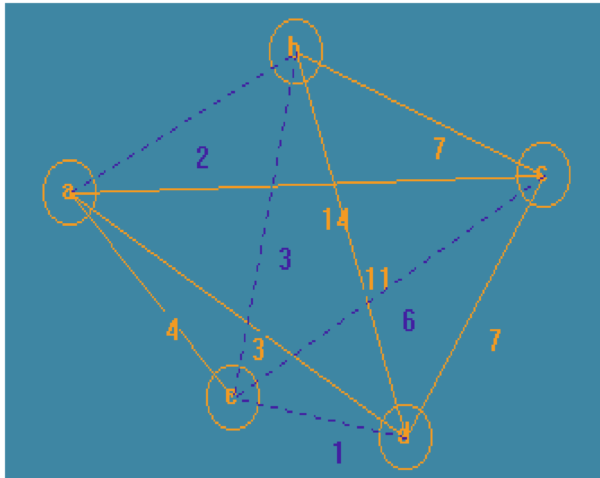
Lösung:



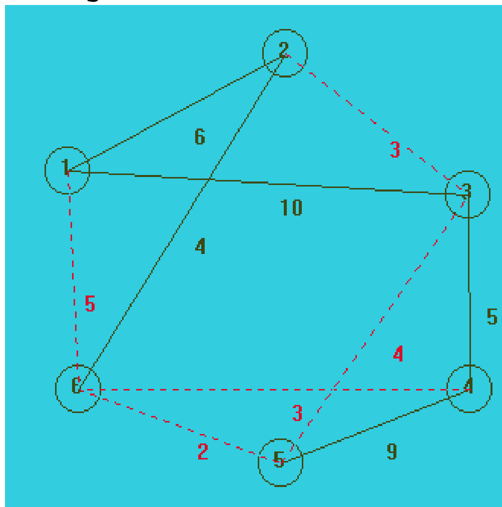
G006.gra

b)Wende den Algorithmus von Kruskal auf folgende Graphen an (als Übung per Hand oder mit Hilfe des selber programmierten Algorithmus bzw. mit Hilfe des Menüs Pfad/Minimales Gerüst des Programm Knotengraphs) und erzeuge jeweils einen minimal spannenden Baum:

Graphen und Lösungen:



G008.gra



G009.gra

Die Lösungen sind jeweils markiert dargestellt.

Inorder-Durchlauf und Binäres Suchen

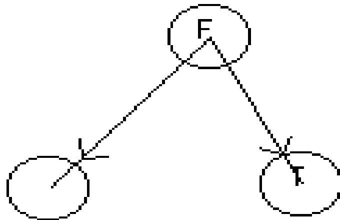
Schließlich sollen noch Algorithmen für den im vorigen Kapitel besprochenen Inorder-Durchlauf und für das Binäre Suchen vorgestellt werden,die beide nur für Graphen,die als Binärbäume aufgefaßt werden können,sinnvoll sind.Falls ein Graph diese Eigenschaft besitzt,wird die Ausführung beider Algorithmen als Option bei den Menüs Alle Pfade und Abstand zwischen zwei Knoten des Programm Knotengraphs automatisch angeboten(vgl. Quelltext im Anhang).Durch diese Algorithmen wird gezeigt,wie der Anschluss an konventionelle Lerninhalte hergestellt wird.

Ein Graph wird als Binärbaum gemäß der Methode GraphistBinärbaum (siehe Quelltext im Anhang) erkannt,wenn jeder Knoten nicht mehr als zwei ausgehende Kanten hat.Jeder Knoten hat maximal eine eingehende Kante.(Die Wurzel hat keine eingehenden Kanten.)Alle Kanten sind gerichtet.Der Graph enthält keine Kreise.

Die erste in die ausgehende Kantenliste eingefügte Kante (Index: 0) zeigt auf den linken Teilbaum,und auf den rechten Teilbaum zeigt die zuletzt

eingefügte Kante (Index 1).Für die Erzeugung eines diesen Regeln entsprechenden Graphen sowie für das Vorliegen der Knotenwert-Ordnung im geordneten Binärbaum ist der Benutzer bzw. Erzeuger des Graphen verantwortlich.

Hat ein Knoten nur eine ausgehende Kante,ist dies automatisch der linke Teilbaum dieses Knotens.Will man ereichen,dass ein Knoten lediglich einen einzelnen rechten Teilbaum besitzt,ist es notwendig einen Dummy-Knoten mit leerem Inhalt als ersten Knoten in die Kantenliste einzufügen.



Linker Teilbaum:Dummy-Knoten

Verbale Beschreibung:

Der Inorder-Durchlauf erfolgt nach dem bekannten Schema:

Algorithmus Inorder:

1)Durchlaufe linken Teilbaum

2)Durchlaufe Wurzelknoten des Teilbaums

3)Durchlaufe rechten Teilbaum

(falls die Teilbäume nicht leer sind)

(Problemorientierte Einstiegsaufgabe siehe Aufgabe C I.3)

Quellcode:

```
procedure TPfadknoten.BinaererBaumInorder;
label Endproc;
var MomentaneKantenliste:TKantenliste;
    P:TGraph;
    Ob:TObject;

  procedure GehezuNachbarknoten(Kn:TKnoten;Ka:TKante);
  Label Endproc;
  var Ob:TObject;
  begin
    if Ka.Zielknoten(Kn).AusgehendeKantenliste.Anzahl>0
    then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kn);
      MomentaneKantenliste.AmEndeanfuegen(Ka);                                  4)
      GehezuNachbarknoten(Ka.Zielknoten(Kn),Ka.Zielknoten(Kn).AusgehendeKantenliste.Kante(0));
      MomentaneKantenliste.AmEndeloeschen(Ob);
    end;
    Ka.Pfadrichtung:=Ka.Zielknoten(Kn);
    MomentaneKantenliste.AmEndeanfuegen(Ka);
    if Ka.Zielknoten(Kn).Wert<>'' then
    Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);                 5)
    MomentaneKantenliste.AmEndeloeschen(Ob);
    if Ka.Zielknoten(Kn).AusgehendeKantenliste.Anzahl>1 then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kn);
      MomentaneKantenliste.AmEndeanfuegen(Ka);                                  6)
      GehezuNachbarknoten(Ka.Zielknoten(Kn),Ka.Zielknoten(Kn).AusgehendeKantenliste.Kante(1));
      MomentaneKantenliste.AmEndeloeschen(Ob);
    end;
```

```
  Endproc:
  end;
begin
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.Pfadlistenloeschen;
  if AusgehendeKantenliste.Anzahl>0
  then
    GehezuNachbarknoten(self,AusgehendeKantenliste.
      Kante(0));                                                          1)
    P:=TGraph.Create;
    P.Knotenliste.AmEndeanfuegen(self);
    {P.Kantengenauigkeit:=TInhaltsgraph((self).Graph).Kantengenauigkeit;
    P.Knoteneinfuegen(self);}
    Pfadliste.AmEndeanfuegen(P);                                          2)
  if AusgehendeKantenliste.Anzahl>1
  then
    GehezuNachbarknoten(self,AusgehendeKantenliste.Kante(1));             3)
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
end;
```

Erläuterung des Algorithmus:

Bei 1) und 4) wird jeweils der linke Teilbaum mittels der rekursiven Procedure GehezuNachbarknoten besucht und bei 3) und 6) entsprechend der rechte Teilbaum,falls die Kanten mit dem Index 0 bzw. 1 existieren.

Bei 2) und 5) wird der Wurzelknoten bzw. der aktuelle Pfad einschließlich der aktuellen Kante zum momentan besuchten Knoten in die Pfadliste des Pfadknotens eingefügt und steht damit als Pfad zur Ausgabe und Anzeige bereit.

Bei jedem Rückwärts-Gehen im Baum werden die Kanten wieder aus der Kantenliste gelöscht.

Algorithmus Binäres Suchen

Verbale Beschreibung:

Der Algorithmus Binäres Suchen folgt dem bekannten Schema:

1)Gegeben sei ein Vergleichswert.

2)Starte bei der Wurzel.

3)Ist der Vergleichswert kleiner als der Knotenwert des momentanen Knotens besuche den linken Teilbaum.

4)Ist der Vergleichswert größer als der Knotenwert des momentanen Knotens besuche den rechten Teilbaum.

5)Ist der Vergleichswert gleich dem Wert des momentanen Knotens,dann beende den Algorithmus.Der Zielknoten ist gefunden.Gib den bisher durchlaufenden Pfad aus.

6)Ist der momentane Knoten ein Blatt und ergibt sich keine Wertübereinstimmung, ist der Algorithmus beendet.Dann gibt es keinen gesuchten Zielknoten im Baum.

(Problemorientierte Einstiegsaufgabe siehe Aufgabe C I.3)

Die folgende Methode entscheidet jeweils selber,welches der Teilbaum mit dem kleineren oder größeren Knotenwert ist.Sie setzt, wie gesagt, aber voraus,dass der Binärbaum geordnet ist.

Quellcode:

```
procedure TPfadknoten.BinaeresSuchen(Kno:TKnoten);
var Index:Integer;
    MomentaneKantenliste:TKantenliste;
```

```
    Gefunden:Boolean;
    P:TGraph;

  procedure GehezuNachbarknoten(Kn:TKnoten;Ka:TKante);
  label Endproc;
  var Ob:TObject;
      Index:Integer;
  begin
    Ka.Pfadrichtung:=Ka.Zielknoten(Kn);
    MomentaneKantenliste.AmEndeanfuegen(Ka);
    Kno.Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
    if Ka.Zielknoten(Kn).Wert=Kno.Wert                                   5)
    then
    begin
      Gefunden:=true;
      Showmessage('Knoten gefunden!');
    end;
    if Ka.Zielknoten(Kn).AusgehendeKantenliste.Anzahl=2
    then
    if (Ka.Zielknoten(Kn).Wert<=Kno.Wert) xor
      (Ka.Zielknoten(Kn).AusgehendeKantenliste.Kante(0).Zielknoten(Ka.Zielknoten(Kn)).Wert<=
       Ka.Zielknoten(Kn).AusgehendeKantenliste.Kante(1).Zielknoten(Ka.Zielknoten(Kn)).Wert)
    then                                                                 6)
      GehezuNachbarknoten(Ka.Zielknoten(Kn),Ka.Zielknoten(Kn).AusgehendeKantenliste.Kante(0))
    else
      GehezuNachbarknoten(Ka.Zielknoten(Kn),Ka.Zielknoten(Kn).AusgehendeKantenliste.Kante(1));
    if Ka.Zielknoten(Kn).AusgehendeKantenliste.Anzahl=1
    then                                                                 7)
      GehezuNachbarknoten(Ka.Zielknoten(Kn),Ka.Zielknoten(Kn).AusgehendeKantenliste.Kante(0));
    MomentaneKantenliste.AmEndeloeschen(Ob);
  Endproc:
  end;

begin
  MomentaneKantenliste:=TKantenliste.Create;
  Gefunden:=false;
  Graph.Pfadlistenloeschen;
  if Wert=Kno.Wert then                                                  1)
  begin
    P:=TGraph.Create;
    P.Knotenliste.AmEndeanfuegen(self);
    {P.Kantengenauigkeit:=TInhaltsgraph((self).Graph).Kantengenauigkeit;
    P.Knoteneinfuegen(self);}
    Pfadliste.AmAnfanganfuegen(P);
    Showmessage('Knoten gefunden!');
  end;
  if AusgehendeKantenliste.Anzahl=2
  then
    if (Wert<=Kno.Wert) xor
       (AusgehendeKantenliste.Kante(0).Wert<=AusgehendeKantenliste.Kante(1).Wert)
    then
      GehezuNachbarknoten(self,AusgehendeKantenliste.Kante(0))           2)
    else
      GehezuNachbarknoten(self,AusgehendeKantenliste.Kante(1));          3)
  if AusgehendeKantenliste.Anzahl=1
  then
    GehezuNachbarknoten(self,AusgehendeKantenliste.Kante(0));            2)
  if not Gefunden then Graph.Pfadlistemloeschen;
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
end;
```

Erläuterung des Algorithmus:

Bei 1) und 5) wird jeweils überprüft,ob der Vergleichswert des Knotens Kno gleich dem Wert des momentanen Knotens ist.Dann wird der Pfad zu diesem Knoten in die Pfadliste eingefügt.

Bei 2) und 6) wird jeweils der kleinere (linke?) Wert-Teilbaum besucht und bei 3) und 7) der größere (rechte?) Wert-Teilbaum,je nachdem wie der Vergleich mit dem momentanen Knotenwert ausgefallen ist.

Der Fall,dass eventuell nur ein Kind-Knoten vorhanden ist,wird entsprechend berücksichtigt. (AusgehendeKantenliste.Anzahl=1)

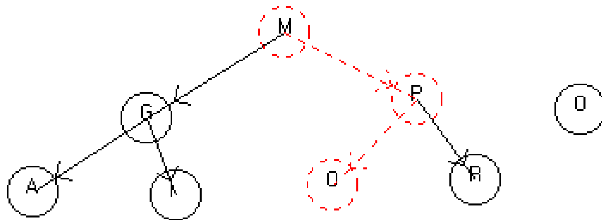
Das Binäre Suchen erzeugt in dem Spezialfall Binärbaum einen minimalen Pfad zwischen dem Start-bzw. Wurzelknoten und dem Zielknoten (falls existent).

Im nächsten Kapitel soll jetzt diese Aufgabe in allgemeinen Graphen mittels des Algorithmus von Kruskal gelöst werden.

C Aufgabe II.7:

Erzeuge beim Graph G0004c.gra ein weiteren Knoten mit dem Inhalt O und lasse diesen Knoteninhalt im Baum binär suchen (mittels des eigens erstellten Algorithmus der Entwicklungsumgebung oder mit Hilfe des Menüs Pfade/Abstand von zwei Knoten/Binäres Suchen des Programm Knotengraphs)

Lösung:



Bemerkung:Der gesuchte Knoteninhalt kann also durch einen Knoten außerhalb des Baumes vorgegegeben werden

C III Alle Pfade sowie minimaler Pfad zwischen zwei Knoten

Ein sehr effizient arbeitender Algorithmus zur Bestimmung der minimalen Pfaden von einem Startknoten aus zu den übrigen Knoten des Graphen ist der Algorithmus von Dijkstra.

Er sollte als Verbesserung des im vorigen Kapitel beschriebenen Verfahrens minimale Pfade,das sich aus dem tiefen Baumdurchlauf herleitetete,unbedingt besprochen werden.(Es ist aber nicht mit diesem Verfahren didaktisch verwandt.)

Durch geringfügige Änderungen kann daraus dann ein Algorithmus zur Bestimmung des **minimale**n Pfads zwischen zwei Knoten des Graphen gewonnen werden.

Die Bestimmung **aller** Pfade zwischen zwei Knoten leitet sich nun aber wiederum vom TiefenBaumdurchlauf ab und wird mit Hilfe des im vorigen Kapitels schon beschriebenen Verfahrens durchgeführt.

Das vorliegende Kapitel ist also als Weiterführung des Unterrichtsganges des vorigen Kapitels aufzufassen und bildet mit ihm zusammen eine komplette Unterrichtsreihe mit dem Titel Pfade in Graphen.

Der Quellcode der Algorithmen ist wiederum in der Unit UPfad zu finden.Der Einstieg in die Problemstellungen kann an Hand des folgenden Einstiegsproblems (Rätsel oder Denksportaufgabe) Umfüllproblem erfolgen,in der sich zwanglos beide Problemstellungen,nämlich das Suchen aller Pfade zwischen zwei Knoten (Umfüllzuständen) als auch das Suchen des minimalen Pfades stellen.

C Aufgabe III.1: (Einstiegsproblem)

Drei Krüge A,B und C fassen 3 l,3 l und 1 l.Nur A ist anfangs mit 3 l gefüllt.B und C sind leer.Durch Umschütten soll erreicht werden,dass am Schluß in A 1 l und in B 2 l vorhanden sind.C ist leer.

Wie muß jeweils umgeschüttet werden,und welches ist der Weg mit der kleinsten Anzahl Umschüttungen?

Wieviel mögliche Lösungen gibt es insgesamt?Lösung zunächst durch Probieren.Codiere dann die Zustände durch Knoten und erstelle dann einen geeigneten Graphen zu dem Problem.

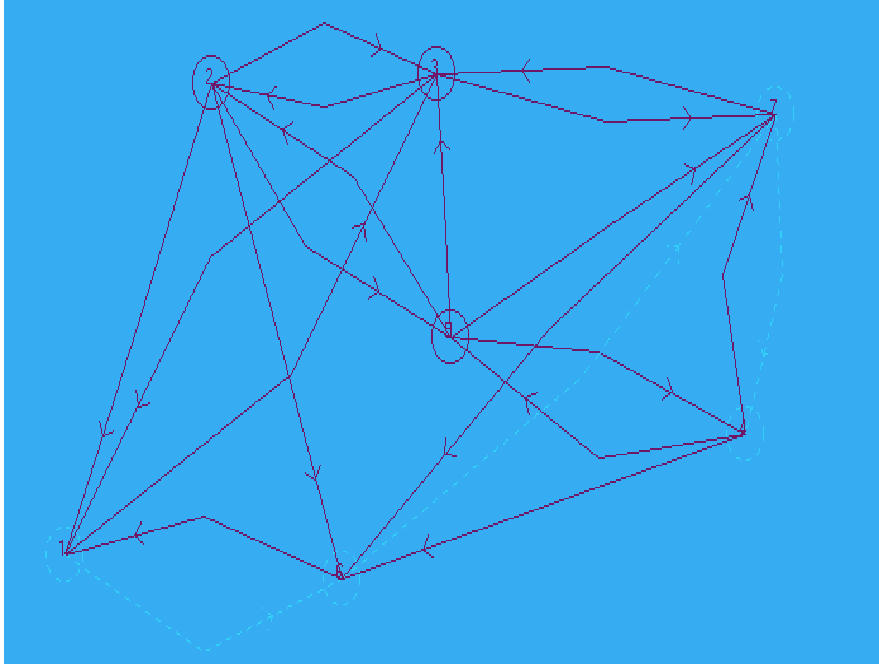
(Benutze nach der manuellen Lösung das Programm Knotengraph Menü Pfade/Abstand von zwei Knoten und Alle Pfade zwischen zwei Knoten zur Ermittlung der Möglichkeiten.)

Lösung:

Aus den nachfolgenden Zuständen können jeweils folgende weitere Zustände durch einmaliges Umschütten erreicht werden:

Zustand:	Krug A:	Krug B:	Krug C:	Übergang zu:
1)	3	0	0	3),6)
2)	2	1	0	1),3),5),6)
3)	2	0	1	1),2),7)
4)	1	2	0	5),6),7)
5)	1	1	1	2),3),4),7)
6)	0	3	0	1),7)
7)	0	2	1	3),4),6)

Darstellung als Graph:



G010.gra

Lösung:

1 6 7 4 Summe: 3

Knotenfolge: 1 6 7 4

Der Weg mit der kleinsten Anzahl von Umschüttungen ist der kürzeste (minimalste) Pfad zwischen den Knoten 1 und 4.

(eine andere Lösung ist die Knotenfolge:1 3 7 4)

Zur Bestimmung aller Lösungen sind alle Pfade zwischen den Knoten 1 und 4 zu suchen:

1 3 7 4 Summe: 3.000 Produkt: 1.000
1 3 2 5 7 4 Summe: 5.000 Produkt: 1.000
1 3 2 5 4 Summe: 4.000 Produkt: 1.000
1 3 2 6 7 4 Summe: 5.000 Produkt: 1.000
1 6 7 4 Summe: 3.000 Produkt: 1.000
1 6 7 3 2 5 4 Summe: 6.000 Produkt: 1.000

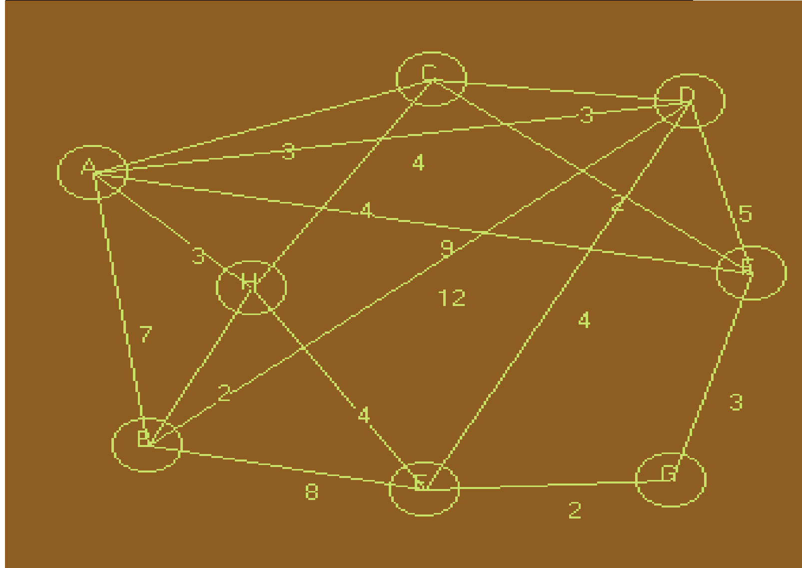
(Lösungen mittels des Programms Knotengraph)

Die Aufgabe erfordert die Bestimmung des kleinsten (minimalsten) Pfades zwischen zwei Knoten sowie die Bestimmung von allen Pfaden zwischen zwei Knoten.Diese Aufgaben sind auch für viele andere Anwendungen von Bedeutung,z.B. wenn es gilt eine kürzeste Fahrstrecke oder alle Fahrstrecken zwischen zwei Städten zu finden.

Zusatzaufgabe: Benutze den Demo-oder Einzelschrittmodus zum Erkennen des Algorithmus Abstand von zwei Knoten.(Der Algorithmus Alle Pfade zwischen zwei Knoten wurde schon im letzten Kapitel II besprochen,vgl. Aufgabe C II.1 e)

Beobachtung und problemorientierte Erarbeitung: Nacheinander werden alle Pfade der Länge 1,2... usw. vom Startknoten aus erzeugt (grüne Markierung),bis bei der Länge 3 ein Pfad zum Zielknoten führt (rote Markierung).

C Aufgabe III.2 (Einstiegsproblem 2. Teil):



G011.gra

Gegeben ist das obige Straßennetz zwischen den Orten A bis H.Ermittle durch das Menü Pfade/Abstand von zwei Knoten mittels des Demo- oder Einzelschrittmodus den minimalen Pfad zwischen Ort B und F (in dem obigen Graph mit Bewertung).Beobachte die Funktionsweise des Algorithmus.

Beobachtung und problemorientierte Erarbeitung: Es werden nacheinander Pfade (grün markiert) mit den folgenden Längen untersucht:BH 2,BHA 5,BHE 6,BHC 6,BA 7,BHEG 8,BHCF 8 (Lösung).Also werden jeweils nacheinander die Pfade in aufsteigender **Länge** vom Startknoten aus und nicht in aufsteigender **Kantenzahl** gewählt.

Das Verfahren wird zu folgenden Algorithmus präzisiert:

Algorithmus Kürzeste Wege zwischen Knoten nach Dijkstra

Vorausgesetzt wird ein Pfad mit nichtnegativer Kantenbewertung.Der Algorithmus von Dijkstra beruht darauf,vom Startknoten aus jeweils schrittweise Kante um Kante auf den Zielknoten vorzurücken und bei jedem Schritt jeweils die Kante zu einem Nachbarknoten von einem der schon erreichten Knoten zu dem Pfad,der zu diesem Knoten führt,hinzuzufügen,so dass der so entstandene erweiterte Pfad die minimalste Pfadlänge unter allen Pfaden,die vom Startknoten aus zu Nachbarknoten der momentan erreichten Knotenmenge aus führen,hat.Die schon momentan aufgesuchten Pfade müssen also zwischengespeichert werden,um den neuen minmalen Pfad auszuwählen.Dies geschieht am besten in einer Schlange,die nach Pfadlängen sortiert wird.

Im folgenden werden zwei Fälle berücksichtigt.

a)Es ist ein Zielknoten vorgegeben,zu dem der minimale Pfad gesucht wird.

oder:

b)Es sollen die minimalen Pfade zu allen Knoten des Graphen vom Startknoten aus gefunden und anschließend gespeichert oder gezeichnet werden.

Der Algorithmus in einzelnen Schritten:

Verbale Beschreibung:

1)Wähle den vorgegeben Knoten als Startknoten.

2)Speichere alle vom Startknoten ausgehenden Kanten als Pfade in einer Pfadliste und sortiere sie absteigend nach Pfadlänge.

3)Ermittle den Pfad p mit kleinster Pfadlänge in der Pfadliste (der letzte Pfad).Wenn a) der minimale Pfad nur zu einem Zielknoten gefunden werden soll,dann überprüfe,ob der Endknoten dieses Pfades schon der Zielknoten ist.Wenn er der Zielknoten ist,ist der gesuchte minimale Pfad gefunden.Dann beende den Algorithmus.Wenn b) die minimalen Pfade zu allen Knoten gefunden werden sollen,dann speichere diesen Pfad oder zeichne ihn.(Bei Gleichheit der Pfadlänge ist im Fall a) jeweils irgendein Pfad zu wählen,dessen Endknoten gleich dem Zielknoten ist und bei b) ist irgendeiner der Pfade zu zeichnen bzw. zu speichern.Dieser Pfad ist dann der Pfad p.)

4)Markiere den Endknoten dieses Pfades als besucht,wenn er noch nicht markiert ist.Wenn a) der Endknoten nicht der Zielknoten ist bzw. wenn b) die minimalen Pfade zu allen Knoten gefunden werden sollen,und es außerdem Nachbarknoten dieses Endknotens gibt,erzeuge neue längere Pfade p_i,indem dem Pfad zu diesem Endknoten jeweils nacheinander alle von diesem Endknoten ausgehenden Kanten zu den noch nicht besuchten Nachbarknoten hinzugefügt werden,und auf diese Weise neue Pfade p_i vom Startknoten aus zu diesen Nachbarknoten entstehen (i ist die Nummerierung der Pfade gemäß der Ordnung der vom Endknoten ausgehenden Kanten).Lösche den Pfad p aus der Pfadliste.

5)Füge diese neuen Pfade p_i (am Anfang) in die Pfadliste ein,und sortiere die Pfadliste absteigend.

6)Solange noch immer Nachbarknoten des Endknotens des minimalen Pfades der Pfadliste des Graphen nicht als besucht markiert sind und die Pfadliste nicht leer ist,setze den Algorithmus bei Schritt 3) fort.

Ansonsten beende den Algorithmus.

Die Pfadliste enthält nach Konstruktion zu jedem Augenblick alle Pfade zu den Nachbarknoten als besucht markierter Knoten.Denn zu Beginn enthält die Pfadliste alle Pfade,die aus den Kanten des markierten Startknoten zu den Nachbarknoten führen.

Sobald ein Pfad aus der Pfadliste entfernt wird,werden alle Pfade die vom Startknoten zu den noch nicht als besucht markierten Nachbarknoten dieses Knotens führen,in die Pfadliste aufgenommen.

Die Pfadliste wird nach jedem Einfügen von Pfaden absteigend sortiert,so dass sie stets an letzter Position den minimalen Pfad zu allen noch nicht besuchten Nachbarknoten enthält.

Falls der Nachbarknoten dann der Zielknoten ist,ist bei a) der gesuchte minimale Pfad gefunden.

Ansonsten wird bei leerer Pfadliste bei a) ein leerer Pfad als Rückgabewert der Function zurückgegeben.

C Satz III.1:

Der Algorithmus von Dijkstra liefert nacheinander bei einem zusammenhängenden Graphen mit nichtnegativer Kantenbewertung den Pfad minimalster Pfadlänge zwischen Startknoten zu jedem anderen Knoten des Graphen (in dem Augenblick,in dem dieser Knoten als besucht markiert wird),wenn der andere Knoten vom Startknoten aus durch einen Pfad erreichbar ist.

Behauptung:

Es seien schon j Knoten als besucht markiert.B_j sei die Menge dieser Knoten.M_j sei die Menge der noch nicht als besucht markierten Knoten des Graphen:

Mit l sei die kleinste Pfadlänge eines Pfades zwischen zwei Knoten bezeichnet.Dann gilt für alle Knoten k_x aus B_j und alle Knoten m_t aus M_j :

$l(k_0,k_x) \leq l(k_0,k_j) \leq (k_0,m_t)$

(falls jeweils ein Pfad zwischen den Knoten existiert.)

Das bedeutet,dass die Pfadlänge der gefundenen Pfade mit jedem erreichten Knoten zunimmt.

Beweis:

Durch vollständige Induktion nach der Anzahl vermindert um 1 (ein markierter Knoten ist schon der Startknoten) der schon als besucht markierten Knoten j:

Induktionsanfang:

Vorgegeben sei ein Startknoten k_0 im Graph G.Der Startknoten k_0 wird als besucht markiert.

Dann werden nach dem Algorithmus von Dijkstra Punkte 2) und 3) alle Kanten der von k_0 erreichbaren Nachbarknoten als Pfad in der Pfadliste gespeichert und die Pfadliste sortiert.

Es wird nun der Pfad,d.h. die Kante mit der kleinsten Länge aus der Pfadliste gelöscht und der Endknoten k_1 bei 4) als besucht markiert.Die Pfade von k_0 zu den erreichbaren Nachbarknoten von k_1 werden in die Pfadliste bei 5) am Anfang eingefügt und die Pfadliste wird absteigend sortiert.

Vom Startknoten k_0 zum Knoten k_1 führt ein minimalster Pfad,da dies die Kante mit der kleinsten Länge ist.

Die Menge der besuchten Knoten B_1 des Graphen besteht dann aus den Knoten k_0 und k_1.Also ist j=1.Die Menge der übrigen Knoten des Graphen sei M_1.Die Menge der von den Knoten von B_1 aus erreichbaren Nachbarknoten sei N_1.N_1 ist dann Teilmenge von M_1 .

Damit ist der Beweis für j=1 geführt.

Induktionsschritt:

Annahme:

Es sind schon die minimalsten Pfade von k_0 zu den j-1 als besucht markierten Knoten k_1,k_2 bis k_{j-1} gefunden.Die Menge dieser Knoten einschließlich k_0 sei mit B_{j-1} bezeichnet.Die Menge der von diesen Knoten erreichbaren Nachbarknoten,die nicht in B_{j-1} liegen,sei mit N_{j-1} bezeichnet.Alle Knoten des Graphen,die nicht in B_{j-1} enthalten sind,seien mit M_{j-1} bezeichnet.N_{j-1} ist dann eine Teilmenge von M_{j-1}.

Mit l sei die kleinste Pfadlänge eines Pfades zwischen zwei Knoten bezeichnet.Dann gilt für alle Knoten k_x aus B_{j-1} und alle Knoten m_t aus M_{j-1}:

$l(k_0,k_x) \leq l(k_0,k_{j-1}) \leq l(k_0,m_t)$

(Falls jeweils ein Pfad zwischen den Knoten existiert.)

Schritt von j-1 auf j:

Zu dem Knoten k aus N_{j-1} mit dem Pfad p,der die minimalsten Pfadlänge von k_0 unter allen Pfaden zu allen Knoten aus N_{j-1} hat,wird nach 3) der Pfad p ausgewählt.Der Knoten k wird mit k_j bezeichnet und als besucht markiert.Zusammen mit den bisherigen besuchten Knoten bildet k_j die Menge B_j.Die Menge M_j ist die Menge der übrigen Knoten des Graphen.Die Menge der

von k_j erreichbaren und noch nicht besuchten Knoten wird dann mit der Menge N_{j-1} aus der k_j entfernt wird,vereinigt und diese Knoten bilden die Menge N_j,die eine Teilmenge von M_j ist.Die Pfade von p_i von k_0 zu den von k_j aus erreichbaren Knoten werden in der Pfadliste gespeichert,so dass jetzt wieder alle Pfade von k_0 aus zu den Knoten aus N_j in der Pfadliste gespeichert sind.

Der Pfad p von k_0 zu k_j ist dann der minimalste Pfad unter allen Pfaden des Graphen.

Gäbe es nämlich einen anderen kürzeren Pfad p' von k_0 zum Knoten k_j mit $l(p')<l(k_0,k_j)=l(p)$,müßte dieser,da k_j ein Knoten aus N_{j-1} bzw. M_{j-1} ist,im Pfad einen (in der Pfadreihenfolge) ersten Knoten P' aus B_{j-1} enthalten,dessen Nachfolger im Pfad P'' in N_{j-1} liegt.Der Pfad von k_0 über P' zu P'' sei p''.

Dann gilt:

$l(p')\geq l(p'')=l(k_0,P'')\geq l(k_0,k_j)=l(p)$

wegen nichtnegativer Kantenbewertung und da die Pfade von k_0 zu allen Nachbarknoten P'' aus N_{j-1} nach Auswahl des Knoten k_j größer als der minimalste Pfad ist,der von k_0 zum Knoten k_j über k_{j-1} führt.(nach Auswahl ist dieser Pfad nämlich der minimalste Pfad der Pfadliste,die alle Pfade zu den Knoten von N_{j-1} enthält)

Das ist ein Widerspruch dazu,dass p' ein kürzerer Pfad zu k_j ist.

In gleicher Weise ergibt sich,dass jeder Pfad von k_0 aus zu einem beliebigen Knoten aus N_j (zu $k=k_j$ führte schon der Pfad mit der kleinsten Pfadlänge aller Pfade zu Knoten aus N_{j-1}) und damit auch M_j mindestens die Länge $l(k_0,k_j)$ hat,denn diese Pfade führen immer über Knoten aus N_{j-1} oder über k_j selber.

Damit ist der Induktionsschritt gezeigt.

Quellcode:

(Unit UPfad)

Die Pfadliste hat im folgenden Quelltext den Namen Wegpfadliste.

```
function TPfadgraph.BestimmeminimalenPfad(Kno1,Kno2:TKnoten):TPfad;
label Endproc;
var WegPfadliste:TPfadliste;
     MomentanerWeg,MomentanerWegneu:TKantenliste;
     Index1,Index2:Integer;
     Ka,Ka1,Ka2:TKante;
     Kno:TKnoten;
     Ob:TObject;
begin
  BestimmeminimalenPfad:=TPfad.Create;                                    1)
  WegPfadliste:=TPfadliste.Create;
  LoescheKnotenbesucht;
  Kno1.Besucht:=true;
  if not Kno1.AusgehendeKantenliste.Leer then                             2)
    for Index1:=0 to Kno1.AusgehendeKantenliste.Anzahl-1  do
    begin
      Ka:=Kno1.AusgehendeKantenliste.Kante(Index1);
      MomentanerWeg:=TKantenliste.Create;
      Ka.Pfadrichtung:=Ka.Zielknoten(Kno1);
      MomentanerWeg.AmAnfanganfuegen(Ka);
      WegPfadliste.AmAnfanganfuegen(Momentanerweg.Graph);
    end;
  WegPfadliste.Sortieren(Pfadvergleich,Bewertung);                        3)
  while not Wegpfadliste.Leer  do                                         4)
  begin
    Wegpfadliste.AmEndeloeschen(Ob);                                      5)
    Momentanerweg:=TGraph(Ob).Kantenliste;                                6)
    Ka1:=MomentanerWeg.Kante(MomentanerWeg.Letztes);
```

```
    Kno:=Ka1.Pfadrichtung;                                                    7)
    if Kno=Kno2 then
    begin
      Bestimmeminimalenpfad:=TPfad(MomentanerWeg.Kopie.Graph);               8)
      goto Endproc;
    end;
    if not Kno.Besucht                                                        9)
    then
    begin
      Kno.Besucht:=true;                                                     10)
      if not Kno.AusgehendeKantenliste.Leer then                             11)
        for Index2:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
        begin
          Ka2:=Kno.AusgehendeKantenliste.Kante(Index2);
          if not Ka2.Zielknoten(Kno).Besucht then
           begin
             Ka2.Pfadrichtung:=Ka2.Zielknoten(Kno);
             MomentanerWegneu:=TKantenliste.Create;
             MomentanerWegneu:=MomentanerWeg.Kopie;
             MomentanerWegneu.AmEndeanfuegen(Ka2);
             WegPfadliste.AmAnfanganfuegen(Momentanerwegneu.Graph);
           end;
        end;
    end
    else
    begin
      MomentanerWeg.Free;
      MomentanerWeg:=nil;
    end;
    WegPfadliste.Sortieren(Pfadvergleich,Bewertung);                         12)
  end;
  Endproc:
  while not Wegpfadliste.Leer do
  begin
    Wegpfadliste.AmEndeloeschen(Ob);
    TPfad(Ob).Free;
    Ob:=nil;
  end;
  Wegpfadliste.Free;
  Wegpfadliste:0nil;
end;
```

Die obige Methode benutzt die Function Pfadvergleich:

```
function PfadVergleich(Ob1,Ob2:TObject;Wert:TWert):Boolean;
var Pfad1,Pfad2:TPfad;
begin
  Pfad1:=TPfad(Ob1);
  Pfad2:=TPfad(Ob2);
  Pfadvergleich:=Pfad1.Pfadsumme(Wert)>Pfad2.Pfadsumme(Wert);
end;
```

Erläuterung des Algorithmus:

Nach den erforderlichen Initialisierungen (Erzeugen der Pfadliste MomentanerWeg und dem Zurücksetzen der Besucht-Markierung aller Knoten des Graphen sowie dem Setzen der Besucht-Markierung des Startknotens) bei 1) werden bei 2) alle Kanten des Startknoten in die Pfadliste Wegpfadliste als Pfade eingefügt.Die Wegpfadliste wird bei 3) absteigend sortiert.Solange die Wegpfadliste nicht leer 4) ist,wird der minimalste (letzte) Pfad bei 5) aus der Liste entfernt und als MomentanerWeg bei 6) gespeichert.Der Endknoten von MomentanerWeg wird bei 7) mittels der letzten Kante ermittelt,und es wird bei 8) überprüft,ob dieses schon der Zielknoten Kno2 ist.Wenn dies der Fall ist,wird der Algorithmus durch einen Sprung zum Label Endproc beendet und der Pfad bei 8) als Resultat der Function zurückgegeben.
Wenn der Endknoten von MomentanerWeg noch nicht als besucht markiert war 9),wird er bei 10) jetzt als besucht markiert,und bei 11) werden die vom Endknoten ausgehenden Kanten jeweils zum Pfad MomentanerWeg hinzugefügt und so für jede Kante neue Pfade als MomentanerWegneu erzeugt,die der Wegpfadliste am Anfang hinzugefügt werden.
Bei 12) wird die Wegpfadliste erneut absteigend sortiert,so dass wieder der kleinste Pfad am Ende der Liste ist.Der Algorithmus endet,wenn die Wegpfadliste leer ist,oder wenn der Zielknoten Kno2 erreicht ist.

Wie erhält man jedoch zwei Knoten des Graphen per Mausklick,zwischen denen man den minimalen Pfad bestimmen lassen möchte?Ein geeignete Methode dazu ist die Methode TInhaltsgraph.ZweiKnotenauswaehlen,die in der folgenden Methode MinimalenPfadzwischenzweiKnotenbestimmen benötigt wird:

```
function TPfadgraph.MinimalenPfadzwischenzweiKnotenbestimmen(X,Y:Integer;
  Ausgabe:TLabel;var SListe:TStringList;Flaeche:TCanvas):Boolean;
label Endproc;
var T:TInhaltsgraph;
    Kno1,Kno2:TPfadknoten;
    Gefunden:Boolean;
begin
  result:=false;
  if self.ZweiKnotenauswaehlen(X,Y,TInhaltsknoten(Kno1),TInhaltsknoten(Kno2),Gefunden)
    and Gefunden                                                            (****)
  then
  begin
    result:=true;
    if Kno1=Kno2
    then
    begin
      Showmessage('Die beiden Knoten sind identisch!');
      Goto Endproc;
    end;
    SListe:=TStringList.Create;
      T:=TInhaltsgraph(self.BestimmeMinimalenPfad(Kno1,Kno2));
      if not T.Kantenliste.Leer then
      begin
        T.FaerbeGraph(clred,psdot);
        T.ZeichneGraph(Flaeche);                                              13)
        Ausgabe.Caption:=                                                     14)
        T.InhaltallerKnoten(ErzeugeKnotenstring)+' Summe: '+
        RundeZahltostring(T.Kantensumme(Bewertung),Kantengenauigkeit);
        SListe.Add(Ausgabe.Caption);
        Ausgabe.Refresh;
        Messagebeep(0);
        Pause(2000);
        Ausgabe.Caption:='';
        Ausgabe.refresh;
        T.ZeichneGraph(Flaeche);
        result:=true;
    end
    else
      ShowMessage('Kein minimaler Pfad zwischen den Knoten');
  end
  else
    result:=false;
  Endproc:
end;
```

Erläuterung des Algorithmus:

Der Variablen T vom Typ TInhaltsgraph wird der minimale Pfad als Graph zugeordnet und anschließend gezeichnet sowie die Knotenfolge des Pfades im Label Ausgabe mit Pfadsumme und Pfadprodukt ausgegeben (Markierungen 13 und 14).

Das Problem dieser Methode ist,dass sie in einer Ereignismethode von Delphi aufgerufen werden soll,die die Koordinaten eines Mausklickpunkts x und y bestimmt.Dabei werden jedoch zunächst einmal nur die Koordinaten eines Knotens,der Ziel des Klicks ist,festgelegt.Erst wenn die Methode ein zweites Mal durch einen Mausklick aktiviert wird,werden die Koordinaten eines weiteren Punkts bestimmt.Danach soll die Ereignismethode verlassen werden.

Dieses Verhalten der Methode wird durch den Aufruf der Methode BeiMausklickzweiKnotenauswaehlen bei (*****) erzeugt.Diese Methode gibt den Wert true zurück,wenn sie zum zweiten Mal mit jeweils einem neuen Paar von x,y-Koordinaten aufgerufen wurde,und wenn dabei die Koordinatenpaare x,y zwei Stellen des Bildschirms auswählen,an denen sich Knoten befinden.In diesem Fall ist auch der Referenzparameter gefunden true.So braucht der Methode MinimalenPfadzwischenzweiKnotenbestimmen nur jeweils (nacheinander)

ein Paar von x,y-Koordinaten von der MausklickEreignismethode übergeben zu werden.

Wenn zwei Knoten erfolgreich ausgewählt wurden,wird der Rückgabewert der function BeiMausklickzweiKnotenauswaehlen auf true gesetzt und damit auch der Rückgabewert result der Funktionsmethode MinimalenPfadzwischenzwei-Knotenbestimmen auf true gesetzt,so dass in der aufrufenden Ereignisprocdu-re entschieden werden kann,wann diese Methode wieder verlassen werden muß (ansonsten ist der Rückgabewert false).

Der obige Algorithmus BestimmemimialenPfad läßt sich leicht so abwan-deln,dass alle minimalen Pfade,die von einem Startknoten eines Graphen zu den Nachbarknoten führen,bestimmt werden können.Er ist eine bessere,weil schnellere Variante des im letzteren Kapitel beschriebenen Verfahrens Er-zeugeminimalePfade:

Algorithmus Alle minimalen Pfade von einem Knoten nach Dijkstra erzeugen

(Unit UPfad)

```
procedure TPfadknoten.ErzeugeminimalePfadenachDijkstra;
label Endproc;
var WegPfadliste:TPfadliste;
     MomentanerWeg,MomentanerWegneu:TKantenliste;
     Index:Integer;
     Ka{,Ka1,Ka2}:TKante;
     Kno:TKnoten;
     Ob:TObject;
begin
  Graph.Pfadlistenloeschen;
  WegPfadliste:=TPfadliste.Create;
  Graph.LoescheKnotenbesucht;
  Besucht:=true;
  if not AusgehendeKantenliste.Leer then
    for Index:=0 to AusgehendeKantenliste.Anzahl-1  do
    begin
      Ka:=AusgehendeKantenliste.Kante(Index);
      MomentanerWeg:=TKantenliste.Create;
      Ka.Pfadrichtung:=Ka.Zielknoten(self);
      MomentanerWeg.AmAnfanganfuegen(Ka);
      WegPfadliste.AmAnfanganfuegen(Momentanerweg.Graph);
    end;
  WegPfadliste.Sortieren(Pfadvergleich,Bewertung);
  while not Wegpfadliste.Leer  do
  begin
    Application.ProcessMessages;
    if Graph.Abbruch then goto Endproc;
    Wegpfadliste.AmEndeloeschen(Ob);
    Momentanerweg:=TGraph(Ob).Kantenliste;
    Ka:=MomentanerWeg.Kante(MomentanerWeg.Letztes);
    Kno:=Ka.Pfadrichtung;
    if not Kno.Besucht
    then
    begin
      Kno.Besucht:=true;
      Kno.Pfadliste.AmEndeanfuegen(MomentanerWeg.Kopie.Graph);                           15)
      Pfadliste.AmEndeanfuegen(MomentanerWeg.Kopie.Graph);                               16)
      if not Kno.AusgehendeKantenliste.Leer then
        for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
        begin
          Ka:=Kno.AusgehendeKantenliste.Kante(Index);
          if not Ka.Zielknoten(Kno).Besucht then
          begin
            Ka.Pfadrichtung:=Ka.Zielknoten(Kno);
            if not Ka.Pfadrichtung.Besucht then
            begin
              MomentanerWegneu:=MomentanerWeg.Kopie;
              MomentanerWegneu.AmEndeanfuegen(Ka);
              WegPfadliste.AmAnfanganfuegen(Momentanerwegneu.Graph);
            end;
          end;
        end;
    end
    else
      MomentanerWeg.Free;
      MomentanerWeg:=nil;
```

```
WegPfadliste.Sortieren(Pfadvergleich,Bewertung);
  end;
  Endproc:
  Wegpfadliste.Freeall;
  Wegpfadliste:=nil;
end;
```

Erläuterung des Algorithmus:

Im Unterschied zur vorigen Methode handelt es sich hier nicht um eine Function,sondern um eine Procedure,und es ist eine Methode von TPfadknoten und nicht von TPfadgraph.

Der Hauptunterschied besteht im Wegfall des Abschnittes 9) (Überprüfen,ob der Zielknoten schon erreicht ist) und im Hinzufügen der Zeilen 15) und 16),in denen bei der Besucht-Markierung des Endknotens eines Pfades der minimale Pfad zu diesem Knoten in der Pfadliste dieses Knotens gespeichert und der Pfadliste des Startknotens hinzugefügt wird,so dass in der Pfadliste des Startknotens beim Ende des Verfahrens alle minimalen Pfade gespeichert sind.
Zeile 15) kann noch entfallen,wenn nicht auf die Pfadlisten der Einzelknoten zurückgegriffen werden soll.Zeile 16) könnte durch eine Farbmarkierung und Zeichnung des Graphen (Pfad MomentanerWeg),wie im vorigen Kapitel beschrieben,ersetzt werden.

z.B.:

```
TPfadgraph(MomentanerWeg.Graph).Graphzeichnen(Flaeche,Ausgabe,Bewertung,Sliste,
  TPfadgraph(self.Graph).Demo,TPfadgraph(self.Graph).Pausenzeit,
  TPfadgraph(self.Graph).Kantengenauigkeit);
```

oder als Einzelanweisungen:

```
T:=TPfadgraph.Create;
TObject(T):=MomentanerWeg.Graph.Kopie;
T.FaerbeGraph(clred,psdot);
T.ZeichneGraph(Flaeche);
....
....

mit var T:TInhaltsgraph;
```

Die Methode BestimmeMinimalenPfad von TGraph der Unit Ugraph arbeitet nach einem modifizierten Rangalgorithmus nach Ford und bestimmt den minimalen Pfad auch im Fall negativer Kantenbewertung (außer bei negativen Kreisen) bei gerichteten Graphen (Lit 45 S.106).Die Funktionsweise wird im Anhang beschrieben.Dieses Verfahren kommt zum Einsatz in dem Algorithmus Minimale-Kosten und den darauf aufbauenden Algorithmen (Kapitel CXIII und CXIV). Dieses Verfahren sollte also dann zusätzlich im Vergleich zur Dijkstra-Methode behandelt werden,wenn der dort verwendete Algorithmus von Busacker und Gowen später im Unterricht benötigt oder programmiert werden soll.Im Menü Abstand von zwei Knoten des Menüs Pfade (Konzeption DWK) kann man zwischen beiden Verfahren wählen.

Algorithmus Alle Pfade zwischen zwei Knoten

(Unit UPfad)

Verbale Beschreibung:

Siehe dazu die entsprechende Beschreibung in Kapitel II,da das Verfahren im letzten Kapitel nämlich schon als Erweiterung des Tiefenbaudurchlaufs im Prinzip besprochen wurde.Als problemorientierte Einstiegsaufgabe zum Erkennen des Algorithmus durch Schüler können deshalb die Aufgaben C II 1.d und e (mittels Demomodus) benutzt werden.(Außerdem können dazu auch als Bespiele die Graphen G002.gra und G005.gra gewählt werden.)

Erläuterung des Algorithmus:

Die Idee dieses Algorithmus ist es,alle Pfade nach dem Verfahren ErzeugeAllePfade,die nach dem TiefenBaumsuchlauf mit jeweiligem Löschen der Besucht-Markierung der Knoten auf dem Rückweg arbeitet,vom Startknoten zu erzeugen,wobei die Pfade des Graphen zum Zielknoten in der Pfadliste des Zielknotens und in der Pfadliste des Startknotens gespeichert werden.
Danach brauchen nur noch die Pfade der Pfadliste des Zielknotens ausgegeben werden.Die Methode AllePfadeerzeugen (siehe im vorigen Kapitel II) ist also dabei so abzuwandeln,dass nur noch die Pfade zum Zielknoten in der Pfadliste des Zielknoten gespeichert werden.Dieses leistet die Methode TKnoten.ErzeugeallePfadeZielknoten.

In dieser Methode fehlt die Zeile zum Speichern der Pfade in der Pfadliste des Startknotens,und sie enthält stattdessen den Eintrag:

```
if Ka.Zielknoten(Kn)=Kno then                                                  (*****)
  Ka.Zielknoten(Kn).Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
```

(statt der Zeile

```
Pfadliste.amEndeanfuegen(MomentaneKantenliste.Kopie.Graph); )
```

Der Quelltext der Methode lautet dann:

```
procedure TPfadknoten.ErzeugeallePfadeZielknoten(Kno:TKnoten);
var Index:Integer;
    MomentaneKantenliste:TKantenliste;


  procedure GehezuallenNachbarknoten(Kn:TKnoten;Ka:TKante);
  var Ob:TObject;
      Index:Integer;
  begin
    if not Ka.Zielknoten(Kn).besucht then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kn);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
      if Ka.Zielknoten(Kn)=Kno then
        Ka.Zielknoten(Kn).Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);  (*****)
      Ka.Zielknoten(Kn).Besucht:=true;
      if not Ka.Zielknoten(Kn).AusgehendeKantenliste.Leer then
        for Index:= 0 to Ka.Zielknoten(Kn).AusgehendeKantenliste.Anzahl-1 do
         GehezuallenNachbarknoten(Ka.Zielknoten(Kn),Ka.Zielknoten(Kn).
         AusgehendeKantenliste.Kante(Index));
      MomentaneKantenliste.AmEndeloeschen(Ob);
      Ka.Zielknoten(Kn).Besucht:=false;
    end;
  end;

begin
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.Pfadlistenloeschen;
  Graph.LoescheKnotenbesucht;
  Besucht:=true;
  if not AusgehendeKantenliste.Leer then
    for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
      GehezuallenNachbarknoten(self,AusgehendeKantenliste.Kante(Index));
  MomentaneKantenliste.Free;
  MomentanerKantenliste:=nil;
end;
```

Der Quelltext von AllePfadezwischenzweiKnotenzeigen lautet:

```
function TPfadgraph.AllePfadezwischenzweiKnotenbestimmen(X,Y:Integer;
  Ausgabe:TLabel;var SListe:TStringList;Flaeche:TCanvas):Boolean;
label Endproc;
var Zaehl:Integer;
    T:TInhaltsgraph;
    Kno1,Kno2:TPfadknoten;
    Gefunden:Boolean;
begin
  result:=false;
  if self.ZweiKnotenauswaehlen(X,Y,TInhaltsknoten(Kno1),TInhaltsknoten(Kno2),Gefunden)
    and Gefunden                                                                     1)
  then
```

```
  begin
    if Kno1=Kno2
    then
    begin
      Showmessage('Die beiden Knoten sind identisch!');
      Goto Endproc;
    end;
    Ausgabe.Caption:='Berechnung läuft...';
    Kno1.erzeugeallePfadeZielknoten(Kno2);
    result:=true;
    if Kno2.Pfadliste.Leer then                                                    2)
    begin
      Ausgabe.Caption:='';
      Ausgabe.Refresh;
      ShowMessage('Keine Pfade zwischen den Knoten');
    end
    else
      Kno2.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);                     3)
  end
    else
      result:=false;
  Endproc:
end;
```

Erläuterung des Algorithmus:

Zunächst werden zwei Knoten nach der schon weiter oben beschriebenen Methode bei 1) erfolgreich ausgewählt.Bei 2) werden dann alle Pfade vom Startknoten Kno1 im Graphen mittels der oben beschriebenen Methode Erzeugealle-PfadeZielknoten erzeugt.

Bei 3) werden alle Pfade der Pfadliste des Zielknotens Kno2 als Graphen rot markiert gezeichnet und die Knotenfolge des Pfades mitsamt Pfadlänge und dem Produkt der Pfadkantenwerte wird im Label Ausgabe ausgegeben.Außerdem werden diese Informationen an die Stringliste SListe,die zur Ausgabe im Ausgabefenster dient,übergeben.

C Aufgabe III.3: (Übung:minimale Pfade)

Gegeben ist der Graph von Aufgabe C III.2 (G011.gra) und dadurch das Straßennetz zwischen den Orten A bis H.

a)Ermittle nach dem Algorithmus nach Dijkstra alle minimalen Pfade von Ort A aus zu den übrigen Orten.

b)Ermittle den minimalen Pfad zwischen Ort F und B.

(Lösung per Hand,mittels eigenem erstellten Quellcode durch die Entwicklungsumgebung oder mittels des Menüs/Abstand von zwei Knoten des fertigen Programms Knotengraph je nach Konzeption)

Lösung:

a)
A H Summe: 3 Produkt: 3
A C Summe: 3 Produkt: 3
A D Summe: 4 Produkt: 4
A H B Summe: 5 Produkt: 6
A C F Summe: 5 Produkt: 6
A H E Summe: 7 Produkt: 12
A C F G Summe: 8 Produkt: 18

b)
F C H B Summe: 8

C IV Eulerbeziehung,ebene Graphen und Färbbarkeit

Dieses Kapitel enthält als wesentlichen Teil die Beschreibung (und Codierung) eines Algorithmus,der untersucht,ob die Knoten eines Graphen mittels einer vorgegeben Anzahl von Farben so zu färben sind,dass jeweils zwei (durch eine Kante) benachbarte Knoten verschiedene Farben erhalten.(Knotenfärbarkeit eines Graphen)

Die kleinste Anzahl dieser Farben heißt chromatische Zahl des Graphen.Daher eignet sich der Algorithmus u.a. auch zur Bestimmung dieser chromatischen Zahl.

Eine Unterrichtsreihe zu diesem Thema ist deshalb von besonderer Bedeutung,weil sie im Zusammenhang steht mit dem berühmten,lange Zeit ungelösten Vierfarbenproblem,dessen Aussage in der Form ausgedrückt werden kann,dass die chromatische Zahl eines ebenen Graphen maximal vier ist.

Damit ist es sinnvoll,in einer Unterrichtsreihe zu diesem Thema auch über die Planarität von Graphen zu sprechen.Mit der Planarität von Graphen ist wiederum die Eulerbeziehung verknüpft,die gerade für die Planarität eine notwendige Voraussetzung ist.

Durch geschicktes Anwenden dieser Beziehung kann nämlich gerade auf die Nichtplanarität spezieller Graphen geschlossen werden.(Entsprechende Test sind im Menü Eigenschaften/Kreise des Programm Knotengraph bzw. der Entwicklungsumgebung enthalten und geben bei positivem Ergebnis die Nichtplanarität zusätzlich zu dem Untersuchungsresultat des Graphen auf Kreise an.)

Schließlich sind noch die Graphen mit der chromatischen Zahl 2 gerade die bipartiten (paaren) Graphen,so dass dieser wichtige Begriff,der z.B. bei der Matchingtheorie eine Rolle spielt,hier didaktisch anschaulich vorbereitet werden kann.

Die Stoffauswahl dieses Kapitels eignet sich insbesondere auch für eine Unterrichtsreihe,die das fertige Programm Knotengraph nur als Werkzeug benutzt,um Probleme der Graphentheorie zu veranschaulichen und dadurch Zusammenhänge zu erkennen,ohne Algorithmen selber zu programmieren,z.B. beim Einsatz im Mathematikunterricht (Konzeption DWK).

In einer solchen Unterrichtsreihe würde man den Färbbarkeitsalgorithmus dann nur theoretisch ohne zu programmieren besprechen und seine Ergebnisse an Hand von Beispielgraphen (im Demomodus des Programms Knotengraph) verfolgen.

Stellt man dagegen die objektorientierte Programmierung des Algorithmus Färbbarkeit der Knoten des Graphen,der auch als Musterbeispiel eines Backtrackingverfahrens behandelt werden kann,in den Vordergrund (z.B. im Informatikunterricht),können die Sätze über die Eulerbeziehung und über die Planarität von Graphen,die eine bessere Sicht auf den Gesamtzusammenhang ermöglicht und die Problematik ausweiten,als Motivation zur intensiveren Beschäftigung mit diesem Themenbereich dienen.

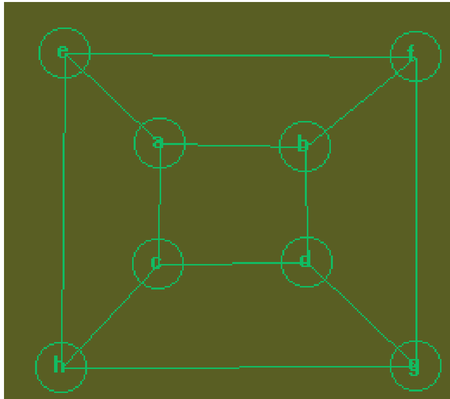
Die folgenden Beispiels-und Einführungsaufgaben sorgen für eine selbstständige Entdeckung der graphentheoretischen Zusammenhänge und Aussagen der mathematischen Sätze seitens der Schüler.

(Siehe dazu auch den entsprechenden Unterrichtsplan im Anhang als Ergänzung)

C Aufgabe IV.1 (Einstiegsproblem):

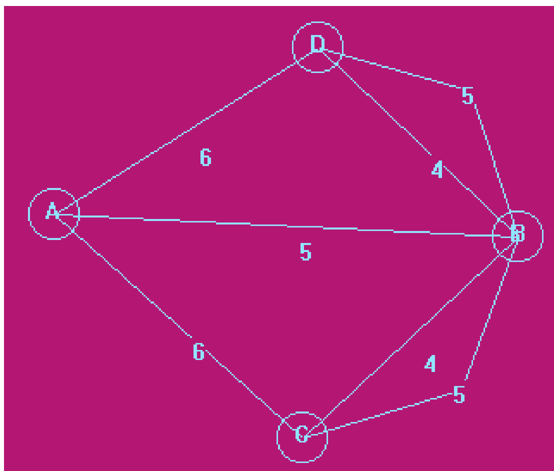
Gegeben sind folgende Graphen:

Graph 1:



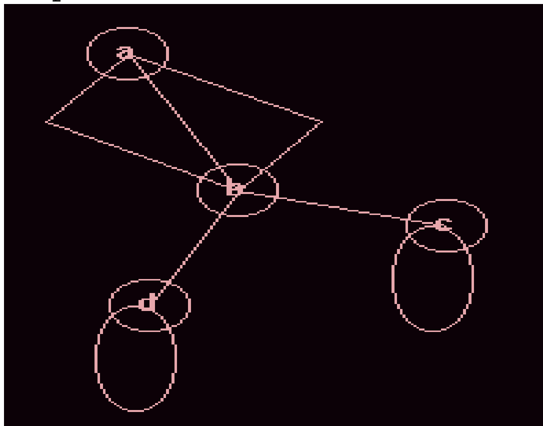
G012.gra

Graph 2:



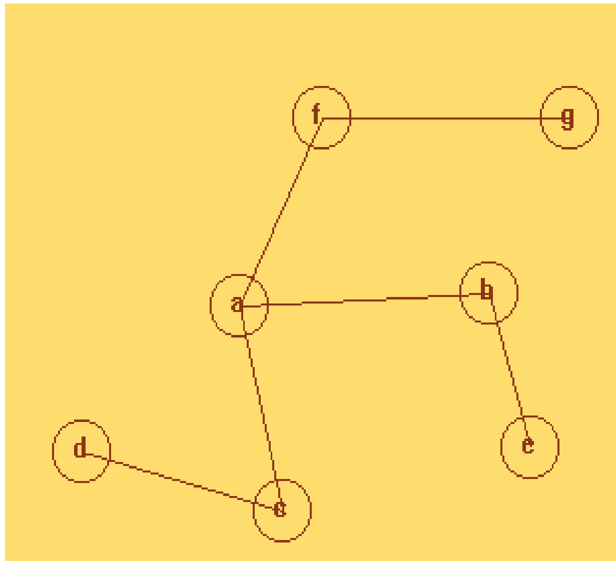
G013.gra

Graph 3:



G014.gra

Graph 4:



G015.gra

Eine von Kanten jeweils eingeschlossene ebene Fläche,die nicht mehr durch weitere Kanten geteilt wird, heißt Gebiet.

Außerdem zählt auch schon die um den Graph befindliche nach allen Seiten unbegrenzte anfangs leere Zeichenfläche,die schon vorhanden ist,bevor der Graph gezeichnet wurde,als ein Gebiet nämlich das äußere Gebiet.

Zähle die Anzahl der Knoten k,die Anzahl der Kanten a und die Anzahl der Gebiete f (Flächen),die jeweils in den einzelnen Graphen vorhanden sind, und stelle die Ergebnisse in einer Tabelle zusammen.

Tabelle:

Graph	k	a	f	k-a+f
1	8	12	6	2
2	4	7	5	2
3	4	7	5	2
4	7	6	1	2

C Aufgabe IV.2 (2.Einstiegsproblem):

Drei Häuser d,e und f sollen mit dem Gaswerk a,dem Wasserwerk b und dem Elektrizitätswerk c durch Leitungen verbunden werden.Der folgende Graph zeigt die Möglichkeit einer Verbindung der Häuser durch Leitungen mit den drei Werken.Die Leitungen überschneiden sich.
Überprüfe durch Knoten-/Kantenverschiebung mittels des Programms Knotengraph (isomorphe Graphen) oder durch zeichnerisches Probieren,dass es nicht möglich ist,die Leitungen durch Verformungen so zu legen,dass sie sich nicht überschneiden.

C Definition IV.1:

Ein Graph ist eben oder planar,wenn er (durch Verändern der Form der Kanten d.h. durch Erzeugen isomorpher Graphen,die dieselben Kanten-Knotenrelationen besitzen) beim Zeichnen in der Ebene so dargestellt werden kann,dass sich seine Kanten nicht gegenseitig überschneiden.

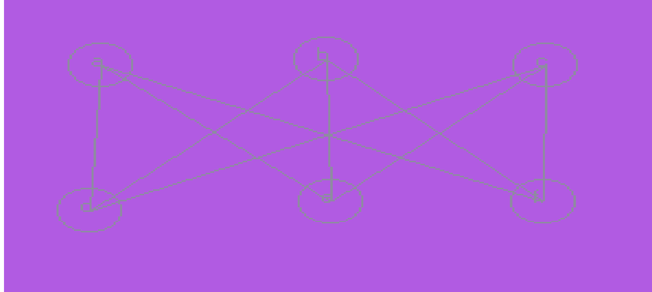
Für den folgenden Graphen gilt:

k=6 a=9

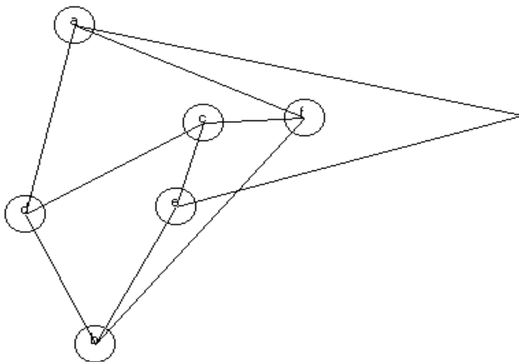
Da die Kanten sich stets überschneiden,gibt es keine Zahl für f.

Der Graph ist also anscheinend nicht eben.Ein Beweis wird bei der Besprechung der Kriterien für ebene Graphen gegeben.
Die obigen Graphen 1 bis 4 sind eben.

Gas-Wasser-Elektrizitäts-Graph:



G016.gra



Durch Verschieben erzeugter isomorpher Graph zu **G016.gra**

Als Ergebnis der Lösungen der Aufgaben IV.1 und IV.2 ergibt sich:

C Satz IV.1:

In einem ebenen Graphen ist k-a+f=2.

(Eulersche Formel)

Beweis:

Gegeben sei ein beliebiger Graph.Wenn der Graph noch kein Gerüst ist,können aus ihm Kanten entfernt werden,ohne dass sich die Anzahl der Komponenten erhöht,bis zum Schluß ein Gerüst entstanden ist.Dabei kann man immer nur solche Kanten entfernen,die zwei verschiedene Gebiete trennen,so dass sich bei jedem Schritt die Anzahl der Gebiete um eins vermindert.Wenn nur noch ein Gebiet vorhanden ist,ist ein Gerüst entstanden.
Es können also l=f-1 Kanten entfernt werden.

Nach Satz C I.4 ist diese Zahl aber auch durch l=a-k+1 gegeben.

Also gilt: f-1=a-k+1

Daraus folgt: k-a+f=2

Der Algorithmus des Menüs Eigenschaften/Anzahl Knoten und Kanten des Programms Knotengraph bzw. der Entwicklungsumgebung benutzt die Eulersche Formel,um bei einem ebenen Graphen die Anzahl der Gebiete zu ermitteln.

C Aufgabe IV.3:

Erstellen Sie die Graphen der Aufgabe IV.1 mit Hilfe des Programms Knotengraph, und ermitteln Sie mit Hilfe des Menüs Eigenschaften/Anzahl Knoten und Kanten die Anzahl der Gebiete dieser Graphen.Erzeugen Sie dabei auch isomorphe Graphen durch Verschieben von Knoten und Kanten.

C Satz IV.2:

Enthält ein zusammenhängender,ebener,schlichter Graph mit k Knoten und a Kanten einen kleinsten Kreis mit n Kanten,dann gilt:

$(n-2)a \le n(k-2)$

Beweis:

Die Anzahl der Gebiete mit n Kanten sei mit a_n bezeichnet.

Mit $a_n + a_{n+1} + \ldots = f$ (Anzahl der Gebiete)

Es gilt dann:

$n f \le na_n + (n+1)a_{n+1} + \ldots \le 2a$

Die zweite Ungleichung folgt,weil jede Kante maximal 2 verschiedenen, benachbarten Gebieten angehören kann.

Mit der Eulerschen Beziehung $f=a-k+2$ folgt:

$n(a-k+2) \le 2a$

Daraus folgt:

$(n-2)a \le n(k-2)$

C Definition IV.2:

Ein ebener Graph heißt maximal planar,wenn durch Hinzufügen einer beliebigen Kante zwischen zwei Knoten,die noch nicht durch eine Kante verbunden sind,ein nicht planarer Graph entsteht.

Ein ebener Graph heißt Dreiecksgraph,wenn alle Gebiete des Graphen von Kreisen mit 3 Kanten und damit drei Knoten gebildet werden.

C Satz IV.3:

Ist G ein vollständiger,ebener,zusammenhängender,schlichter Graph,so wird jedes Gebiet von einem Kreis aus drei Kanten begrenzt.Ein solcher Graph ist ein Dreiecksgraph.

Dann gilt:

$a=3k-6$

Ein solcher Graph heißt maximal planar.

Für beliebige ebene Graphen gilt:

$a \le 3k-6$

Beweis:

In diesem Fall gilt:

$3f = 3a_3 = 2a$, da der Term 3f (Anzahl der Dreiecksgebiete multipliziert mit jeweils der Anzahl von 3 Kanten) jede Kante doppelt zählt.

Also folgt aus dem vorigen Satz:

$(n-2)a=n(k-2)$ mit $n=3$

d.h. $a=3(k-2)$

also: $a=3k-6$

Jeder ebene,schlichte zusammenhängende Graph,dessen kleinste Kreise keine Dreiecke sind,läßt sich durch Zeichen von zusätzlichen Kanten im Inneren der Nicht-Dreiecksgebiete so ergänzen,daß ein Dreiecksgraph entsteht.

Also gilt für beliebige ebene Graphen:

$a \leq 3k-6$

Bemerkung:

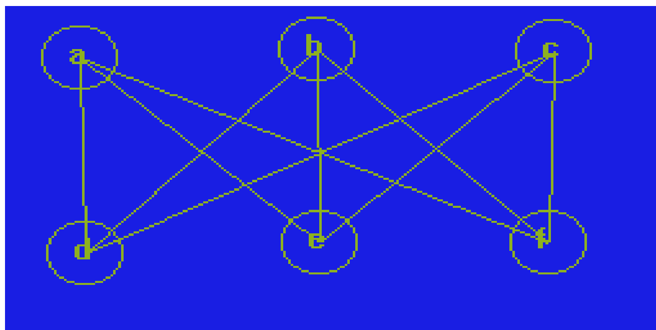
Die Formeln $(n-2)a \leq n(k-2)$ und $a \leq 3k-6$ sind notwendige aber keine hinreichenden Bedingungen für die Existenz eines planaren Graphen.Sie können aber zum Nachweis der Nichtplanarität benutzt werden.Die Formeln werden im Algorithmus des Menüs Eigenschaften/ Kreise benutzt,um gegebenenfalls (falls die Kriterien eine Aussage zulassen) die Nichtplanarität oder die maximale Planarität eines Graphen (zusätzlich zur Eigenschaft,ob eine Eulerlinie existiert) anzugeben.

C Aufgabe IV.4:

Erzeuge mit Hilfe von Knotengraph die folgenden Graphen und benutze die Menüpunkte Eigenschaften/Anzahl Knoten und Kanten sowie Eigenschaften/ Kreise,um die Anzahl von Knoten und Kanten sowie die Kantenzahl des kleinsten Kreises anzeigen zu lassen.
Wende die obigen notwendigen Bedingungen an,um die Nichtplanarität der folgenden Graphen zu zeigen.Überprüfe das Ergebnis jeweils mit Hilfe des Menüs Eigenschaften/ Kreise,in dem gegebenenfalls die Nichtplanarität gemäß den obigen Bedingungen angezeigt wird.
Erzeuge auch weitere eigenen Graphen und prüfe auf Planarität.

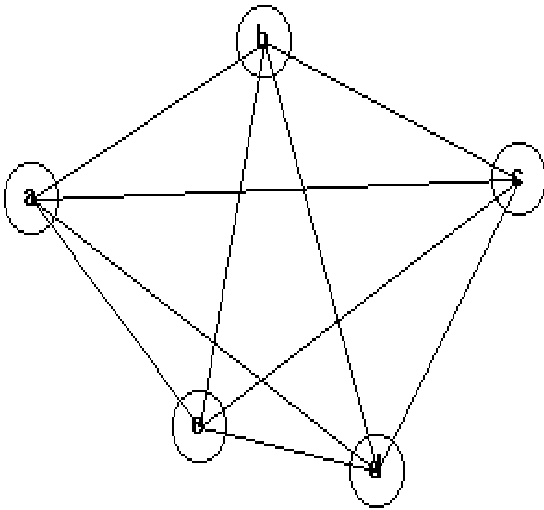
Bemerkung:Das Menü Kreise ermittelt u.a. die Kantenzahl eines kleinsten (mit Kantenzahl k>2) und eines größten Kreises in dem aktuellen,als ungerichtet aufgefaßten Graphen.Außerdem können alle Kreise und ihre Anzahl mit vorzugebener Kantenzahl in dem aktuellen,als ungerichtet aufgefaßten Graph bestimmt werden.Die Art der Algorithmen ist bei der Bedienungsanleitung im Anhang beschrieben.

Graph 1:



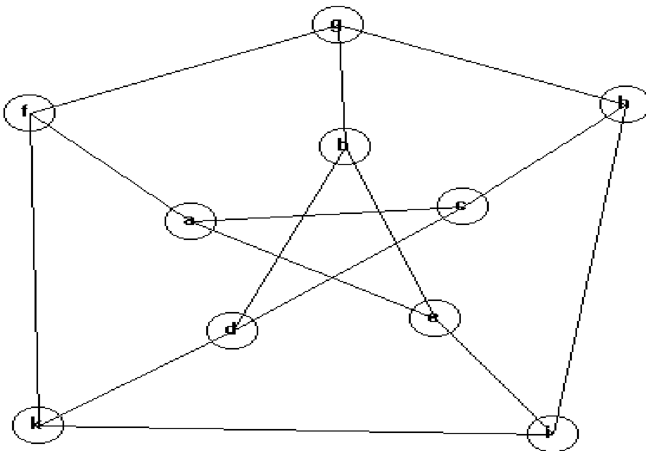
G016.gra

Graph 2:



G017.gra

Graph 3:



G018.gra

Graph 1:

Es ist n=4,k=6 und a=9.
Nach der 1.Ungleichung ist:

2*9>4*(6-2) d.h. 18>16

Graph 2:

Es ist: n=3,k=5 und a=10

Nach der 2.Ungleichung ist:

10>3*5-6 d.h. 10>9

Graph 3:

Es ist:n=5,k=10 und a=15

Nach der ersten Ungleichung ist:

3*15>5(10-2) d.h. 45>40

Damit ist gezeigt,dass die Graphen nicht planar sind.
Die Graphen 1 und 2 haben besondere Namen und eine besondere Bedeutung.
Der Gas-Wasser-Elektrizitäts-Graph heißt auch $K_{3,3}$-Graph.Der 2.Graph hat den Namen K_5-Gaph.

Der K_5-Graph sollte mittels Kanten-und Knotenverschiebung auf Planarität von den Schülern untersucht werden.

Die bisher angegebenen Bedingungen waren nur notwendige Bedingungen für die Planarität eines Graphen.

Wenn man unter der Unterteilung eines Graphen einen Graphen versteht,der entsteht,indem man die einzelnen Kanten des vorhanden Graphen jeweils so unterteilt,dass man in sie als Teilungsknoten weitere Knoten einfügt, wodurch der ursprüngliche Graph ein Teilgraph des entstandenen Graphen wird (und der zusätzlichen Möglichkeit der isomorphen Verformung des so entstandenen Graphen z.B. durch Knotenverschiebung),so gilt nach Kuratowski:

C Satz IV.4:

Ein Graph ist genau dann eben,wenn er weder eine Unterteilung der Graphen $K_{3,3}$ noch K_5 enthält.

Ein Beweis dieses Satzes ist für die Schule zu aufwendig und soll deshalb hier nicht vorgestellt werden.

Außerdem ist dieses hinreichende Kriterium in den meisten Fällen insbesondere für die Untersuchung mit Hilfe eines Algorithmus zu aufwendig.Es sind bisher keine einfach zu handhabenden hinreichenden Bedingungen bekannt,deshalb ist man z.B. auf die oben genannten notwendigen Bedingungen angewiesen.

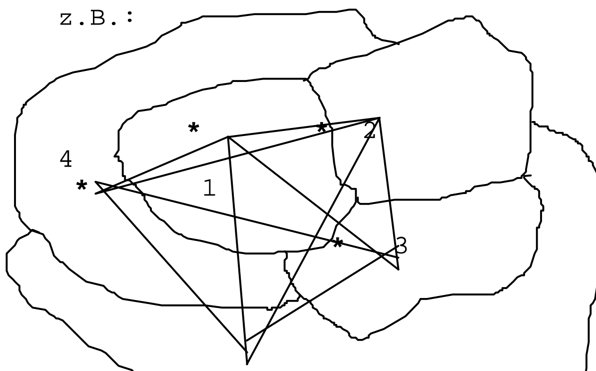
Als Übungsaufgaben sollten von Schülern an dieser Stelle weitere Graphen mit Hilfe der notwendigen Kriterien auf Nichtplanarität untersucht werden.Zur Bestimmung der Kreis-,Kanten-und Knotenzahl ist das Programm Knotengraph zu benutzen,womit auch das Ergebnis mittels des Menüs Eigenschaften/Kreise kontrolliert werden kann,das bei Nichterfüllung der notwendigen Bedingungen die Nichtplanarität anzeigt.

C Aufgabe IV.5 (Einstiegsproblem):

Ein König hatte 5 Söhne, und weil die Söhne sehr zerstritten waren,sollte das Land nach seinem Tode so an die Söhne aufgeteilt vererbt werden,dass jeder Sohn in das Land eines anderen Sohnes gelangen konnte,ohne das Land eines 3. Sohnes überqueren zu müssen,d.h. von den 5 Gebieten sollte jedes Gebiet mit jedem anderen eine Grenze haben.Grenze soll hier eine Linie und nicht nur einen Punkt bedeuten.(Über einen Punkt kann man nicht gehen.)

Versuche mittels einer Zeichnung von 5 jeweils zusammenhängenden Ländern eine Lösung zu geben.

z.B.:



5

Das Gebiet 1 hat hier keine Grenze mit Gebiet 5.

Durch Probieren stellt man fest,dass es anscheinend keine Lösung gibt.

Das Problem läßt sich auf ein Problem der Graphentheorie zurückführen,wenn man die Hauptstädte der Länder jeweils als Knoten auffaßt (mit * gekennzeichnet) und jeweils die Verbindungsstraßen zwischen den Hauptstädten als Kanten.Dann sollte es,wenn das Länder-Problem lösbar ist,möglich sein,die Verbindungskanten (Straßen) der Knoten (Hauptstädte) so in einer Ebene zu zeichnen,dass sie sich nicht schneiden.

Während die Kanten 5_2,2_4 noch mittels „Verformung" durch die entsprechenden Länder noch überschneidungsfrei gelegt werden können,muß die Kante 1_5 dann stets eine der anderen Kanten schneiden z.B. 3_4.

Der entstehende Graph ist der K_5-Graph.

Das Problem ist also zurückgeführt auf die Konstruktion eines ebenen K_5-Graphen,bei dem oben gezeigt wurde,dass er nicht plättbar ist.

Deshalb ist das Problem unlösbar.

Das gestellte Problem läßt sich noch anders ausdrücken:

Man benötigt zur Färbung der 5 Länder jeweils nur 4 verschiedene Farben und trotzdem werden jeweils zwei benachbarte Länder dabei so mit verschiedenen Farben gefärbt,dass die gemeinsame Grenze sichtbar ist.

z.B.:

Land	Farbe;
1	rot
2	grün
3	blau
4	gelb
5	rot

Für den K_5-Graph bedeutet das,dass man zur Färbung seiner Knoten mit 4 Farben auskommt,ohne dass zwei durch eine Kante benachbarte Knoten mit der gleichen Farbe gefärbt sind.

Das Problem läßt sich sofort auf allgemeine Landkarten mit zusammenhängenden Gebieten erweitern.Zu jeder dieser Landkarten läßt sich ein dualer Graph wie oben konstruieren,indem die Hauptstädte als Knoten und Verbindungsstraßen zwischen den Hauptstädten als Kanten aufgefaßt werden.

Es ergibt sich dann die allgemeine Frage nach der kleinsten Anzahl von Farben,mit der sich diese Karten bzw. Graphen färben lassen.

C Definition IV.3:

Unter chromatischen Zahl eines Graphen versteht man die kleinste Anzahl von Farben,mit der sich die Knoten dieses Graphen färben lassen,so das jeweils durch eine Kante verbundene Knoten des Graphen verschiedene Farben besitzen.

Es ist also sinnvoll,einen Algorithmus zu entwickeln,der jeweils zu einer vorgegebenen Zahl von Farben entscheidet,ob der Graph mit dieser Anzahl von Farben so gefärbt werden kann,dass jeweils zwei durch eine Kante verbundene Knoten verschiedene Farben besitzen und die Farbverteilung der Knoten ermittelt.Durch Variation der Zahl der Farben,läßt sich dann auch die chromatische Zahl bestimmen.

Algorithmus Färbbarkeit eines Graphen:

(Unit UMath1)

Der Algorithmus beruht auf einem Backtracking-Verfahren mit der Knotennummerierung als Stufenzahl.Jeder Knoten des Graphen besitzt eine Nummer der Reihenfolge,in der er in den Graph eingefügt wurde,nämlich die Position in der Knotenliste.Diese Position sei im folgenden mit Index bezeichnet.Die Nummerierung beginnt mit 0 endet mit der Zahl Anzahl-1 der Knotenliste.

Außerdem wird jedem Knoten vom Typ TFarbknoten als Nachkomme von TInhaltsknoten eine Knotenfarbe in Form einer Integerzahl zugeordnet gemäß der folgenden Typ-Deklaration:

```
TFarbknoten = class(TInhaltsknoten)
  private
    Knotenfarbe_:Integer;
    Ergebnis_:string;
    procedure SetzeFarbzahl(Fa:Integer);
    function WelcheFarbzahl:Integer;
    procedure SetzeErgebnis(S:string);
    function WelchesErgebnis:string;
  public
    constructor Create;
    property Knotenfarbe:Integer read WelcheFarbzahl write SetzeFarbzahl;
    property Ergebnis:string read WelchesErgebnis write SetzeErgebnis;
    function Wertlisteschreiben:TStringlist;override;
    procedure Wertlistelesen;override;
  end;
```

Die Werte für die Knotenfarbe werden bestimmt durch einen vorgegebenen Wert der maximalen Anzahl der Farben (AnzahlFarben).Die (Property) Knotenfarbe (mit dem Feld Knotenfarbe_) der einzelnen Knoten wird bei dem Algorithmus jeweils von 0 ausgehend in Schritten um 1 erhöht.Wenn AnzahlFarben erreicht wird,wird die Knotenfarbe bei der nächsten Erhöhung wieder auf 0 zurückgesetzt.
Das Feld Ergebnis dient dazu,um das Ergebnis,das ein (Gesamt-)string aus dem Knoten-Wert (Feld Inhalt_) (Bezeichnung des Knoten) und der (in einen string umgewandelten) Integer-Zahl Knotenfarbe ist,aufzunehmen.

Verbale Beschreibung:

I)Zu Beginn des Algorithmus werden alle Knotenfarben auf 0 gesetzt.

Knotenindex:	0	1	2	3	4	5	6
Knotenfarbe:	0	0	0	0	0	0	0

I)Setze den Index auf 0.

Rufe mit Index als Stufenparameter folgende Backtrackingmethode auf:

Backtrackingprocedure:

II)Erhöhe die Knotenfarbe des Knoten mit der Nummer Index um Eins modulo Anzahl der Farben+1.

III)Wenn die Knotenfarbe nach der Erhöhung (wieder) 0 ist (dann wurden alle Farben untersucht),dann verlasse die aktuelle Rekursionsebene der Backtrackingprocedure (und setze die Rekursion in der vorigen Ebene mit der vorigen Stufenzahl (Index) fort.Wenn die Ebene die Ausgangs-Rekursionsebene ist,dann beende den Algorithmus.)

IV)Überprüfe ob,die Nachbarknoten des Knotens mit der Nummer Index (d.h. die Knoten,die durch Kanten mit dem Knoten verbunden sind) die gleiche Farbzahl haben.

a)Wenn kein Nachbarknoten die gleiche Knotenfarbe hat und wenn Index gleich Anzahl-1 ist,dann ist eine Farbverteilung gefunden.Bilde dann das Ergebnis und gib die Farbverteilung aus.Setze dann das Verfahren bei Schritt II fort.
(Wenn nur eine Lösung gesucht wird,verlasse an dieser Stelle alle Rekursionsebenen und beende den Algorithmus.)

b)Wenn kein Nachbarknoten die gleiche Farbe hat und Index kleiner als Anzahl-1 ist,dann rufe die Backtrackingmethode (d.h. die nächste Rekursionsebene mit dem um 1 erhöhten Index als Stufenzahl auf und setze dort den Ablauf bei Schritt II fort.
(Dann Backtrack!)

c)Wenn mindestens ein Nachbarknoten die gleiche Farbe hat,dann setze den Algorithmus bei Schritt II innerhalb derselben Rekursionsebene fort.

Bei dem geschilderten Verfahren werden gerade alle möglichen Farbverteilungen gefunden.

Der Algorithmus kann als Backtracking-Verfahren mit geeigneter Abbruchbedingung rekursiv programmiert werden,indem bei Schritt IV) b) die Methode sich selber mit der um 1 erhöhten Index-Zahl selbst wieder aufruft.

Die folgende Tabelle gibt zur Veranschaulichung einen möglichen momentanen Zustand der Farbverteilung während das Ablauf des Algorithmus wieder.Das Backtracking-Verfahren hat dabei die Knoten 4 bis 6 noch nicht erreicht.

Knotenindex:	0	1	2	3	4	5	6
Knotenfarbe:	3	4	2	1	0	0	0

Als Datenstruktur für den Graphen wird TFarbgraph als ein Nachkomme von TInhaltsgraph mit folgenden Methoden verwendet:

Quellcode:

```
TFarbgraph = class(TInhaltsgraph)
    constructor Create;
    procedure SetzebeiallenKnotenAnfangsfarbe;
    function Knotenistzufaerben(Index:Integer;AnzahlFarben:Integer):Boolean;
    procedure Farbverteilung(Index:Integer;AnzahlFarben:Integer;var Gefunden:Boolean;
      EineLoesung:Boolean;Flaeche:TCanvas;Ausgabe:TLabel;var Ausgabeliste:TStringlist);
    procedure ErzeugeErgebnis;
    procedure FaerbeGraph(Flaeche:TCanvas;Ausgabe:TLabel;var SListe:TStringlist);
  end;
```

Der Constructor ruft die Vorgängermethode auf und registriert die im Graphen verwendeten Datentypen für Knoten und Kanten:

```
constructor TFarbgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TFarbknoten;
  Inhaltskanteclass:=TInhaltskante;
end;
```

Die Methode SetzebeiallenKnotenAnfangsfarbe setzt bei allen Knoten des Graphen die Farbzahl auf 0:

```
procedure TFarbgraph.SetzebeiallenKnotenAnfangsfarbe;
var Index:Integer;
begin
  if not Leer then
  for Index:=0 to Knotenliste.Anzahl -1 do
    TFarbknoten(Knotenliste.Knoten(Index)).Knotenfarbe:=0;
end;
```

Die Function Knotenistzufaerben erhöht die Farbzahl des Knotens mit der Nummer Index um 1 und überprüft,ob die Nachbarknoten dieses Knoten dieselbe Farbzahl wie er selber haben.

Der Rückgabewert dieser Function ist false,wenn mindestens ein Nachbarknoten dieselbe Farbe wie der untersuchte Knoten hat oder die (um 1 erhöhte) Farbzahl 0 ist.Ansonsten ist der Rückgabewert true.

```
function TFarbgraph.Knotenistzufaerben(Index:Integer;AnzahlFarben:Integer):Boolean;
var Kno:TFarbknoten;
    GleicheFarbe:Boolean;
    Zaehl:Integer;

  function NachbarknotenhabengleicheFarbe(Ka:TKante):Boolean;
  var Kno1,Kno2:TFarbknoten;
  begin
    Kno1:=TFarbknoten(Ka.Anfangsknoten);
    Kno2:=TFarbknoten(Ka.Endknoten);
    if Ka.KanteistSchlinge                                                    6)
    then
      NachbarknotenhabengleicheFarbe:=false                                   7)
    else
      NachbarknotenhabengleicheFarbe:=(Kno1.Knotenfarbe=Kno2.Knotenfarbe);    8)
  end;

begin
  Kno:=TFarbknoten(Knotenliste.Knoten(Index));                                1)
  repeat
    Kno.Knotenfarbe:=(Kno.Knotenfarbe+1) mod (AnzahlFarben+1);                2)
    if Kno.Knotenfarbe=0                                                      3)
    then
    begin
      Knotenistzufaerben:=false;
      exit;
    end;
    GleicheFarbe:=false;
    if not Kno.AusgehendeKantenliste.Leer  then
      for Zaehl:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
        if  NachbarknotenhabengleicheFarbe(Kno.AusgehendeKantenliste.Kante(Zaehl)) then
          GleicheFarbe:=true;                                                 4)
    if not Kno.EingehendeKantenliste.leer  then
      for Zaehl:=0 to Kno.EingehendeKantenliste.Anzahl-1 do
        if NachbarknotenhabengleicheFarbe(Kno.EingehendeKantenliste.Kante(Zaehl)) then
          GleicheFarbe:=true;                                                 5)
  until (Not GleicheFarbe) ;
  KnotenistzuFaerben:=true;
end;
```

Bei 1) wird der Knoten mit Nummer Index ausgewählt und bei 2) wird die Knotenfarbe um 1 erhöht oder auf Null gesetzt.Wenn die Knotenfarbe gleich 0 ist,ist der Rückgabewert der Function false (3).Ansonsten werden die Nachbarknoten bei 4) und 5) auf gleiche Farbzahl untersucht.Wenn keiner der Knoten die gleiche Farbe hat,ist der Rückgabewert der Function true, ansonsten false.

Die Methode greift,um die Farben zweier Knoten einer Kante zu vergleichen, auf die Function NachbarknotenhabengleicheFarbe zurück.(Die Kanten,die untersucht werden,sind die Kanten zu den Nachbarknoten des aktuellen Knoten.) Wenn die dort untersuchte Kante eine Schlinge ist (6),ist der Rückgabewert dieser Function false (7),ebenso wenn die Farbzahl von Anfangs- und Endknoten dieser Kante gleich sind (8).Ansonsten ist der Rückgabewert true (8).

```
procedure TFarbgraph.Farbverteilung(Index:Integer;AnzahlFarben:Integer;var Gefunden:Boolean;
  EineLoesung:Boolean;Flaeche:TCanvas;Ausgabe:TLabel;var Ausgabeliste:TStringlist);
label Endproc;
var Knotenzufaerben:Boolean;
    K:TFarbgraph;
    Zaehl:Integer;
    Kno:TFarbknoten;
    S:string;

  function Farbe(F:Integer):TColor;
  begin
```

```
    case F of

      1:Farbe:=clred;
      2:Farbe:=clblue;
      3:Farbe:=clgreen;
      4:Farbe:=clgray;
      5:Farbe:=clpurple;
      6:Farbe:=clmaroon;
      7:Farbe:=claqua;
      8:Farbe:=clyellow;
      9:Farbe:=clnavy;
      10:Farbe:=clteal;
      11:Farbe:=cllime;
      12:Farbe:=clfuchsia;
      else Farbe:=clblacK;
    end;
  end;

begin
  if Gefunden and EineLoesung then goto Endproc;
  repeat
    if Abbruch then  exit;
    Knotenzufaerben:=Knotenistzufaerben(Index,AnzahlFarben);                9)
    if (Knotenzufaerben) and (Index<AnzahlKnoten-1)                        10)
    then
      Farbverteilung(Index+1,AnzahlFarben,Gefunden,EineLoesung,Flaeche,
        Ausgabe,Ausgabeliste)
    else
      if (Index=Anzahlknoten-1) and Knotenzufaerben then
      begin
        Gefunden:=true;
        ErzeugeErgebnis;                                                   12)
        S:='';
        for Zaehl:=0 to Knotenliste.Anzahl-1 do                            13)
        begin
          Kno:=TFarbknoten(Knotenliste.Knoten(zaehl));
          Kno.Farbe:=Farbe(Kno.Knotenfarbe);
          S:=S+' ' +Kno.Ergebnis;
        end;
        Knotenwertposition:=2;
        ZeichneGraph(Flaeche);                                             14)
        Demopause;
        Knotenwertposition:=0;
        Ausgabeliste.Add(S);                                               15)
        Ausgabe.Caption:=S;
      end
  until (not Knotenzufaerben) or (Gefunden and EineLoesung)                11)
  Endproc:
end;
```

Erläuterung des Algorithmus:

Die Methode Farbverteilung ist die Backtracking-Methode.Sie wird anfangs mit dem Wert für Index=0 als Stufenzahl aufgerufen.

Sie testet bei 9) mit Hilfe der eben besprochenen Function,ob der dieser Nummer entsprechende Knoten mit der um 1 erhöhten Farbzahl zu färben ist.Wenn dies der Fall ist,wird bei 10) die nächste Rekursionsebene dieser Methode mit der um 1 erhöhten Stufenzahl Index (d.h. mit dem nächsten Knoten) aufgerufen.(Backtracking-Schritt)

Die letzten beiden Schritte werden solange wiederholt,bis gerade entweder der letzte Knoten (Index=AnzahlKnoten-1) untersucht wurde,der Knoten nicht zu färben war (Knotenfarbe ist wieder 0) oder aber,wenn nur eine Lösung gesucht wird,eine Lösung schon gefunden wurde.

Wenn der Knoten zu färben war und es der letzte untersuchte Knoten war,wird bei 11) gefunden auf true gesetzt,bei 12) wird das Ergebnis erzeugt und bei 13) wird mit Hilfe der Function Farbe die Knoten gemäß ihrer Zahl Knotenfarbe gefärbt.

Bei 14) wird der gefärbte Graph gezeichnet,bei 15) wird die Ausgabeliste für das Ausgabefenster mit den Ergebnissen erzeugt und bei 15) wird die Ausgabe für das Label Ausgabe gesetzt.

```
procedure TFarbgraph.ErzeugeErgebnis;
var Index:Integer;
    Kno:TFarbknoten;
begin
  if not Leer then
  for Index:=0 to Knotenliste.Anzahl-1 do
  begin
    Kno:=TFarbknoten(Knotenliste.Knoten(Index));
    Kno.SetzeErgebnis(Kno.Wert+':'+IntegertoString(Kno.Knotenfarbe));                   16)
  end;
end;
```

Die Methode erzeugt bei 16) im Feld Ergebnis jedes Knotens einen String aus der Knotenbezeichnung und der Farbzahl,der bei der Zeichnung des Graphen (14) angezeigt wird.

```
procedure TFarbgraph.FaerbeGraph(Flaeche:TCanvas;Ausgabe:TLabel;var SListe:TStringlist);
var AnzahlFarben:Integer;
    StringFarben:string;
    Gefunden,EineLoesung:Boolean;
    Index:Integer;
begin
  if Leer then exit;
  SetzebeiallenKnotenAnfangsfarbe;
  repeat
    StringFarben:=Inputbox('Eingabe Farbzahl','Anzahl Farben:','4');
    AnzahlFarben:=StringtoInteger(StringFarben);
    if (AnzahlFarben<0) or (AnzahlFarben>19) then
      ShowMessage('Fehler: 0<Anzahl Farben <20 !');
  until (AnzahlFarben>0) and (AnzahlFarben<20);
  if MessageDlg('Nur eine Lösung?',
       mtConfirmation, [mbYes, mbNo], 0) = mrYes
  then
    EineLoesung:=true
  else
    EineLoesung:=false;
  Gefunden:=false;
  Farbverteilung(0,AnzahlFarben,Gefunden,EineLoesung,Flaeche,Ausgabe,SListe);
end;
```

Die Backtrackingmethode Farbverteilung wird dann z.B. wie in der Methode FaerbeGraph durch

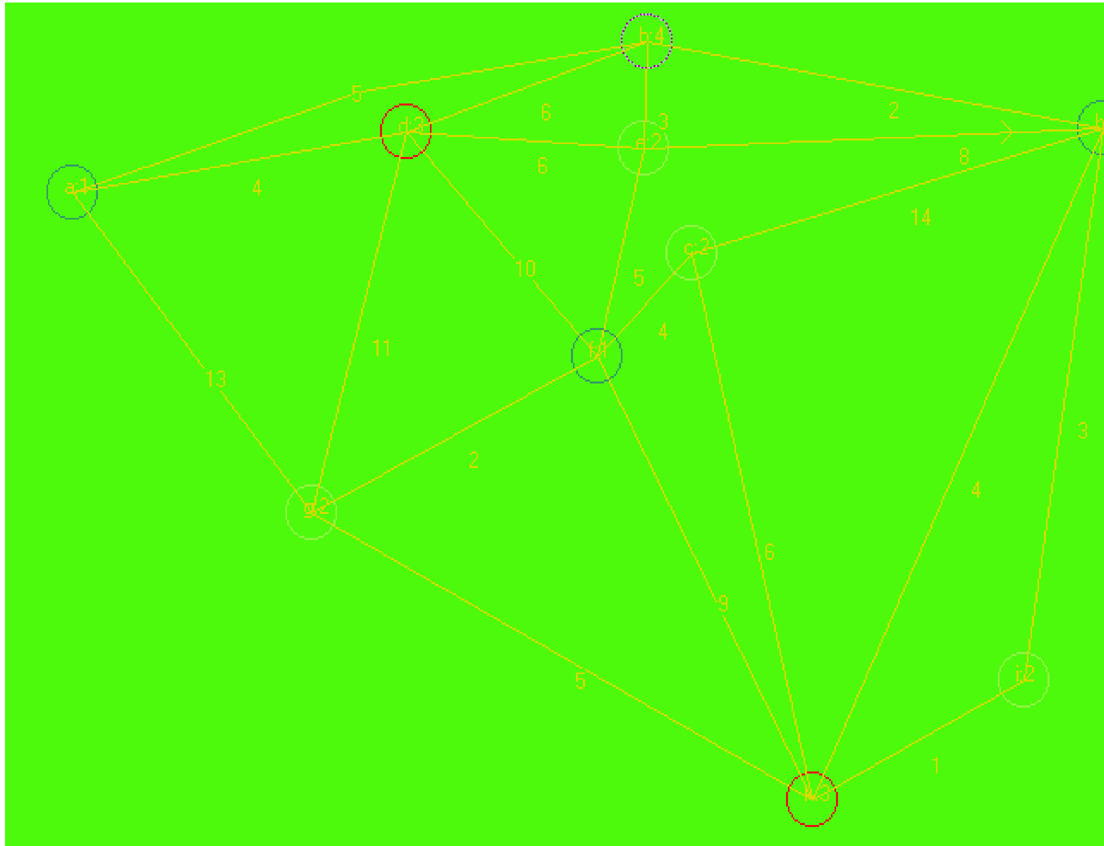
```
if MessageDlg('Nur eine Lösung?',
       mtConfirmation, [mbYes, mbNo], 0) = mrYes
  then
    EineLoesung:=true
  else
    EineLoesung:=false;
  Gefunden:=false;
  Farbverteilung(0,AnzahlFarben,Gefunden,EineLoesung,Flaeche,Ausgabe,SListe);
  ErzeugeErgebnis;
```

aufgerufen,nachdem die maximale Anzahl der Farben AnzahlFarben und der Wert für EineLoesung vorgegeben wurden.

C Aufgabe IV.6:

Erzeuge mit Hilfe von Knotengraph verschiedene ebene Graphen (z.B. G019.gra) und bestimme mit Hilfe des Menüs Anwendungen/Färbbarkeit des Programm Knotengraphs (bzw. durch den eigenen erstellten Quellcode in der Entwicklungsumgebung) durch Variation der einzugebenden Maximalzahl der Farben die chromatische Zahl.Benutze auch den Demomodus.

Lösung:



G019.gra

Der obige ebene Graph z.B. ist läßt sich mit 4 Farben färben (die Farben sind durch ihre Farbzahlen in den Knoten angegeben),mit 3 Farben jedoch nicht.Also ist die chromatische Zahl 4.
Wenn man die Kante zwischen b und e entfernt,genügen 3 Farben.

Für alle ebenen Graphen gilt,dass die chromatische Zahl maximal 4 ist.

Diese berühmte Aussage heißt Vierfarbenproblem und konnte 1976 (mit Hilfe eines Computerprogramms) allgemein bewiesen werden.

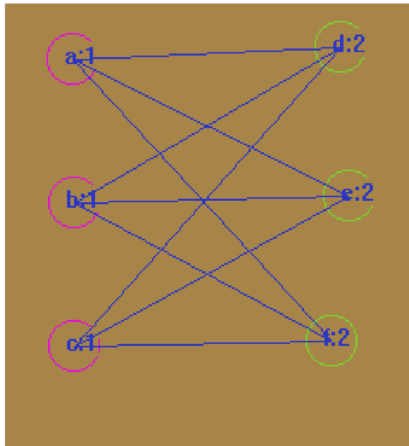
C Satz IV.5:

Die chromatische Zahl jedes planaren Graphen ist kleiner oder gleich 4.

Dies bedeutet,dass jede Landkarte mit maximal 4 Farben gefärbt werden kann.(Ohne,dass zwei benachbarte Länder die gleiche Farbe haben.)

Der Beweis ist natürlich für die Schule zu schwierig und aufwendig und wird deshalb hier nicht vorgestellt.

Der folgende $K_{3,3}$- Graph ist nicht eben,wie oben gezeigt wurde.Bei ihm ist die chromatische Zahl sogar 2:



G020.gra

Der Grund dafür ist leicht zu erkennen.Die Knoten können jeweils in zwei Mengen disjunkt eingeteilt werden,von denen die Elemente der ersten Menge die erste Farbe und die Elemente der zweiten Menge die zweite Farbe haben.Dies gelingt nur dann,wenn zwischen den Knoten der ersten Menge keine Kanten existieren und auch nicht zwischen den Knoten der zweiten Menge.Kanten sind nur jeweils zwischen einem Knoten der ersten und einem Knoten der zweiten Menge gezogen.

Dies gibt Anlaß für folgende Definition:

C Definition IV.4:

Ein Graph heißt paar oder bipartit,wenn er die chromatische Zahl 2 hat.

Paare Graphen sind u.a. bei der Bestimmung eines Matchings in Graphen in Kapitel X und XIII von Bedeutung,weil bei ihnen die Bestimmung eines Matchings besonders leicht ist.

C Aufgabe IV.7:

Erzeuge mit Hilfe von Knotengraph mehrere paare Graphen,die jeweils einen oder mehrere Kreise enthalten.
Ermittle mit Hilfe des Menüs Pfade/Alle Kreise alle Kreise dieser Graphen von einem bestimmten Startknoten aus und ermittle die Anzahl der Kanten dieser Kreise.

(Die Anzahl der Kanten kann als Summe der Pfade direkt abgelesen werden,wenn der Kanteninhalt (Feld Inhalt_) vom Typ string gewählt wird,weil dann jede Kante mit 1 bewertet wird.)

Bestätige,dass die Summe der Kanten aller Kreise gerade ist.

Lösung:

Ergebnis z.B. für den obigen $K_{3,3}$-Graph mit Startknoten a:

a e b d a Summe: 4 Produkt: 1
a e b d c f a Summe: 6 Produkt: 1
a e b f a Summe: 4 Produkt: 1
a e b f c d a Summe: 6 Produkt: 1
a e c d a Summe: 4 Produkt: 1
a e c d b f a Summe: 6 Produkt: 1
a e c f a Summe: 4 Produkt: 1
a e c f b d a Summe: 6 Produkt: 1
a d b e a Summe: 4 Produkt: 1
a d b e c f a Summe: 6 Produkt: 1
a d b f a Summe: 4 Produkt: 1
a d b f c e a Summe: 6 Produkt: 1
a d c e a Summe: 4 Produkt: 1

a d c e b f a Summe: 6 Produkt: 1
a d c f a Summe: 4 Produkt: 1
a d c f b e a Summe: 6 Produkt: 1
a f b d a Summe: 4 Produkt: 1
a f b d c e a Summe: 6 Produkt: 1
a f b e a Summe: 4 Produkt: 1
a f b e c d a Summe: 6 Produkt: 1
a f c d a Summe: 4 Produkt: 1
a f c d b e a Summe: 6 Produkt: 1
a f c e a Summe: 4 Produkt: 1
a f c e b d a Summe: 6 Produkt: 1

Alle Summen sind gerade.Es gilt nämlich der Satz:

C Satz IV.6:

Ein Graph ist paar genau dann,wenn er keine Kreise mit ungerader Kantenzahl besitzt.

Beweis:

Es sei G ein paarer Graph mit der Färbung rot/blau und p ein Kreis in G.Dann folgen in p jeweils abwechselnd rote und blaue Knoten aufeinander,weil nur zwischen diesen gefärbten Knoten Kanten verlaufen.Wenn z.B. mit einem roten Knoten begonnen wurde,endet der Kreis auch wieder bei diesem roten Knoten.Dies geht nur bei gerader Knotenzahl.

Es sei umgekehrt k_0 ein Knoten des Graphen,und die Mengen M_1 sowie M_2 seien dadurch definiert,dass M_1 alle Knoten des Graphen enthält,die auf kürzestem Pfad mit gerader Kantenzahl von k_0 aus erreichbar sind und bei M_2 mit ungerader Kantenzahl.Da man ohne Beschränkung der Allgemeinheit G als zuammenhängend annehmen kann (sonst wird jede Komponente von G einzeln betrachtet),läßt sich jeder Knoten von G eindeutig einer der beiden Mengen bei einem ungerichteten Graphen zuordnen.
Es wird jetzt gezeigt,dass innerhalb der Mengen M_1 und M_2 keine Kanten zwischen Knoten vorhanden sein können,so dass damit ein paarer Graph hergestellt ist.
Es seien nämlich k_x und k_y zwei Elemente aus M_1.Dann gibt es zwei kürzeste Pfade mit gerader Anzahl von Kanten zu k_x und k_y von k_0 aus.Wenn diese Pfade gemeinsame Knoten enthalten sollten,sei k_1 als der letzte gemeinsame Knoten bezeichnet.
Wenn nun der Pfad von k_0 nach k_1 von gerader Kantenzahl ist,müssen auch die Pfade von k_1 nach k_x bzw nach k_y gerade sein.Wenn der Pfad von k_0 nach k_1 ungerade ist,so sind die Pfade von k_1 nach k_x bzw. nach k_y ungerade.
Die Summe der Zahl der Kanten von k_1 nach k_x und von k_1 nach k_y ist dann jedoch von gerader Kantenzahl.
Wenn nun eine Kante a zwischen k_x und k_y existieren sollte,so liegt ein Kreis von k_1 nach k_1 über die Knoten k_x und k_y längs der Kante a vor.
Da die Kante a zu der letzten Summe hinzugezählt werden muß,entsteht ein Kreis von ungerader Kantenzahl.
Das ist ein Widerspruch zur Voraussetzung,dass es keine Kreise ungerader Kantenzahl gibt.

C V Euler-und Hamiltonlinien

Eine Unterrichtsreihe zum Thema Pfade sollte unbedingt auch die wichtigen verwandten Euler- und Hamiltonlinien behandeln.Insofern stellt der Inhalt dieses Kapitels die konsequente Fortsetzung der Kapitel C II und C III dar.

Man kann jedoch auch direkt in einer Unterrichtsreihe mit der Behandlung der Euler-und Hamiltonlinien oder auch nur eines dieser beiden Themen beginnen,da Vorkenntnisse aus den Kapiteln C II und C III nicht unbedingt erforderlich sind.

Die Algorithmen zu den Euler- bzw. Hamiltonlinien sind als in der Struktur zueinander sehr ähnliche Backtrackingalgorithmen gestaltet,da die Aufgabenstellungen formal sehr verwandt sind.

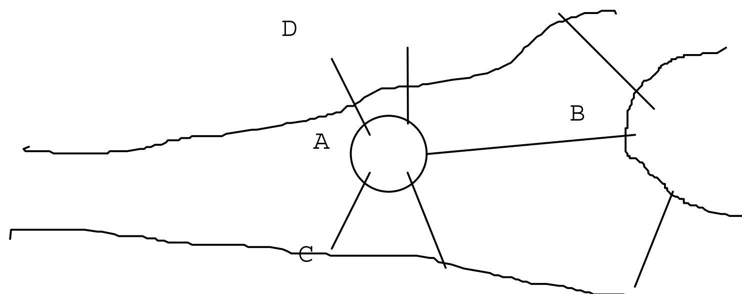
Man braucht nur jeweils die Bedeutung von Kanten und Knoten gegeneinander auszutauschen,um den analogen Algorithmus zu erhalten.Deshalb bietet es sich an,einen der Algorithmen als vom Lehrer gelenktes Unterrichtsprojekt zu realisieren,während der andere Algorithmus ganz alleine von den Schülern bearbeitet werden kann.Auch ist es möglich das jeweils andere Verfahren als Klausuraufgabe zu stellen.

Der im folgenden vorgestellte Unterrichtsgang beginnt mit dem Euler-Problem,das historisch als der Beginn der Graphentheorie gilt.An Hand dieses Problems läßt sich nämlich mittels des Programms Knotengraph das bekannte Kriterium (gerader Knotengrad) für die Existenz von Eulerlinien sehr anschaulich darstellen,so dass es von Schülern einschließlich Begründungsmöglichkeit selbstständig gefunden werden kann.(Für Hamiltonlinien existiert bekanntlich ein solches einfaches Kriterium nicht.)

Es empfiehlt sich also in einer Unterrichtsreihe nicht direkt mit der Programmierung der Algorithmen zu beginnen,sondern zunächst das Problem an Hand der folgenden Einführungsaufgaben zu behandeln.Der Übergang zum Hamilton-Problem erfolgt dann durch Analogiebetrachtung auf Grund des Austausches von Kanten und Knoten.

(Vgl. zu diesem Thema auch den entsprechenden Unterrichtplan im Anhang sowie die Bemerkungen im Kapitel B III,Abschnitt 5,wo die Reihenfolge Eulerpfade-Hamiltonlinien vertauscht ist.)

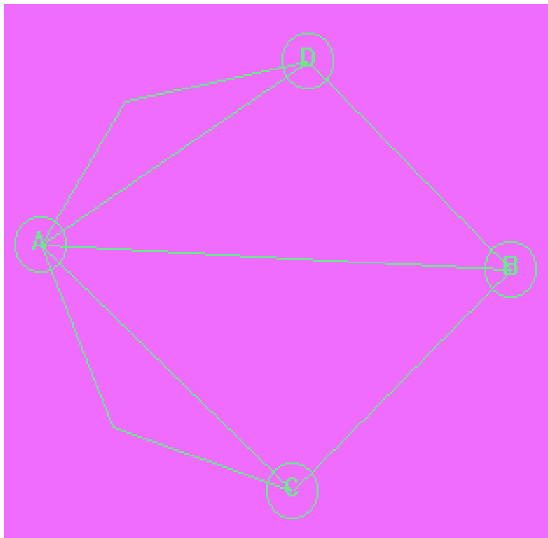
C Aufgabe V.1: (Einstiegsproblem)



Von dem Mathematiker Euler wurde 1736 die Frage gestellt,ob es über die 7 Brücken in Königsberg,die über den alten und neuen Pregel sowie zu einer im Zusammenfluss der beiden Flüsse gelegenen Insel führen,ein Spaziergang möglich ist,bei dem jede Brücke nur einmal überquert wird und der zum Ausgangspunkt zurückführt.

Werden die Gebiete,die mit A,B,C und D bezeichnet werden als Knoten aufgefaßt und die Brücken als Kanten zwischen ihnen,ergibt sich folgender Graph:

Graph 1:

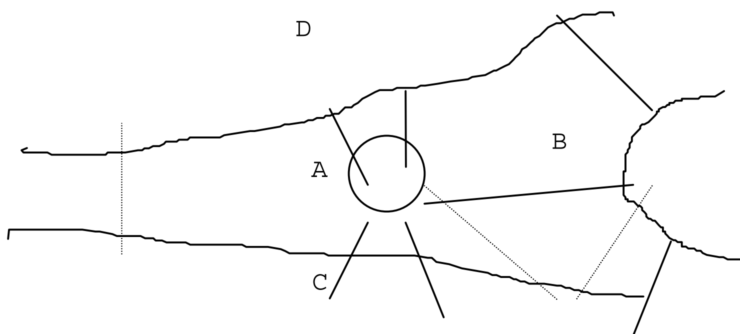


G021.gra

In dem obigen Graphen ist also eine geschlossene Linie gesucht,die alle Kanten genau einmal enthält.

Gibt außerdem es eine offene Linie,bei der der Spaziergang zwar nicht wieder zum selben Ausgangspunkt (Knoten) zurückführt,die aber wiederum alle Brücken (Kanten) genau einmal enthält?

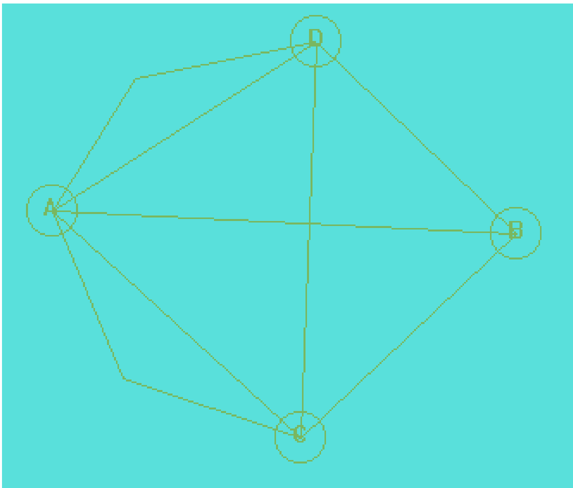
Es werden nacheinander (von links nach rechts) 3 Brücken zusätzlich gebaut,die in dem folgenden Plan gestrichelt eingetragen sind.Den Ausbaustufen entsprechen jeweils die folgenden drei Graphen.



Untersuche (zuerst per Hand) auch,ob für die folgenden Graphen,die den Ausbaustufen entsprechen,geschlossene bzw. offene Linien mit den oben genannten Eigenschaften existieren.

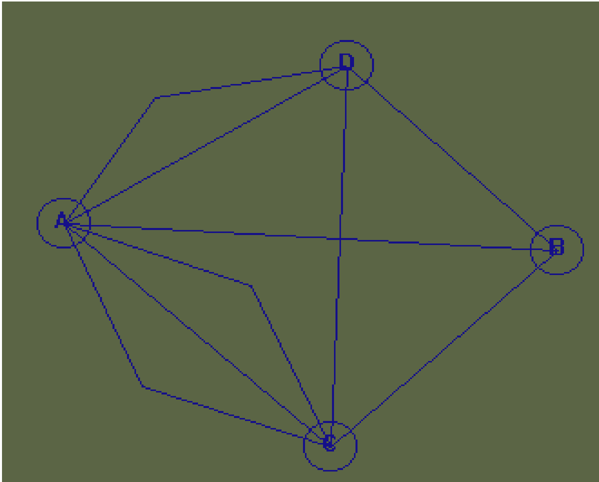
Überprüfe dann alle Ergebnisse mit Hilfe des Programms Knotengraph mit Hilfe der Menüs Anwendungen/Eulerlinie (geschlossen) oder Anwendungen/Eulerlinie (offen) bzw. Eigenschaften/Eulerlinie.

Graph 2:



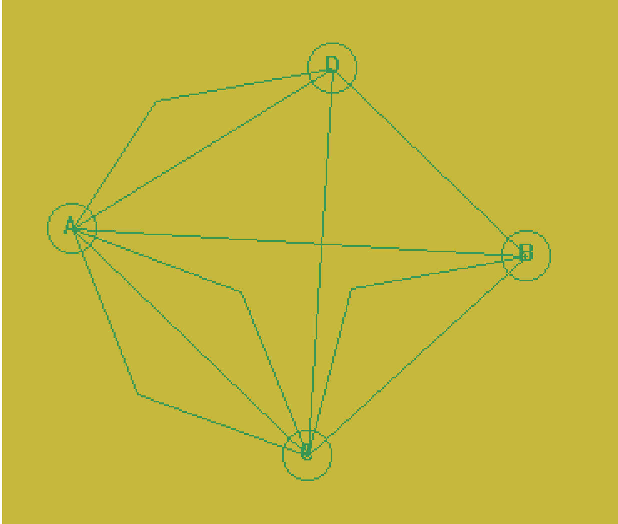
G022.gra

Graph 3:



G023.gra

Graph 4:



G024.gra

Lösungen:

Graph 1:

Es zeigt sich,dass es für beide Probleme keine Lösung gibt.

Graph 2:

Es gibt keinen geschlossene Linie aber eine offene Linie zwischen A und B: ACDABCDB

Graph 3:

Es gibt keine geschlossene Linie aber eine offene Linie zwischen B und C: BACADACBDC

Graph 4:

Es gibt keine offene aber eine geschlossene Linie z.B.: ACABABCBDCA

C Definition V.1:

Gibt es in einem Graphen G einen geschlossenen Kantenzug p,der jede Kante von G genau einmal enthält,heißt p geschlossene Eulerlinie von G und G ein Eulerscher Graph.

Gibt es in einem Graphen G einen offenen Kantenzug p zwischen zwei verschiedenen Knoten von G,der jede Kante von G genau einmal enthält,so heißt p eine offene Eulerlinie von G.

Woran kann man nun bei einem Graph erkennen,ob er eine geschlossenen Eulerlinie bzw. eine offene Eulerlinie besitzt?Eine notwendige Bedingung ist sicher,dass der Graph zusammenhängend ist.Um weitere Bedingungen für eine geschlossene Eulerlinie zu erkennen soll die nächste Anwendungsaufgabe gelöst werden.

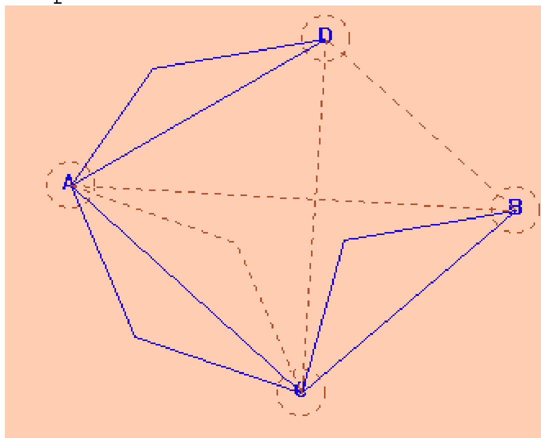
C Aufgabe V.2: Teil 1 (2. Einstiegsproblem)

Gegeben sind die beiden folgenden Graphen I und II (Graph I ist der obige Graph 4).Beide Graphen enthalten eine geschlossene Eulerlinie.

Erzeuge dazu Graph I und II mit Knotengraph,und überprüfe die Existenz der Eulerlinie mittels des Menüs Anwendungen/Eulerlinie (geschlossen).

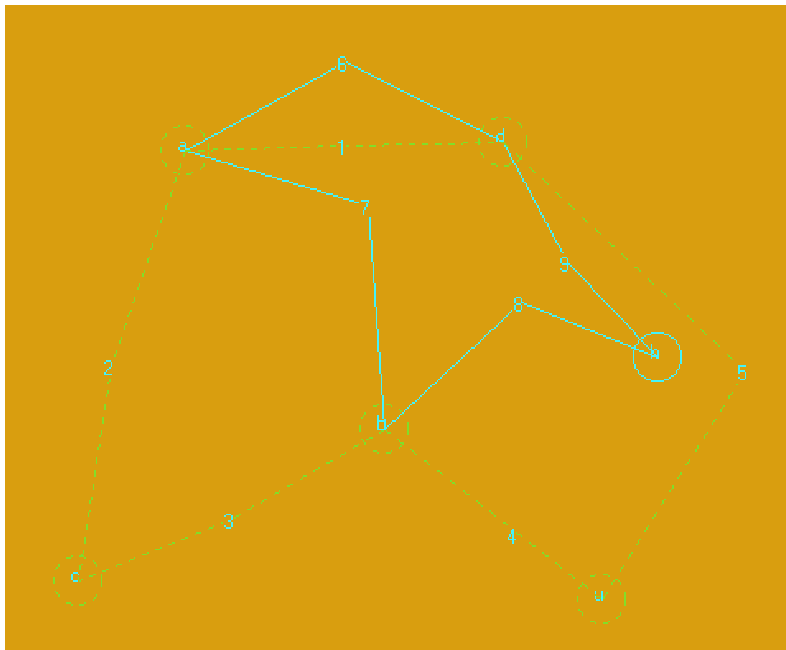
Erzeuge dann mit Hilfe des Menüs Pfade/Alle Kreise die Kreise in beiden Graphen von Knoten A aus.Der letzterzeugte Kreis wird auf der Zeichenfläche rot markiert angezeigt:

Graph I:



G024.gra

Graph II:



G025.gra

Lösche bei beiden Graphen die Kanten dieses Kreises.

Bei Graph 2 bleiben dann die Kanten 6,7,8,9 übrig,die wiederum beim Erzeugen aller Kreise einen rot-markierten Kreis bilden und wiederum gelöscht werden können.Dann sind alle Kanten des Graphen gelöscht.

Bei Graph 1 kann das Verfahren noch dreimal nämlich mit den Kreisen aus den Parallelkanten zwischen den Knoten A und C,B und C sowie B und D durchgeführt werden.(Man vergesse dabei nicht als Anfangsknoten beim Erzeugen der Kreise einen nichtisolierten Knoten durch Mausklick zu wählen.)

In beiden Fällen können jeweils die Kanten eines kompletten Kreises gelöscht werden bis alle Kreise gelöscht sind.

Das heißt,dass der Graph jeweils aus Kreisen mit disjunkten Kantenmengen besteht,wenn eine geschlossene Eulerlinie existiert.

Dies bedeutet,dass durch jeden Knoten jeweils ein oder mehrere voneinander unabhängige Kreise verlaufen müssen.Daher muß die von diesem Knoten ausgehende Kantenzahl 2,4,6...usw. Kanten betragen,d.h. eine gerade Zahl sein.Anders ausgedrückt:Der (ungerichtete) Knotengrad ist gerade.

Erzeuge weitere Graphen,in denen geschlossene Eulerlinien existieren,und überprüfe die obige Aussage nach dem eben beschriebenen Verfahren.

Lasse außerdem durch Knotengraph mittels des Menüs Eigenschaften/Knoten zeigen die (ungerichteten) Knotengrade anzeigen und überprüfe die Aussage.

Es gilt der folgende Satz:

Satz C V.1:

Äquivalent sind:

1)G ist ein Eulerscher zusammenhängender,ungerichteter Graph.
2)Jeder Knoten von G hat einen (ungerichteten) geraden Grad.

3)Die Kanten des Kreises lassen in disjunkte Kantenmengen aufteilen,die jeweils einen Kreis bilden.

Zusatz:

Wenn G ein Graph mit gerichteten Kanten ist,existiert genau dann eine geschlossene Eulerlinie,wenn alle Knoten vom (gerichteten) Knotengrad 0 sind.

Beweis:

1)⇒2):

Wenn G ein Eulerscher Graph ist,muß zu jeder Kante die auf der Eulerlinie zu einem Knoten führt,eine zweite noch nicht besuchte Kante gehören,die von dem Knoten wieder wegführt.

Deshalb kommen diese Kanten bei jedem Knoten paarweise vor.Der Knotengrad ist also gerade.

2)⇒3):

Es gibt dann in G sicher einen Kreis.Da der Knotengrad gerade ist,kann man jeder Kante eines Knoten eine Partnerkante zuordnen.Denn wenn man bei einem Knoten startet und jeweils einen Pfad aufbaut,indem man zu einem Knoten durch eine Kante gelangt,kann man ihn durch die zu dieser Kante gehörende zweite Partnerkante wieder verlassen.Da es nur endlich viele Kanten gibt,muß dieses Verfahren nach endlich vielen Schritten in einem schon besuchten Knoten enden.

Die Kanten dieses Kreises werden aus dem Graph entfernt.Danach wird das Verfahren erneut angewendet.Nach endlich vielen Schritten sind so alle zueinander disjunkten Kreise gefunden.(Die Knoten bleiben als isolierte Knoten zurück.)

3)⇒1):

Es gibt dann sicher einen Kreis in G.Ist dies schon die Eulerlinie,ist man fertig.Anderfalls muß es wegen des Zusammenhangs von G einen Knoten k_1 geben,durch den ein 2. Kreis mit disjunkter Kantenmenge verläuft.Dann durchläuft man von k_1 aus zuerst den 2. und verfolgt dann den 1. Kreis weiter.Dieses Verfahren setzt man weiter mit den anderen Kreisen mit disjunkter Kantenmenge fort,bis alle Kanten des Graphen einbezogen sind.Diese Linie ist dann die geschlossene Eulerlinie.

Zusatz:

Es ist leicht einzusehen,dass im Falle eines gerichteten Graphen die Anzahl der von einem Knoten ausgehenden Kanten gleich der Anzahl der einlaufenden Kanten sein muß,damit eine Eulerlinie existiert.Der Knotengrad eines solchen Knotens ist dann gleich Null.Der obige Beweis läßt sich dann in gleicher Weise führen.

C Aufgabe V.3:

Untersuche die Knotengrade der Graphen 2 und 3 der Aufgabe V.1 mittels des Menüs Eigenschaften/Knoten zeigen von Knotengraph.Diese Graphen enthielten eine offene Eulerlinie.

Füge danach eine weitere Kante bei Graph 2 zwischen A und B und bei Graph 3 zwischen B und C ein.Überprüfe mit dem Menü Eulerlinie (geschlossen),ob die jetzt entstandenen Graphen eine geschlossene Eulerlinie haben.

Lösung:

A und B haben bei Graph 2 und B und C haben bei Graph 3 ungeraden Knotengrad.

Wenn zusätzlich eine Kante zwischen den Knoten mit ungeradem Knotengrad eingefügt wird,haben die Graphen eine geschlossenen Eulerlinie.

Es liegt die Folgerung nahe,dass wenn genau zwei Knoten des Graphen einen ungeraden Knotengrad haben (und alle anderen geraden Grad) diese beiden Knoten Start-und Zielpunkt einer offenen Eulerlinie sind.Also gilt,dass wenn genau zwei Knoten des Graphen im ungerichteten Fall einen ungeraden Knotengrad haben,der Graph eine offene Eulerlinie enthält.

Satz V.2:

Ein ungerichteter,zusammenhängender Graph hat genau dann eine offene Eulerlinie zwischen zwei verschiedenen Knoten des Graphen,wenn genau diese beiden Knoten des Graphen ungeraden Knotengrad und alle anderen Knoten geraden Grad haben.

Zusatz:

Wenn alle Kanten des Graphen gerichtet sind,dann hat der Graph genau dann eine offene Eulerlinie zwischen zwei verschiedenen Knoten des Graphen,wenn diese beiden Knoten jeweils einen (gerichteten) Knotengrad ungleich 0 (1 und -1) besitzen, und die anderen Knoten jeweils vom (gerichteten) Grad 0 sind.

Beweis:

Zwischen den Knoten mit ungeradem Knotengrad sei eine zusätzliche Kante gezogen.Dann haben alle Knoten geraden Knotengrad,und es existiert nach Satz V.1 eine geschlossene Eulerlinie,die auch durch die neu eingefügte Kante verlaufen muß.Entfernt man diese Kante wieder aus dem Graph und der geschlossenen Eulerlinie,entsteht die gesuchte offene Eulerlinie zwischen den Knoten ungeraden Grades.

Nach Satz I.2 ist die Anzahl der Knoten ungeraden Grades eine gerade Zahl.Deshalb gibt es mindestens zwei Knoten ungeraden Grades.

Wenn es umgekehrt mehr als zwei Knoten ungeraden Grades (im gerichteten Fall ungleich 0 mit -1 und +1) geben sollte,so muß es einen Knoten ungeraden Grades (gerichteter Fall ungleich 0 (-1 oder +1)) geben,der nicht Anfangs-oder Endknoten der offenen Eulerlinie ist.Bei anderen Knoten des Graphen müssen aber die Kanten paarweise vorkommen (vgl. Beweis von Satz V.1 /ein-und auslaufende Kante),so dass diese Knoten nur von geradem Grad sein können.

Das ist ein Widerspruch.

Zusatz:

Bei einem gerichteten Graph muß,damit eine zusätzliche Kante eingefügt werden kann,der eine Knoten von (gerichtetem) Knotengrad (ungleich 0) 1 und der andere von -1 sein.Nach dem Einfügen der Kante,haben dann beide Knoten den (gerichteten) Grad 0.
Danach verläuft der Beweis wie oben angegeben.

Bemerkung:

Mit Hilfe des Kriteriums des Knotengrades wird im Menü Eigenschaften/Eulerlinie des Programms Knotengraph oder der Entwicklungsumgebung die

Existenz von (geschlossenen oder offenen) Eulerlinien für ungerichtete und gerichtete Graphen überprüft.

Eine Folgerung aus Satz V.1 ist:

C Satz V.3:

Jeder beliebige zusammenhängende, ungerichtete Graph enthält stets einen Kantenzug,bei dem jede Kante genau zweimal durchlaufen wird.

Beweis:

Jede Kante zwischen zwei Knoten wird verdoppelt,so dass eine zweite gleiche Kante zwischen den Knoten existiert.Dann muß der Knotengrad jedes Knoten gerade sein.Also existiert nach Satz V.1 eine geschlossene Eulerlinie,in dem jede Kante genau einmal durchlaufen wird.Faßt man die neu eingefügten Kanten mit den ursprünglichen wieder zusammen,wird jede dieser Kanten genau zweimal durchlaufen.

Bemerkung: Damit ergibt sich für einen Hamiltokreis (s.u.) eine obere Abschätzung des maximalen Weges von zweimal der Summe aller Kantenwerte.

Wie jedoch läßt sich eine Eulerlinie konstruieren?

C Aufgabe V.2: Teil 2 (2.Einstiegsproblem)

Beobachte den Ablauf des Algorithmus Eulerlinie (geschlossen) an Hand des Graphen I(G024.gra)mit dem Programm Knotengraph mittels des Demo-oder Einzelschrittmodus.Beschreibe die Reihenfolge der Auswahl der rot markierten Kanten,und versuche die Art des Algorithmus allgemein zu erkenn-nen.(Startknoten A)

Beobachtung und problemorientierte Erarbeitung: Vom Knoten A aus werden zunächst die Kanten in der Knotenreihenfolge ACA und dann weiter ADA durchlaufen.Obwohl der Startknoten hier schon zum zweiten Mal wieder erreicht wurde,bemerkt der Algorithmus an Hand der Zahl der bisher durchlaufenden Kanten,dass das Verfahren noch nicht beendet ist.Also muß es eine (Stufen-)Variable geben,die die Anzahl der schon im Pfad vorhandenen Kanten mitzählt.Die weitere Auswahl ist ABCA.Hier tritt zusätzlich die Situation ein,dass keine weiteren von A ausgehenden Kanten,die sich nicht schon in der Linie befinden, mehr vorhanden sind.Um nicht schon in der Linie befindliche Kanten doppelt aufzunehmen,muß das Programm solche Kanten also als besucht markieren.Die Markerung der Kante von C nach A wird wieder gelöscht,d.h. die Linie verkürzt sich um diese Kante (Rückwärtsgehen) und als nächste Kante wird von C aus die Kante nach B gewählt (Vorwärtsgehen).Die weitere Linie ist dann: BDCA (Lösung),d.h. die Kante CA wird hier erneut gewählt.Also muß ihre Besucht-Markierung beim Rückwärtsgehen gelöscht worden sein.Demnach besteht der Algorithmus jeweils aus einem Vor-und Rückwärtsgehen,wobei das letztere immer dann nötig wird,wenn sich der Algorithmus in einer „Sackgasse" befindet.Beim Rückwärtsgehen werden die Besucht-Markierungen wieder gelöscht,und auch die Stufenvariabble muß um Eins vermindert vermindert worden sind.Die Lösung ist gefunden,wenn der Startknoten wieder - mit einer Stufenvariablen gleich der Anzahl der im Graph vorhanden Kanten - erreicht wird.Wie der weitere Ablauf nach Finden der Lösung zeigt,werden bei jedem Knoten noch anschließend alle Möglichkeiten der Kantenauswahl realisiert.
Das Verfahren heißt Backtrackingverfahren mit der Anzahl der Kanten als Stufenparameter,das jetzt folgendermaßen präzisiert werden kann:

Algorithmus Eulerlinie

Verbale Beschreibung:

I)Setze Stufe (Anzahl der gewählten besuchten Kanten) gleich 0.Wähle im Falle geschlossener Linie den aktuellen Knoten als Start-und Zielknoten und im Falle offener Linie die Knoten mit ungeradem Grad (für ungerichteten

Graph) bzw. Grad ungleich 0 (für gerichteten Graph) als Start-und Zielknoten.Wähle eine leere Kantenliste zur Aufnahme der Eulerlinie.

Rufe mit diesen Parameter die Backtrackingmethode auf.

Backtrackingmethode:

II)Wähle vom aktuellen Knoten die in der Kantenreihenfolge nächste ausgehenden Kante aus.Wenn alle ausgehenden Kanten des aktuellen Knotens untersucht worden sind,verlasse diese Rekursionsebene der Backtrackingmethode.Wenn es die Anfangsebene ist,beende den Algorithmus.

III)Prüfe,ob der Wert für Stufe kleiner oder gleich der Anzahl der Kanten des Graphen ist und die Kante noch nicht als besucht markiert ist.Wenn ja,setze bei IV) den Algorithmus fort.Wenn nein,setze bei VIII) den Algorithmus fort.

IV)Markiere die Kante als besucht.Füge die Kante in die Kantenliste ein.Erhöhe die Stufenzahl (Zahl der ausgewählten Kanten) um 1.

V)Prüfe, ob der Zielknoten dieser Kante gleich dem Zielknoten ist und gleichzeitig der Wert für Stufe gleich der Anzahl der Kanten des Graphen vergrößert um 1 ist.(Dann sind alle Kanten besucht)

VI)Wenn ja,füge die Kantenliste in die Pfadliste des Zielknotens ein.(Oder gebe die Lösung aus.)Dann setze den Algorithmus bei VIII) fort.Wenn nur eine Lösung gesucht wird,beende alle Rekursionsebenen der Backtrackingmethode.Wenn nein setze bei bei VII) den Algorithmus fort.

VII)Rufe die nächste Rekursionsebene der Bachtrackingprocedure mit dem aktuellen Wert für Stufe (um 1 erhöht),dem Zielknoten der aktuellen Kante als aktuellem Knoten und der aktuellen Kantenliste (sowie Start-und Zielknoten) als Parameter auf.
(Dann Backtrack!)

VIII)Vermindere Stufe (Anzahl der ausgewählten Kanten) um 1 und lösche die letzte Kante aus der Kantenliste.Lösche die Besucht-Markierung dieser Kante.Setze den Algorithmus bei II) fort.

Nach Ende des Algorithmus sind in der Pfadliste des Zielknotens die möglichen Eulerlinien gespeichert.Wenn diese leer ist,gibt es keine Lösung.

(Bei einer manuellen Lösung sollten die Pfade wie oben in der Reihenfolge der Knoten also z.B. ACADABCA rückwärts CBDCA beschrieben und zusätzlich die jeweilige Stufenzahl notiert werden.)

Quellcode:

(Unit UMath1)

Die Methode Euler ist die Backtracking-Methode.Stufe ist die Stufenzahl der Backtrackingmethode und bedeutet die Anzahl der schon ausgewählten Kanten. Der aktuell besuchte Knoten ist Kno und Zielknoten ist der Knoten,in dem die Eulerlinie endet.(Im Falle einer geschl. Eulerlinie ist dies auch der Startknoten.)Die Kantenliste ist der Parameter Kliste,Flaeche ist die Objektzeichenfläche,EineLoesung gibt an,ob nur eine Lösung gesucht werden soll,Gefunden wird true,wenn eine Lösung gefunden ist und der Label Ausgabe dient zur Anzeige der Knotenfolge sowie der Kantensumme der Eulerlinie.
Die Methode Euler wird von der Methode Eulerlinie aufgerufen.

Deklaration des Datentyps:

```
TEulergraph   = class(TInhaltsgraph)
    constructor Create;
    procedure Euler(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;var Kliste:TKantenliste;
      Flaeche:TCanvas;EineLoesung:Boolean;var Gefunden:Boolean);
    procedure Eulerfix(Kant:TInhaltskante;Kno,Zielknoten:TInhaltsknoten;
```

```
    var Kliste:TKantenliste;Flaeche:TCanvas;Ausgabe:TLabel);
    procedure Eulerlinie(Flaeche:TCanvas;Ausgabe:TLabel;var SListe:TStringlist;
    Anfangsknoten,Endknoten:TInhaltsknoten);
  end;
```

Methoden:

```
constructor TEulergraph.Create;
begin
  inherited Create;
end;

procedure TEulergraph.Euler(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;
var Kliste:TKantenliste;Flaeche:TCanvas;EineLoesung:Boolean;var Gefunden:Boolean);
var Index:Integer;
    Zkno:TInhaltsknoten;
    Ka:TInhaltskante;
begin
  if EineLoesung and Gefunden then exit;                                          1)
  if not Kno.AusgehendeKantenliste.Leer then
    for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do                         2)
    begin
      Ka:=TInhaltskante(Kno.AusgehendeKantenliste.Kante(Index));
      Zkno:=TInhaltsknoten(Kno.AusgehendeKantenliste.Kante(Index).Zielknoten(Kno));      3)
      if ((not Ka.Besucht) and (Stufe<=Self.AnzahlKantenmitSchlingen)) then        4)
      begin
        Ka.Pfadrichtung:=Zkno;                                                      5)
        Ka.Besucht:=true;                                                           6)
        Kliste.AmEndeanfuegen(Ka);                                                  7)
        Stufe:=Stufe+1;                                                             8)
        if (Zkno=Zielknoten)and (Stufe=Self.AnzahlKantenmitSchlingen+1)  then       9)
        begin
          Zielknoten.Pfadliste.AmEndeanfuegen(Kliste.Kopie.Graph);                 10)
          Gefunden:=true;                                                          11)
          if EineLoesung then exit;                                                12)
        end
        else
          Euler(Stufe,Zkno,Zielknoten,Kliste,Flaeche,EineLoesung,Gefunden);        13)
        Stufe:=Stufe-1;                                                            14)
        Ka.Besucht:=false;                                                         15)
        if not Kliste.Leer then                                                    16)
          Kliste.AmEndeloeschen(TObject(Ka));;
      end;
    end;
end;

procedure TEulergraph.Eulerlinie(Flaeche:TCanvas;Ausgabe:TLabel;var SListe:TStringlist;
  Anfangsknoten,Endknoten:TInhaltsknoten);
var MomentaneKantenliste:TKantenliste;
    Zaehl:Integer;
    T:TInhaltsgraph;
    EineLoesung:Boolean;
    Gefunden:Boolean;
begin
  EineLoesung:=false;
  Gefunden:=false;
  if MessageDlg('Nur eine Loesung?',mtConfirmation,[mbYes,mbNo],0)=mryes then     17)
  EineLoesung:=true;
  if Leer then exit;
  SListe:=TStringlist.Create;
  MomentaneKantenliste:=TKantenliste.Create;                                        18)
  LoescheKantenbesucht;
  Pfadlistenloeschen;                                                               19)
  Ausgabe.Caption:='Berechnung läuft';
  Ausgabe.Refresh;
  if EineLoesung then
  begin
   if MessageDlg('Schneller Euleralgorithmus?', mtConfirmation,
    [mbYes, mbNo], 0) = mrYes
    then
    begin
      Eulerfix(nil,Anfangsknoten,Endknoten,MomentaneKantenliste,Flaeche,Ausgabe);
      Endknoten.Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
      Gefunden:=true;
    end
    else
      Euler(1,Anfangsknoten,Endknoten,MomentaneKantenliste,Flaeche,EineLoesung,Gefunden, 20)
      Ausgabe)
  end
 else
   Euler(1,Anfangsknoten,Endknoten,MomentaneKantenliste,Flaeche,EineLoesung,Gefunden,Ausgabe);
 Endknoten.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
  if not Endknoten.Pfadliste.leer then
```

```
  begin
    T:=TInhaltsgraph(Endknoten.Pfadliste.Pfad(0));                          21)
    if not T.Leer then
    begin
      Ausgabe.Caption:='Eulerlinie: '+                                      22)
      T.InhaltallerKnoten(ErzeugeKnotenstring)+' Summe: '+
        RundeZahltostring(T.Kantensumme(BeWertung),Kantengenauigkeit)
        +' Produkt: '+
        RundeZahltostring(T.Kantenprodukt(Bewertung),Kantengenauigkeit);
      Ausgabe.Refresh;
      T.FaerbeGraph(clred,psdot);                                           23)

      T.ZeichneGraph(Flaeche);                                              24)
      ShowMessage(Ausgabe.Caption);
    end
  end;
  Ausgabe.Caption:='';
  Ausgabe.Refresh;
end;
```

Erläuterung des Algorithmus:

Wenn bei 1) nur eine Lösung gesucht wird, und diese schon gefunden wurde, wird die Rekursionsebene der Backtrackingmethode verlassen.Bei 2) werden dann alle vom aktuellen Knoten Kno ausgehenden Kanten nacheinander ausgewählt,und bei 3) wird der Zielknoten der Kante bestimmt.Wenn bei 4) die als aktuell ausgewählte Kante noch nicht als besucht markiert wurde und Stufe (Anzahl der schon ausgewählten Kanten) noch kleiner oder gleich der Anzahl der Kanten ist,werden folgende Programmanweisungen ausgeführt (ansonsten,wenn alle Kanten untersucht wurden,wird zur vorigen Rekursionsebene zurückgekehrt,indem diese Rekursionsebene beendet wird):Bei 5) wird die Pfadrichtung der Kante gesetzt,die Kante wird bei 6) als besucht markiert,bei 7) wird die Kante in die Kantenliste,die den bisher gefundenen Teil der Eulerlinie darstellt,eingefügt und Stufe (Anzahl der schon ausgewählten Kanten) wird bei 8) um 1 erhöht.Bei 9) wird geprüft,ob der Zielknoten der Kante schon identisch ist mit dem vorgegebenen Zielknoten des Graphen und ob gleichzeitig Stufe gleich der Anzahl der Kanten des Graphen vergrößert um 1 ist.(Dies bedeutet,da Stufe zuvor um 1 erhöht wurde,dass alle Kanten des Graphen ausgewählt wurden.)Wenn ja,wird die Kantenliste als Pfad in der Pfadliste des Zielknoten bei 10) gespeichert,gefunden auf true gesetzt (11) und,wenn nur eine Lösung gesucht wird,die weitere Ausführung des Verfahrens bei 12) abgebrochen.Wenn nein wird die nächste Rekursionsebene der Backtrackingmethode bei 13) mit den neuen Werten für die Parameter Stufe und aktueller Knoten aufgerufen.Bei Rückkehr aus der nächsten Rekursionsebene wird der Ablauf bei 14) fortgesetzt.Dort wird dann zunächst wieder Stufe um 1 vermindert,die Besucht-Markierung der Kante wieder gelöscht (15) und die Kante aus der Kantenliste gelöscht (16).Die Methode Eulerlinie ruft die Backtrackingmethode Euler auf.Bei 17) wird eingegegeben,ob nur eine Lösung bestimmt werden soll (da alle Eulerlinie gleich sind,genügt es im allgemeinen,diese Option zu wählen.)Bei 18) wird die Kantenliste und die Ausgabeliste initiiert.Bei 19) werden alle Pfadlisten gelöscht und bei 20) wird die Backtrackingmethode Euler aufgerufen.Da sich die Eulerlinie dann in der Pfadliste des Zielknotens befinden,wird bei 21) ein Graph aus dem 0. Pfad dieser Pfadliste gebildet,dessen Knotenfolge sowie Kantensumme bei 22) erzeugt und im Label Ausgabe gespeichert wird.Bei 23) und 24) wird der Graph rot-markiert gezeichnet (d.h. alle Kanten als Eulerlinie).Statt des Speicherns der Eulerlinien bei Markierung 10) in der Pfadliste kann vereinfacht auch eine direkte Ausgabe der Linien als Graphen erfolgen,wenn die Anweisung ersetzt wird durch:

```
TEulergraph(Kliste.Graph).Graphzeichnen(Flaeche,Ausgabe,Bewertung,SListe,
  TEulergraph(self.Graph).Demo,TEulergraph(self.Graph).Pausenzeit,
  TEulergraph(self.Graph).Kantengenauigkeit);
```

Dadurch können die die Anweisungen ab Markierung 21) in der Methode Eulerlinie entfallen.Die Parameter in der Methode Euler müssen dann natürlich entsprechend ergänzt werden.(vgl. dazu auch die Methode ErzeugeTiefeBaumpfadeeinfach im Kapitel C II)
Bemerkung: Obwohl die Eulerlinien keine Pfade (Definition C I.2) sind,werden oben trotzdem die Pfadlisten aus didaktischen Gründen wegen der 1:1-Übertragung des Verfahrens auf den Hamilton-Algorithmus (s.u.) benutzt.

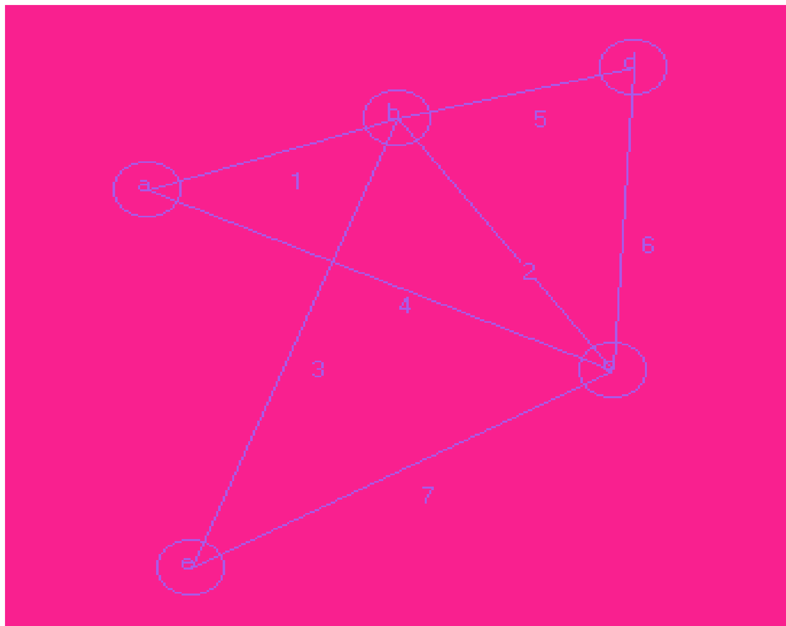
C Aufgabe V.4:

a)Bestimme in dem folgenden Graph G026.gra nach dem obigen Backtracking-Algorithmus einen Eulerkreis von Knoten a aus.

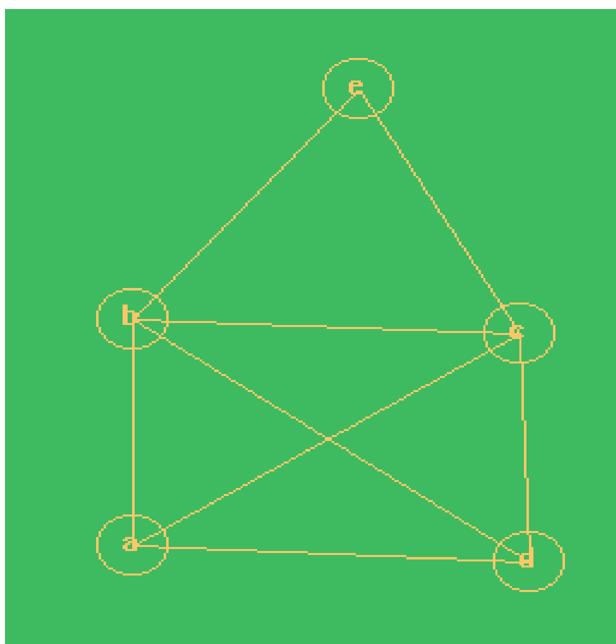
b)Zwischen a und c wird die Kante gelöscht.Bestimme nun nach dem Back-trackingalgorithmus eine offene Eulerlinie zwischen a und c.

c)Bestimme nach dem obigen Bachtrackingalgorithmus in dem Graph G027.gra „Haus vom Nikolaus“ eine offene Eulerlinie.Das bedeutet , dass der Graph ohne den Stift abzusetzen in einem Stück gezeichnet werden kann.

(Lösung per Hand,mit dem Programm Knotengraph mit Hilfe des Demo-Modus und dem Algorithmus des Menüs Anwendungen/Eulerlinie oder mit Hilfe des eigenen Quellcodes mittels der Entwicklungsumgebung EWK je nach Konzeption.)



G026.gra



G027.gra

Lösungen:

a) a b d c b e c a
b) a b d c b e c
c) a b c a d b e c d

Das Thema Euleralgorithmen wird auf Seite 242 wieder aufgenommen.Zunächst soll jetzt zunächst das Hamiltonproblem erläutert werden:

Ein Eulerlinie ist eine Linie,der alle Kanten genau einmal enthält.Eine andere Möglichkeit einen Kreis in einem Graphen zu erzeugen besteht darin,dass die Linie statt alle Kanten alle Knoten genau einmal enthält.

C Definition V.2:

Ein Kreis in einem Graphen,der alle Knoten genau einmal enthält,heißt Hamiltonkreis.

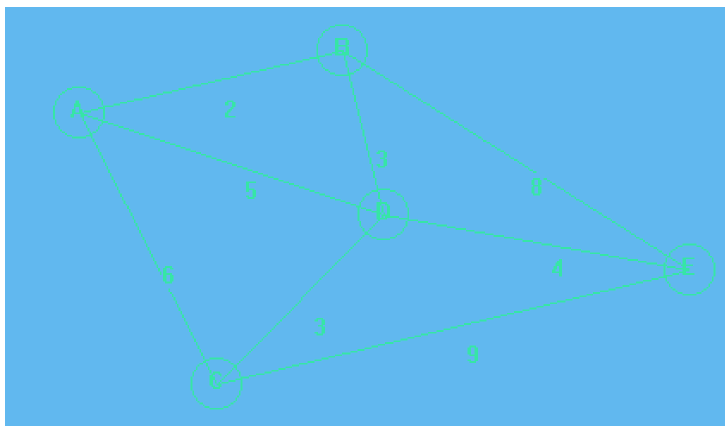
Besitzt ein Graph eine positive Kantenbewertung so heißt ein Hamiltonkreis mit kleinster Kantensumme unter allen möglichen Hamiltonkreisen Lösung des Traveling-Salesman-Problems.

C Aufgabe V.6 (Einstiegsproblem):

Gegeben ist das durch den folgenden Graphen dargestellte Straßennetz zwischen Städten.Die Knoten bedeuten Städte und die Kanten sind die Verbindungsstraßen zwischen ihnen.Die Bewertung der Kanten bedeutet die Straßenlänge zwischen zwei Städten.
Ein Handelsreisender soll nun alle Städte so auf einer Rundreise besuchen,dass keine Stadt zweimal besucht wird, und dabei außerdem ein Kreisweg von minimaler Länge gewählt wird.

Die Lösung des Problems bedeutet:Suche den Hamiltonkreis mit der kleinsten Kantensumme im vorliegenden Graphen,d.h. löse das Traveling-Salesman-Problem.

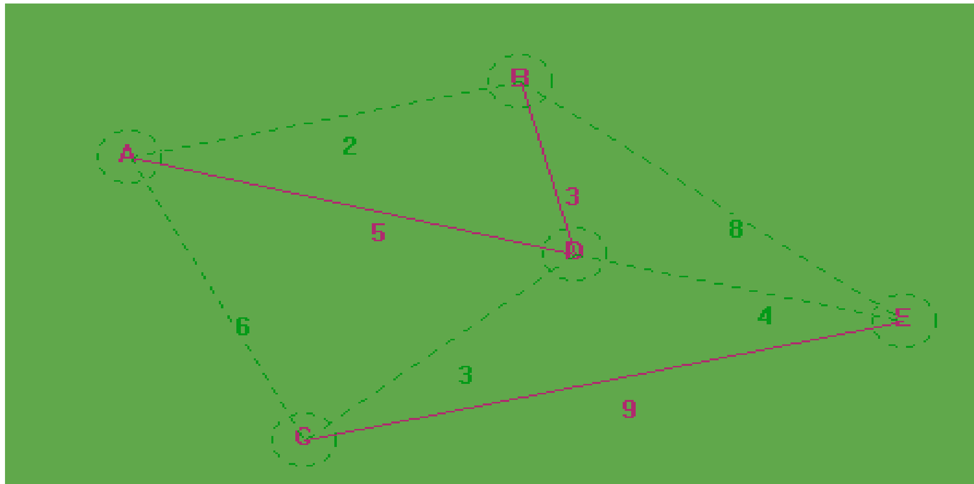
Suche per Hand alle Hamiltonkreise im folgenden Graphen,berechne deren Kantensumme und bestimme dann die Lösung des Traveling-Salesman-Problems.
Kontrolliere das Ergebnis mit Hilfe von Programm Knotengraph mit Hilfe des Menüs Anwendungen/Hamiltonkreise im Demomodus.



G028.gra

Lösung:

Die Lösung des Traveling-Salesman-Problems ABEDCA mit der Kantensumme 23 ist rot-markiert eingezeichnet:



Hamiltonkreise:

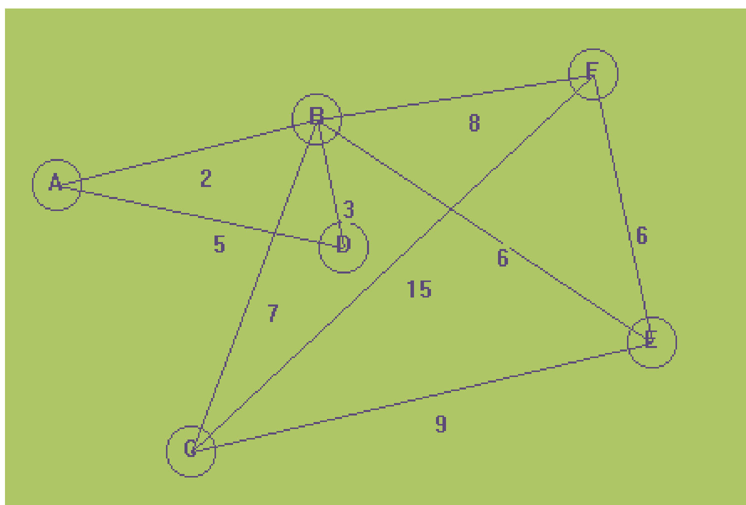
A B D E C A Summe: 24 Produkt: 1296
A B E C D A Summe: 27 Produkt: 2160
A B E D C A Summe: 23 Produkt: 1152
A C D E B A Summe: 23 Produkt: 1152
A C E B D A Summe: 31 Produkt: 6480
A C E D B A Summe: 24 Produkt: 1296
A D B E C A Summe: 31 Produkt: 6480
A D C E B A Summe: 27 Produkt: 2160

Traveling Salesmann Lösung: A B E D C A Summe: 23 Produkt: 1152

Jeder Pfad kommt beim ungerichteten Graphen natürlich zweimal,nämlich jeweils noch zusätzlich in umgekehrter Knotenfolge vor.

C Aufgabe V.7: (2. Einstiegsproblem)

a)Ermittele die Hamiltonkreise des folgenden Graphen (per Hand):



G029.gra

b)Ermittle die Hamiltonkreises des Graphen von Aufgabe V.6,wenn dort die Kanten zwischen A und C sowie zwischen B und D gelöscht sind.

Überprüfe jeweils das Ergebnis mit dem Programm Knotengraph mittels des Algorithmus des Menüs Anwendungen/Hamiltonkreise.

Lösungen:

Es gibt in beiden Fällen keine Hamiltonkreise.

Beim zuletzt untersuchten Graph ist dies sofort klar,da der Knoten D jeweils mindestens zweimal besucht werden muß.

Es ergeben sich damit die Fragen,wie zu entscheiden ist,ob ein Graph Hamiltonkreise besitzt,und wenn ja,wie sich diese Kreise bestimmen lassen.

Leider ist bis heute dafür kein so einfach anzuwendendes Kriterium wie bei den Eulerlinien bekannt.

Die Frage muß stattdessen durch einen Algorithmus,der systematisch alle möglichen Hamiltonkreise sucht,entschieden werden.

Zur problemorientierten Erarbeitung des Hamilton-Algorithmus siehe dazu die problemorientierte Einstiegsaufgabe im Kapitel BIII,Abschnitt 5 (CAK), die sich auf den Graph G028.gra und Aufgabe C V 6 bezieht.

Algorithmus Hamiltonkreise

Verbale Beschreibung:

Der Backtracking-Algorithmus,der beim Suchen der Eulerlinien nützlich war,muß dazu nur geringfügig abgeändert werden:

Als Stufe des Backtrackingalgorithmus dient hier statt der Anzahl der schon gewählten Kanten jetzt die Anzahl der schon gewählten Knoten des Graphen.

Da ein Kreis nämlich der Hamiltonkreis gesucht wird,werden wie beim Algorithmus Eulerlinie die schon ausgewählten Kanten in einer Kantenliste gespeichert.

Deshalb ergeben sich nur geringfügige Unterschiede zwischen den beiden Algorithmen,die hauptsächlich die Abbruchbedingungen betreffen:

I)Setze Stufe (Anzahl der gewählten besuchten Knoten) gleich 1.Wähle einen beliebigen Knoten als Start-und Zielknoten.Wähle eine leere Kantenliste zur Aufnahme des Hamiltokreises.

Rufe mit diesen Parameter die Backtrackingmethode auf.

Backtrackingmethode:

II)Wähle vom aktuellen Knoten die in der Kantenreihenfolge nächste ausgehenden Kante aus.

Wenn alle ausgehenden Kanten des aktuellen Knotens untersucht worden sind,verlasse diese Rekursionsebene der Backtrackingmethode.Wenn es die Anfangsebene ist,beende den Algorithmus.

III)Prüfe,ob der Wert für Stufe kleiner oder gleich der Anzahl der Knoten (bzw. ob Stufe kleiner als Anzahl+1 ist) des Graphen ist und der Zielknoten der aktuellen Kante noch nicht als besucht markiert ist.Wenn ja,setze bei IV) den Algorithmus fort.Wenn nein,setze bei VIII) den Algorithmus fort.

IV)Markiere den Zielknoten als besucht.Füge die Kante in die Kantenliste ein.Erhöhe die Stufenzahl (Zahl der ausgewählten Kanten) um 1.

V)Prüfe,ob der Zielknoten dieser Kante gleich dem Zielknoten ist,und gleichzeitig der Wert für Stufe gleich der Anzahl der Knoten des Graphen vergrößert um 1 ist und gleichzeitig Stufe größer als 2 ist.(Dann sind alle Knoten besucht und der Hamiltonkreis besteht aus mehr als 2 Knoten.Mit dem

Letzteren wird ausgeschlossen,dass ein Hamiltonkreis nur aus dem Durchlaufen zweier paralleler Kanten besteht.)

VI)Wenn ja,füge die Kantenliste in die Pfadliste des Zielknotens ein.(Oder gebe die Lösung aus.)Dann setze den Algorithmus bei VIII) fort.Wenn nein setze bei bei VII) den Algorithmus fort.

VII)Rufe die nächste Rekursionsebene der Backtrackingprocedure mit dem aktuellen Wert für Stufe (um 1 erhöht),dem Zielknoten der aktuellen Kante als aktuellem Knoten und der aktuellen Kantenliste (sowie dem Zielknoten) als Parameter auf.
(Dann backtrack!)

VIII)Vermindere Stufe (Anzahl der ausgewählten Knoten) um 1,und lösche die letzte Kante aus der Kantenliste.Lösche die Besucht-Markierung des Zielknotens dieser Kante.Setze den Algorithmus bei II) fort.

Nach Ende des Algorithmus sind in der Pfadliste des Zielknotens die möglichen Hamiltonkreise gespeichert.Wenn diese leer ist,gibt es keine Lösung.

(Bei einer manuellen Lösung sollten dieLinien wie oben bei den Eulerlinien in der Reihenfolge der Knoten beschrieben und zusätzlich die jeweilige Stufenzahl notiert werden.)

Quellcode:

(Unit UMath1)

Die Methode Hamilton ist die Backtracking-Methode.Stufe ist die Stufenzahl der Backtrackingmethode und bedeutet die Anzahl der schon ausgewählten Knoten.Der aktuell besuchte Knoten ist Kno und Zielknoten ist gleich dem zuerst ausgewählten aktuellen Startknoten.Die Kantenliste ist der Parameter Kliste,Flaeche ist die Objektzeichenfläche.
Die Methode Hamilton wird von der Methode Hamiltonkreise aufgerufen.

Deklaration des Datentyps:

```
THamiltongraph = class(TInhaltsgraph)
    constructor Create;
    procedure Hamilton(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;var Kliste:TKantenliste;
      Flaeche:TCanvas);
    procedure Hamiltonkreise(Flaeche:TCanvas;Ausgabe:TLabel;var SListe:TStringlist);
  end;
```

Methoden:

```
constructor THamiltongraph.Create;
begin
  inherited create;
end;

procedure THamiltongraph.Hamilton(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;
Var Kliste:TKantenliste;
  Flaeche:TCanvas);
var Index:Integer;
    Zkno:TInhaltsknoten;
    Ka:TInhaltskante;
begin
  if not Kno.AusgehendeKantenliste.leer then                                          1)
    for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do                             2)
    begin
      Ka:=TInhaltskante(Kno.AusgehendeKantenliste.Kante(Index));
      Zkno:=TInhaltsknoten(Kno.AusgehendeKantenliste.Kante(Index).Zielknoten(Kno));   3)
      if ((Not Zkno.Besucht) and (Stufe<Self.AnzahlKnoten+1)) or
        ((Zkno=Zielknoten)and (Stufe=Self.AnzahlKnoten)and(Stufe>2) )                 4)
      then
      begin
        Ka.Pfadrichtung:=Zkno;                                                        5)
        Zkno.Besucht:=true;                                                           6)
```

```
        Kliste.AmEndeanfuegen(Ka);                                         7)
        Stufe:=Stufe+1;                                                     8)
        if (Zkno=Zielknoten)and (Stufe=self.AnzahlKnoten+1)                 9)
        then
          Zielknoten.Pfadliste.AmEndeanfuegen(Kliste.kopie.Graph)          10)
        else
          Hamilton(Stufe,Zkno,Zielknoten,Kliste,Flaeche);                  11)
        Stufe:=Stufe-1;                                                    12)
        if Zkno<>Zielknoten then Zkno.Besucht:=false;                      13)
        if not Kliste.Leer then
          Kliste.AmEndeloeschen(TObject(Ka));                              14)
      end
    end;
end;

procedure THamiltongraph.Hamiltonkreise(Flaeche:TCanvas;Ausgabe:TLabel;
var SListe:TStringlist);
var Kno:TInhaltsknoten;
    MomentaneKantenliste:TKantenliste;
    zaehl:Integer;
    T:TInhaltsgraph;
begin
  if Leer then exit;
  SListe:=TStringlist.Create;                                              15)
  MomentaneKantenliste:=TKantenliste.Create;
  LoescheKnotenbesucht;                                                    16)
  Pfadlistenloeschen;
  Kno:=LetzterMausklickknoten;                                             17)
  Kno.Besucht:=true;                                                       18)
  Ausgabe.Caption:='Berechnung läuft';
  Hamilton(1,Kno,Kno,MomentaneKantenliste,Flaeche);                        19)
  Kno.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);                  20)
  FaerbeGraph(clblack,pssolid);
  ZeichneGraph(Flaeche);
  T:=TInhaltsgraph(Kno.MinimalerPfad(Bewertung));                          21)
  if not T.Leer then
  begin
    Ausgabe.Caption:='Traveling Salesmann Lösung: '+                       22)
    T.InhaltallerKnoten(ErzeugeKnotenstring)+' Summe: '+
      RundeZahltostring(t.Kantensumme(BeWertung),Kantengenauigkeit)
      +' Produkt: '+
      RundeZahltostring(T.Kantenprodukt(Bewertung),Kantengenauigkeit);
    SListe.Add(Ausgabe.Caption);
    Ausgabe.Refresh;
    T.FaerbeGraph(clred,psdot);
    T.ZeichneGraph(Flaeche);
    ShowMessage(Ausgabe.Caption);
    T.FaerbeGraph(clred,psdot);                                            23)
    T.ZeichneGraph(Flaeche);                                               24)
    MomentaneKantenliste.Free;
    MomentaneKantenliste:=nil;
    if Abbruch then exit
  end;
  Ausgabe.Caption:='';
  Ausgabe.Refresh;
end;
```

Erläuterung des Algorithmus:

Wenn es bei 1) vom aktuellen Knoten Kno ausgehende Kanten gibt, werden bei 2) dann alle vom aktuellen Knoten ausgehenden Kanten nacheinander ausgewählt,und bei 3) wird der Zielknoten der Kante bestimmt.Wenn bei 4) der Zielknoten der als aktuell ausgewählte Kante noch nicht als besucht markiert wurde und Stufe (Anzahl der schon ausgewählten Knoten) noch kleiner als die Anzahl der Knoten des Graphen ist,oder Stufe gleich der Anzahl der Knoten des Graphen ist,der Zielknoten gleich dem aktuellen Knoten ist,und der Hamiltonkreis aus mehr als zwei Knoten besteht,werden folgende Programmanweisungen ausgeführt (ansonsten,wenn alle Knoten untersucht wurden,wird zur vorigen Rekursionsebene zurückgekehrt,indem diese Rekursionsebene beendet wird):

Bei 5) wird die Pfadrichtung der Kante gesetzt,der Zielknoten der Kante wird bei 6) als besucht markiert,bei 7) wird die Kante in die Kantenliste,die den bisher gefundenen Teil des Hamiltonkreises darstellt,eingefügt,und Stufe (Anzahl der schon ausgewählten Kanten) wird bei 8) um 1 erhöht.Bei 9) wird geprüft,ob der Zielknoten der Kante schon identisch ist mit dem vorgegebenen Zielknoten des Graphen und ob gleichzeitig Stufe gleich der Anzahl der Kanten des Graphen vergrößert um 1 ist.(Dies

bedeutet,da Stufe zuvor um 1 erhöht wurde,dass alle Kanten des Graphen ausgewählt wurden.)

Wenn ja,wird die Kantenliste als Pfad in der Pfadliste des Zielknoten bei 10) gespeichert.

Wenn nein,wird die nächste Rekursionsebene der Backtrackingmethode bei 11) mit den neuen Werten für die Parameter Stufe und aktueller Knoten aufgerufen.

Bei Rückkehr aus der nächsten Rekursionsebene wird der Ablauf bei 12) fortgesetzt.Dort wird dann zunächst wieder Stufe um 1 vermindert,die Besucht-Markierung des Zielknotens der Kante wird wieder gelöscht (13),wenn der Zielknoten der Kante nicht gleich dem Zielknoten (gleich Startknoten) des Graphen ist,und die Kante wird aus der Kantenliste gelöscht (14).

Die Methode Hamiltonkreise ruft die Backtrackingmethode Hamilton auf.Bei 15) wird die Kantenliste und die Ausgabeliste initiiert.Bei 16) werden alle Pfadlisten gelöscht, und die Besucht-Markierungen der Knoten gelöscht,bei 17) wird als Startknoten für den Hamiltonkreis der letzte mit der linken Maustaste ausgewählte Knoten für Kno,falls existiert (oder andernfalls der erste in den Graph eingefügte Knoten) ausgewählt,der Knoten wird bei 19) als besucht markiert,und bei 20) wird die Backtrackingmethode Hamilton aufgerufen.

Da sich danach alle Hamiltonkreise in der Pfadliste des Zielknotens befinden,werden bei 20) alle Pfade der Pfadliste rot markiert gezeichnet, und bei 21) wird die Lösung des Traveling-Salesman-Problems als der Kantenbewertung nach minimalste Pfad der Pfadliste des Zielknotens als Graph gespeichert,dessen Knotenfolge sowie Kantensumme bei 22) gebildet, und im Label Ausgabe gespeichert wird.Bei 23) und 24) wird der Graph rot-markiert gezeichnet (d.h. alle Kanten der Lösung des Traveling-Salesman-Problems).

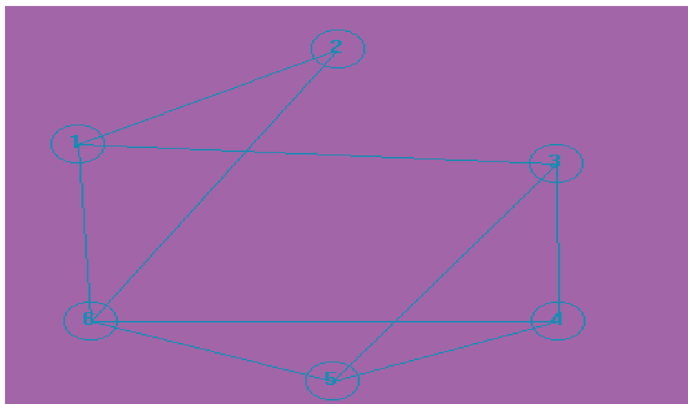
Bemerkung:Die Hamiltonkreise werden auch bei einem gerichteten Graph durch den obigen Algorithmus bestimmt.

C Aufgabe V.8:

Bestimme in den folgenden Graphen 1) bis 3) nach dem obigen Backtrackingverfahren einen sich ergebenden Hamiltonkreis.Beim Graphen 4 bestimme die Traveling-Salesman-Lösung.
Ermittle die Lösungen manuell (Graph 1 ,2 und 4),mit Hilfe des Programms Knotengraph und dem Algorithmus des Menüs Anwendungen/Hamiltonkreise im Demo-Modus oder mit Hilfe des selbst erstellten Quellcodes mittels der Entwcklungsumgebung,je nach Konzeption.)

Graph 1)

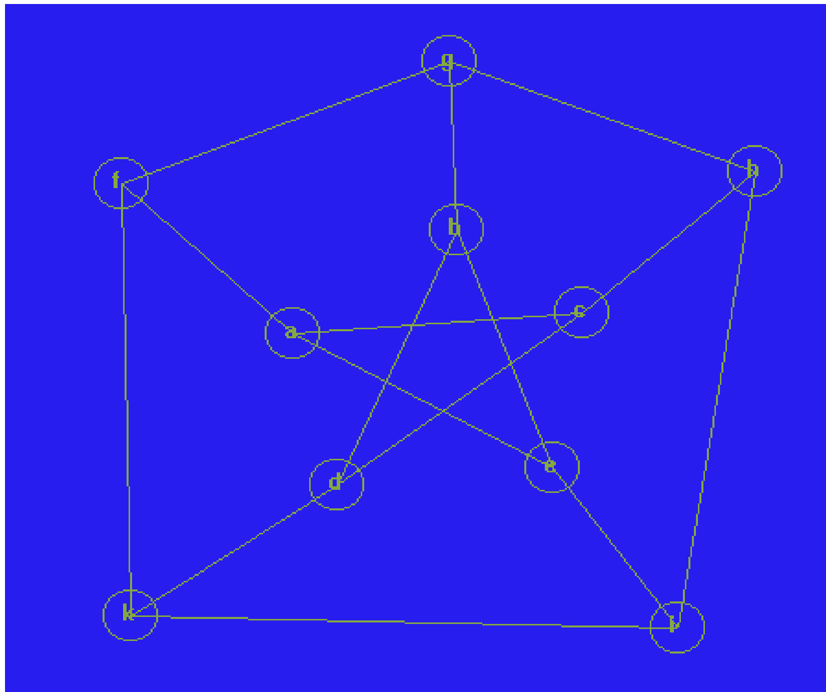
Startknoten 1:



G030.gra

Graph 2:

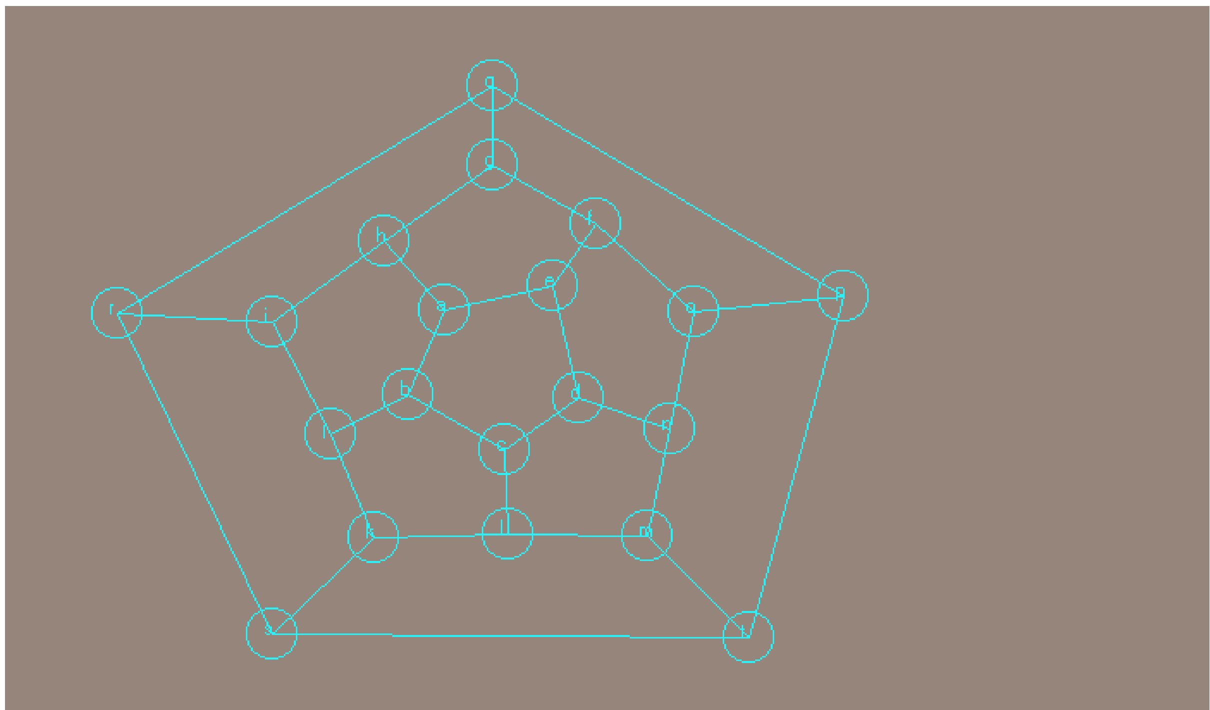
Startknoten a:



G031.gra

Graph 3:

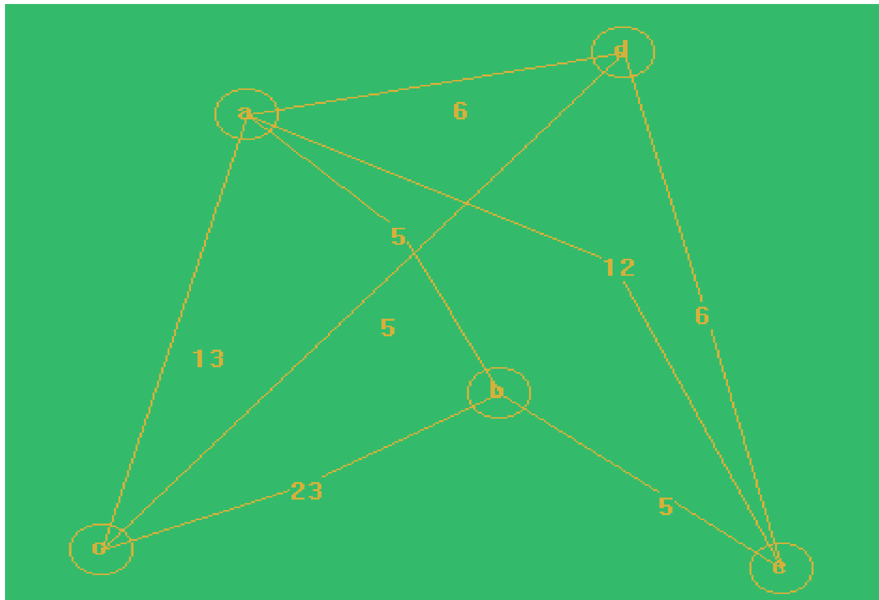
Startknoten a:



G032.gra

Graph 4:

Startknoten a:



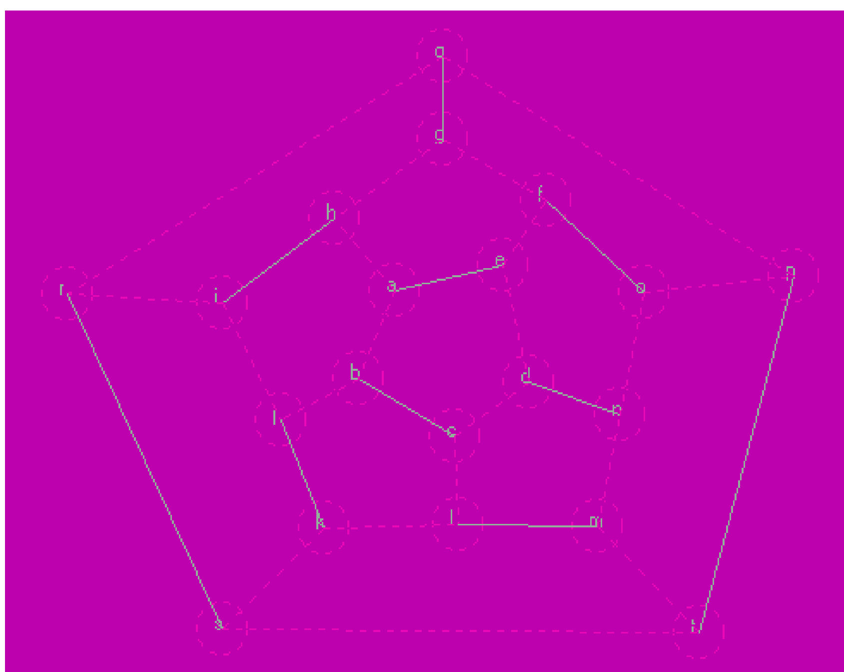
G033.gra

Lösungen:

1) 1 2 6 4 5 3 1
   oder
   1 2 6 5 4 3 1
   oder
   1 3 4 5 6 2 1
   oder
   1 3 5 4 6 2 1

2) Es gibt keinen Hamiltonkreis.

3) z.B: abcdenopqrstmlkjihgfa



Pfade:

a b c d e f g q r s t p o n m l k j i h a Summe: 20.000 Produkt: 1.000
a b c d e f o n m l k j i r s t p q g h a Summe: 20.000 Produkt: 1.000
a b c d n m l k j i h g q r s t p o f e a Summe: 20.000 Produkt: 1.000
a b c d n o p q r s t m l k j i h g f e a Summe: 20.000 Produkt: 1.000

usw. (insgesamt 60 Lösungen)

Traveling Salesmann Lösung: a h g f e d c l k s t m n o p q r i j b a Summe: 20.000 Produkt: 1.000

Dieser Graph wurde von Hamilton selbst untersucht.Er ist isomorph zum Kantennetz des Dodekaeders.

4)edcabc Kantensumme: 30

An dieser Stelle ist es nun im Unterricht Zeit zu zeigen,was „mathematische Intelligenz",die in einen Algorithmus „hineingesteckt" wird,bewirken kann:

Das bisher für das Hamilton-und Eulerproblem benutzte Backtrackingverfahren nutzt kaum die besonderen Eigenschaften eines Problems aus und ist im wesentlichen ein Durchsuchen aller vorhandenen Möglichkeiten,aus denen die geeigneten ausgewählt werden.

Dagegen ist z.B. als spezielle mathematische Eigenschaft nach dem weiter oben Gesagten für das Eulerproblem bekannt,dass ein geschlossener Eulergraph aus mehreren Kantendiskjunkten Kreisen besteht:Durchläuft man nun vom Startknoten immer weiter eine Kantenfolge unter Benutzung immer weiterer bisher nicht durchlaufender Kanten,so gelangt man schließlich auf einer geschlossenen Kantenfolge,die mehrere der Kreise enthalten kann,zurück zum Startknoten (oder Endknoten) (der dabei mehrfach durchlaufen worden sein kann),in dem die Folge nicht mehr weiter fortgesetzt werden kann,weil auf Grund der schon besuchten Startkante alle weiteren Kanten schon durchlaufen wurden.Die beschriebene Kantenfolge muß,falls existent,von einem weiteren Kreis in mindestem einem Knoten geschnitten werden (da sonst mehrere Komponenten vorhanden sind).Dieser Kreis muß wiederum von mindestens einem (evtl. noch existenten) weiteren Kreis geschnitten werden usw.).Schreitet man deshalb vom Start-oder Endknoten nun jeweils um eine oder mehrere Kanten die (anfangs) durchlaufende Kantenfolge zurück,so muß es wieder einen (Zwischen-)Knoten Kno geben,von dem eine neue geschlossene Kantenfolge erneut vorwärts bis zum Knoten Kno durchlaufen werden kann.Ebenso erfolgt das Zurückschreiten in der neuen Kantenfolge vom eben genannten Knotens aus.Durch Wiederholung des Zurück-und Vorschreitens werden so alle geschlossenen Kantenfolgen und somit inbesondere rückwärts alle Kanten des Graphen durchlaufen (,bis der Startknoten wieder erreicht ist).Werden diese in einer Kantenliste gespeichert,erhält man die Eulerlinie.Auf diesem Verfahren beruht der folgende Algorithmus:

Zunächst wird eine neue Methode Eulerfix zum Objekt TEulergraph hinzugefügt:

```
procedure TEulergraph.Eulerfix(Kant:TInhaltskante;Kno,Zielknoten:TInhaltsknoten;var Kli-
ste:TKantenliste;
  Flaeche:TCanvas;Ausgabe:TLabel);
var Index:Integer;
    ZKno:TInhaltsknoten;
    Ka:TInhaltskante;
begin
  Application.Processmessages;
  if not Kno.AusgehendeKantenliste.Leer then
    for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do                    1)
    begin
      Ka:=TInhaltskante(Kno.AusgehendeKantenliste.Kante(Index));
      if (not Ka.Besucht)  then
      begin
        ZKno:=TInhaltsknoten(Ka.Zielknoten(Kno));
        Ka.Pfadrichtung:=ZKno;
        Ka.Besucht:=true;
        if Demo then
```

```
        begin
          Ka.Farbe:=clred;
          Ka.Stil:=psdot;
          Ka.ZeichneKante(Flaeche);
          Demopause;
        end;
        if Abbruch then exit;
          Eulerfix(Ka,Zkno,Zielknoten,Kliste,Flaeche,Ausgabe);              2)
      end;
    end;
   if Kant<>nil then                                                          3)
   begin
     if Demo then
     begin
       Kant.Farbe:=clblue;
       Kant.Stil:=psdot;
       Kant.ZeichneKante(Flaeche);
       Demopause;
     end;
     Kliste.AmAnfanganfuegen(Kant);                                           4)
     if Demo then Ausgabe.Caption:=KListe.Kantenlistealsstring;
   end
end;
```

Erläuterung des Algorithmus:

Bei 1) und 3)werden die Kanten und Knoten des Graphen vom Startknoten bzw. von einem (Zwischen-)Knoten Kno aus,die noch nicht besucht worden sind,rekursiv nacheinander durchlaufen und jeweils als besucht markiert.Dies funktioniert bis der Start-bzw. Endknoten bzw. der oben genannte Knoten Kno (der evtl. schon mehrfach durchlaufen wurde) als „Sackgasse wieder erreicht ist.Bei 2) wird dann über die zu dem aktuellen Knoten führende Kante die Rekursionsebene bzw. der Knoten wieder verlassen und in der vorigen Ebene (beim vorigen Knoten) wird das gleiche Verfahren erneut angewendet (eventuell kann hier z.B. nur zurückgeschritten werden).Bei 4) werden die Kanten beim Rückwärtsschreiten gespeichert.Der Algorithmus funktioniert auch bei offenen Eulerlinien.

Es liegt auf der Hand den vorliegenden Algorithmus mit dem Backtracking-Verfahren zu vergleichen:Wendet man beide Algorithmen (Menü Anwendungen/Euler/Eine Lösung/Schneller Algorithmus ja/nein auf einen umfangreichen Graph an,wartet man man Bachtracking-Verfahren sehr lange auf eine Lösung,während die Lösung des eben beschriebenen Algorithmus sofort da ist.

Eine Begründung gibt eine Abschätzung des Laufzeitverhaltens der Algorithmen,die hier besonders einfach ist und deshalb im Unterricht durchgeführt werden sollte.

Legt man als „Worst Case" einen vollständigen (schlichten) Graphen zugrunde,durchsucht der Backtracking-Algorithmus alle Möglichkeiten (d.h. über alle zunächst vom Startknoten und danach den folgenden Knoten ausgehenden Kanten).Dies führt bei n Knoten zu $(n-1)^{(n-1)}$ Möglichkeiten,da die Kantenfolgen nur bis zur Länge n verfolgt werden.Der Eulerfix-Algorithmus durchläuft dagegen alle Kanten des Graphen genau zweimal (einmal vorwärts und einmal rückwärts).

Also erhält man folgende Aufwandsabschätzungen: $o((n^{(n-1)})$ und $o(n)$.Für n=30 ergeben diese Terme: $6,86\ 10^{42}$ und 30.Wie man sieht ist bei solchen Graphen,nur der Algorithmus Eulerfix brauchbar.Der Schüler sieht an Hand dieses Beispiels,welchen großen Einfluß die geeignete mathematische Idee auf den Ablauf eines Algorithmus hat.Da beim Hamiltonverfahren das gleiche Backtracking-Verfahren verwendet wurde,ist klar,wie wünschenswert auch hier (die bisher nicht existierende) mathematische Idee zur Vereinfachung wäre.An dieser Stelle sollte natürlich jetzt das Problem der NP-Vollständigkeit besprochen werden.

Zum Schluß noch die Nachteile des obigen Algorithmus:
1)Er berechnet nur eine Lösung des Eulerproblems (Backtracking:alle).
2)Bei Vorliegen eines Nicht-Eulergraphs terminiert er nicht sofort von selber,sondern gibt stattdessen eine falsche Lösung aus.(Daher ist eine Überprüfung mit dem Knotengradkriterium vorher nötig.)

C VI Graph als Netzplan

Das Thema dieses Kapitels ist aus dem Gebiet des Operations Research gewählt.Es wird ein Optimierungsproblem behandelt,nämlich das Bestimmen der minimalen Zeit,in der ein Gesamtvorgang ausgeführt werden kann,zu dessen Fertigstellung die gegenseitige Reihenfolge von Tätigkeiten zueinander als auch deren Zeitdauer festgelegt sind.

Während im herkömmlichen Mathematikunterricht nur hauptsächlich Optimierungsprobleme behandelt werden,die auf Extremwertaufgaben von Funktionen bzw. Funktionsgraphen führen und mit Hilfe der Methoden der Analysis gelöst werden,ist dieses Optimierungsproblem in seinem Lösungsverfahren fundamental verschieden und deshalb als alternatives Lösungsverfahren von grundsätzlicher didaktischer Bedeutung.

Es handelt sich hier um ein Optimierungsverfahren der diskreten, endlichen Mathematik,während in der Analysis immer der Aspekt des Unendlichen im Vordergrund steht.Weitere Optimierungsverfahren wie z.B. das Maximalflussproblem,das minimale Kostenproblem,das Transportproblem,das Problem des maximalen oder optimalen Matchings,das chinesische Briefträgerproblem werden in späteren Kapiteln behandelt.Das hier behandelte Problem des Netzwerkzeitplans zeichnet sich gegenüber den eben genannten Verfahren durch die besondere Einfachheit und Überschaubarkeit des Algorithmus aus.

Außerdem läßt sich der Algorithmus auch manuell-anschaulich auf einem Graphen mit Papier und Bleistift ausführen.

Aufgabe C VI.1 (Einstiegsproblem):

In ein Haus soll eine Ölheizung neu eingebaut werden.Dazu sind die in der folgenden Tabelle mit ihrer Ausführungszeit aufgeführten Tätigkeiten notwendig:

Nummer:	Beschreibung:	Zeit:
1	Heizkessel aufbauen	9
2	Warmwasserboiler aufbauen	5
3	Wasserrohre legen	6
4	Rohre mit Kessel und Boiler verbinden	3
5	Ölbrenner am Kessel montieren	8
6	Wasserrohranlage vervollständigen	2
7	Elektroanschluß für Brenner legen	2
8	Heizkörper installieren	7
9	Elektrosteuerung der Anlage installieren	6
10	Heizöltank aufbauen und installieren	12

Die Reihenfolge der Arbeiten kann nicht beliebig gewählt werden,sondern ist durch folgende Bedingungen festgelegt:

(> bedeutet zeitlich nach,< bedeutet zeitlich vor.)

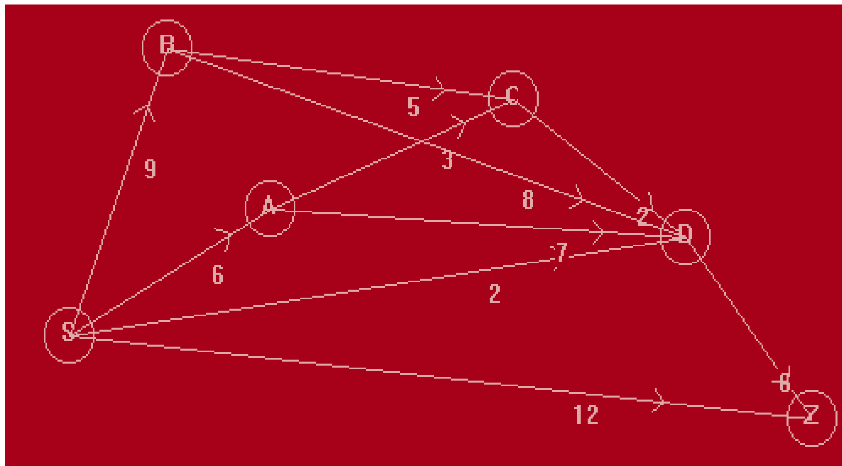
1<2;1<5;2<6;6<9;3<4;4<6;5<9;7<9;3<8;8<9

Außerdem sind alle Tätigkeiten nach einen Startzeitpunkt S und vor einem Zielzeitpunkt Z auszuführen.

Am übersichtlichsten lassen sich die Reihenfolgebedingungen durch einen Graph veranschaulichen.In diesem Graph bedeuten die Kanten die oben genannten Tätigkeiten (Vorgänge).Die Bewertung der Kanten ist die für ihre Ausführung benötigte Zeit. Die zeitliche Reihenfolge der Tätigkeiten in Relation zueinander wird durch die Richtung der Kanten vorgegeben.
Es ergibt sich dann folgende Zuordnung zu den Kanten:

Nummer/ Kante:		Beschreibung:
1	SB	Heizkessel aufbauen
2	BC	Warmwasserboiler aufbauen
3	SA	Wasserrohre legen
4	AC	Rohre mit Kessel und Boiler verbinden
5	BD	Ölbrenner am Kessel montieren
6	CD	Wasserrohranlage vervollständigen
7	SD	Elektroanschluß für Brenner legen
8	AD	Heizkörper installieren
9	DZ	Elektrosteuerung der Anlage installieren
10	SZ	Heizöltank aufbauen und installieren

Graph:



G034.gra

Aufgabe C VI.1:(Einstiegsproblem Fortsetzung)

In welcher kleinsten Zeit kann die Heizung unter Einhaltung der Reihenfolgebedingungen komplett aufgebaut werden?

Zu welchem Zeitpunkt muß jeweils eine der Tätigkeiten 1 bis 10 frühestens bzw. spätestens beginnen oder enden,wenn der minimale Zeitplan eingehalten werden soll?

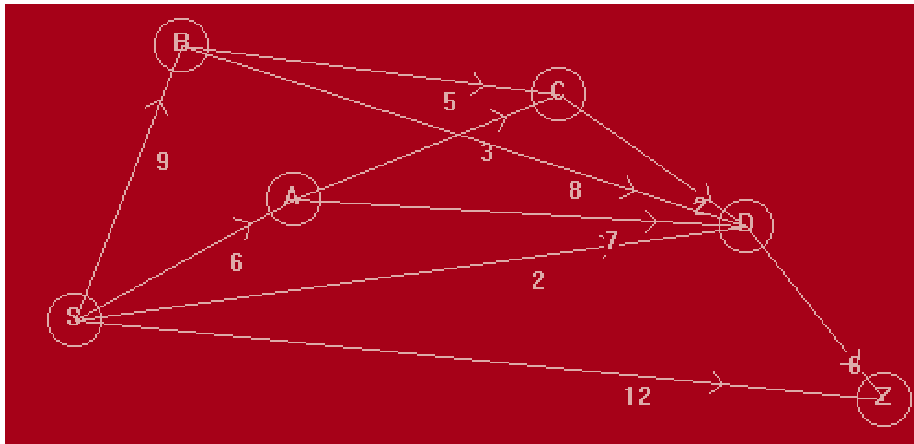
a)Löse die Aufgabe durch Probieren.

b)Löse die Aufgabe mit Hilfe des Algorithmus des Menüs Anwendungen/Netzwerkzeitplan mit dem Programm Knotengraph.

Lösung:

Minimale Zeit: 23

Die Markierung zeigt die Vorgangsfolge,die die minimale Zeit bestimmt (markiert sind die kritischen Knoten bzw. Kanten, Definition s.u.):



G034.gra

Eine Übersicht,zu welchen Zeitpunkten die Ereignisse S bis Z für eine Minimalzeit 23 frühestens und spätestens eintreten müssen,zeigt folgende Tabelle:

Knoten:

S: Anfang: 0 Ende: 0 Puffer: 0 kritisch
C: Anfang: 14 Ende: 15 Puffer: 1
A: Anfang: 6 Ende: 10 Puffer: 4
D: Anfang: 17 Ende: 17 Puffer: 0 kritisch
Z: Anfang: 23 Ende: 23 Puffer: 0 kritisch
B: Anfang: 9 Ende: 9 Puffer: 0 kritisch

Kanten:

S-B Früh. Anfg: 0 Früh. Ende:9 Spät. Anfg: 0 Spät. Ende: 9 Puffer: 0 kritisch
S-A Früh. Anfg: 0 Früh. Ende:6 Spät. Anfg: 4 Spät. Ende: 10 Puffer: 4
S-D Früh. Anfg: 0 Früh. Ende:2 Spät. Anfg: 15 Spät. Ende: 17 Puffer: 15
S-Z Früh. Anfg: 0 Früh. Ende:12 Spät. Anfg: 11 Spät. Ende: 23 Puffer: 11
C-D Früh. Anfg: 14 Früh. Ende:16 Spät. Anfg: 15 Spät. Ende: 17 Puffer: 1
A-C Früh. Anfg: 6 Früh. Ende:9 Spät. Anfg: 12 Spät. Ende: 15 Puffer: 6
A-D Früh. Anfg: 6 Früh. Ende:13 Spät. Anfg: 10 Spät. Ende: 17 Puffer: 4
D-Z Früh. Anfg: 17 Früh. Ende:23 Spät. Anfg: 17 Spät. Ende: 23 Puffer: 0 kritisch
B-C Früh. Anfg: 9 Früh. Ende:14 Spät. Anfg: 10 Spät. Ende: 15 Puffer: 1
B-D Früh. Anfg: 9 Früh. Ende:17 Spät. Anfg: 9 Spät. Ende: 17 Puffer: 0 kritisch

Eine Erklärung der verwendeten Zeitbegriffe gibt die folgende Definition:

Definition C VI.1:

Ein mit nichtnegativen Zahlen Kantenbewerteter,gerichteter Graph kann als Netzplan aufgefaßt werden,wenn gilt:

Die Kantenrichtungen induzieren eine Reihenfolgerelation (vor bzw. nach), in der die den Kanten zugeordneten Vorgänge in zeitlicher Reihenfolge ablaufen sollen.
Vorgegeben ist eine Startzeit und eine Endzeit einer Menge von Vorgängen.Alle Kantenbewertungen bedeuten Zeiten für Vorgänge,die erst nach oder zu der Startzeit beginnen können und vor bzw. zu der Endzeit beendet sein müssen.In dem Graph gibt es genau zwei Knoten,den Start-und Zielknoten,denen jeweils eine Anfangszeit- bzw. Endzeit zugeordnet wird.Der Startknoten ist dabei der einzige Knoten mit nur ausgehenden Kanten,der Endknoten ist der einzige Knoten mit nur eingehenden Kanten.

Allen übrigen Knoten des Graphen werden die Zeiten (Ereignisse) zugeordnet,in denen jeweils frühestens alle zu ihnen führenden Vorgänge (Tätigkeiten) abgeschlossen sein können (Anfang des Ereignisses) oder aber spätestens alle zu ihnen führenden Vorgänge (Tätigkeiten) abgeschlossen sein müssen (Ende des Ereignisses),damit danach alle von ihnen ausgehenden Vor-

gänge (Tätigkeiten) auf Grund der Reihenfolgerelation der Kanten (Vorgänge) neu beginnen können.

Der Puffer ist dann die Differenz zwischen Ende und Anfang des Ereignisses.

Wenn der Puffer Null ist,heißt der Knoten (das Ereignis) kritisch.
(Der Puffer muß immer nichtnegativ sein,sonst wurde die Endzeit zu klein gewählt.)

Den Kanten (d.h. den Vorgänge bzw. Tätigkeiten) werden folgende Zeiten zugeordnet:

Frühstmögliche Anfangszeit:

Das ist die Zeit,zu der dieser Vorgang frühestens (auf Grund der Vorgabe der Startzeit und der Zeitrelation der Kanten untereinander) beginnen kann.Diese Zeit ist identisch mit der Zeit Anfang des Ereignisses des Startknotens der Kante.

Spätmöglichste Endzeit:

Das ist die Zeit,zu der dieser Vorgang spätestens (auf Grund der Vorgabe der Endzeit und der Zeitrelation der Kanten untereinander) beginnen muß.Diese Zeit ist identisch mit der Zeit Ende des Ereignisses des Endknotens der Kante.

Pufferzeit:

Diese Zeit ist die Differenz zwischen der Zeit Ende des Ereignisses des Endknotens der Kante und der Zeit Anfang des Ereignisses des Startknotens der Kante vergrößert um die Vorgangszeit der Kante.(Wenn diese Pufferzeit bei einer Kante negativ sein sollte,gibt es keine Netzplanlösung,da die Endzeit zu klein gewählt war.)

Wenn die Pufferzeit Null ist,muß der Vorgang,der zur Kante gehört,genau im Zeitpunkt Anfang des Startknotens beginnen und im Zeitpunkt Ende des Ereignisses des Endknotens der Kante enden.

Eine Kante heißt kritisch,wenn die Pufferzeit Null ist.

Wenn die Pufferzeit größer als Null ist,ist eine Zeitverschiebung des zeitlichen Anfangs-bzw. des zeitlichen Ende (oder beides) des Vorgangs innerhalb des Zeitintervalls Anfang des Startknotens bis Ende des Endknotens möglich.

Daraus ergeben sich sinnvoller Weise folgenden weitere Zeitdefinitionen:

Spätmöglichste Anfangszeit:

Das ist die Zeit,zu der dieser Vorgang spätestens (auf Grund der Vorgabe der Zeit Ende des Endknotens und der Vorgangszeit der Kante) beginnen muß.
Die Zeit berechnet sich als Differenz aus Zeit Ende des Endknotens der Kante vermindert um die Vorgangszeit der Kante.

Frühstmögliche Endzeit:

Das ist die Zeit,zu der dieser Vorgang frühestens (auf Grund der Vorgabe der Zeit Anfang des Anfangsknotens und der Vorgangszeit der Kante) beendet sein kann.

Die Zeit berechnet sich als Summe aus Zeit Anfang des Anfangsknotens der Kante vergrößert um die Vorgangszeit der Kante.

Bedienung des Ablauf des Algoritmus des Menüs Anwendungen/Netzwerkzeitplan:

Der Algorithmus erkennt selbstständig den Start-und Zielknoten und fordert den Benutzer auf,die Start-und Endzeit einzugeben.Bei beiden ist der Vorgabewert Null.Wird Null als Zeit auch für den Zielknoten übernommen,wird gerade die minimale mögliche Gesamtzeit für die Endzeit des Zielknoten und damit für den gesamten Ablauf als Vorgabe gewählt.

In diesem Fall gibt es einen kritische Weg aus kritischen Kanten bzw. Knoten vom Startknoten zum Zielknoten.

Kritische Kanten und Knoten werden rot-markiert angezeigt.
Wenn die Endzeit kleiner als die minmal mögliche Zeit vorgegeben wird,wird stattdessen die minimal mögliche Zeit als Vorgabe für die Endzeit des Zielknotens gewählt.

Ansonsten werden die vom Benutzer eingegebenen Werte als Berechnungsgrundlage gewählt.

Alle Ergebnisse können durch Mausklick mit der linken Maustaste auf die einzelnen Knoten oder Kanten des Graphen nach Ablauf des Algorithmus angezeigt werden und werden auch als Ergebnis im Ausgabefenster sowie durch Klick auf die Panel-Leiste sichtbar.

Die Darstellung der Vorgänge als Kanten und der Ereignisse als Knoten eines Graphen heißt CPM-Methode („critical Path method“ auf Grund des kritischen Pfades).

Eine andere äquivalente Beschreibung eines Netzplans,ist die Potentialmethode (MPM),bei der die Vorgänge als Knoten beschrieben werden und die Kanten nur die zeitliche Relation zwischen den Vorgängen beschreiben.

Da diese Netzpläne in Graphen nach der CPM-Methode umgewandelt werden können und daher keine neuen Aspekte liefern,beschränkt sich das Programm Knotengraph nur auf die Darstellung der CPM-Methode.

Wie sieht nun ein Algorithmus aus,der die Ergebnisse zu einem Netzplan-Graphen systematisch bestimmt?

Aufgabe C VI.2 (Einstiegsproblem und problemorientierte Erarbeitung):

Beobachte den Ablauf des Algorithmus im Menü Anwendungen/Netzwerkzeitplan des Programms Knotengraph im Demomodus an Hand des obigen Graphen der Aufgabe C VI.1, und versuche die Schritte des Algorithmus zu erkennen.
Vollziehe den Algorithmus per Hand an dem Graphen nach.

Lösung:

Beobachtung und problemorientierte Erarbeitung:

Die Knoten S B A C D und Z werden nacheinander blau markiert und in Ihnen erscheinen als Werte die Zahlen 0 9 6 14 17 23.Sie lassen sich deuten als Maximum der Summen des Wertes des Ausgangsknotens und des Wertes der von dem Knoten ausgehenden Kanten aller Kanten,die in den gerade beobachteten Knoten einlaufen,z.B. anfangs bei Knoten A 0+6=6 oder später bei Knoten D Maximum(9+8,14+2)=17.Sobald der Knoten Z mit einem Wert (23) versehen ist,läuft das Markierungsverfahren rückwärts:23 15 10 9 0.Dabei werden allerdings den gerade aktuellen Knoten das Minimum aus den Differenzen der Werte der Knoten und des Wertes der in den Knoten einlaufenden Kante aller von einem Knoten ausgehenden Kanten als Wert zugeordnet,z.B: dem Knoten A Minimum(17-7,15-3)=10.Auf diese Weise erhält der Knoten S den Wert 0 und der Knoten Z den Wert 23.

Das Verfahren läßt sich dann folgendermaßen systematisieren:

Algorithmus Netzzeitplan

Verbale Beschreibung:

1)Der Algorithmus beginnt beim Startknoten (dieser Knoten wird markiert) und bestimmt jeweils für jeden Knoten,bei dem schon alle Anfangsknoten der zu ihm führenden einlaufenden Kanten berechnet sind (diese Knoten werden markiert) die Zeit Anfang des Knotens als Maximum der Zeiten Anfang der Anfangsknoten vergrößert um die Vorgangszeit der jeweiligen Kanten.

(Technik:Die Knoten werden dazu in einer Liste gespeichert.Der letzte Knoten wird jeweils entfernt,auf Berechenbarkeit überprüft und, wenn er nicht berechenbar ist (dann sind nicht alle Anfangsknoten der zu ihm führenden Kanten markiert),wird er wieder am Anfang der Liste angefügt.Wenn er berechenbar ist,wird er markiert und nicht in die Liste eingefügt.Das Verfahren ist zu Ende,wenn die Liste leer ist.)

So ergibt sich die Zeit Anfang des Zielknotens.

(Lösche alle Markierungen)

2)Setze diese Zeit als Zeit Ende des Zielknotens.

3)Falls die vorgegebenen Endzeit des Projekts größer ist als die eben ermittelte Zeit Ende,setze diese Zeit als Zeit Ende für den Endknoten.

4)Bestimme jeweils für jeden Knoten vom Zielknoten (dieser Knoten wird markiert) rückwärts ausgehend,bei dem schon alle Endknoten der von ihm wegführenden auslaufenden Kanten berechnet sind (diese Knoten werden markiert) die Zeit Ende des Knotens als Minimum der Zeiten Ende der Zielknoten vermindert um die Vorgangszeit der jeweiligen Kanten.

(Technik:Die Knoten werden dazu in einer Liste gespeichert.Der letzte Knoten wird jeweils entfernt,auf Berechenbarkeit überprüft und, wenn er nicht berechenbar ist (dann sind nicht alle Zielknoten der von ihm ausgehenden Kanten markiert),wird er wieder am Anfang der Liste angefügt.Wenn er berechenbar ist,wird er markiert und nicht mehr in die Liste eingefügt.Das Verfahren ist zu Ende,wenn die Liste leer ist.)

So ergibt sich auch die Zeit Ende des Startknotens.

5)Aus Start und Ende jedes Knotens lassen sich dann alle anderen Größen von Knoten und Kanten wie Puffer,Pufferzeit und die Anfangs-und Endzeiten (vgl. Definition VI.1) bestimmen.

Der Graph darf bei diesem Algorithmus keine Kreise enthalten.

Die verwendeten Datentypen sind TNetzknoten,TNetzKante und TNetzgraph,die sich von TInhaltsknoten,TInhaltskante und TInhaltsgraph durch Vererbung ableiten:

Quellcode:

```
TNetzknoten = class(TInhaltsknoten)
  private
    Anfang_:Extended;
    Ende_:Extended;
    Puffer_:Extended;
    Ergebnis_:string;
    procedure SetzeAnfangszeit(Z:Extended);
    function WelcheAnfangszeit:Extended;
    procedure SetzeEndzeit(Z:Extended);
    function WelcheEndzeit:Extended;
    procedure SetzePuffer(Z:Extended);
    function WelcherPuffer:Extended;
    procedure SetzeErgebnis(S:string);
    function WelchesErgebnis:string;
  public
```

```
    constructor Create;
    property Anfang:Extended read WelcheAnfangszeit write setzeAnfangszeit;
    property Ende:Extended read WelcheEndzeit write setzeEndzeit;
    property  Puffer:Extended read WelcherPuffer write setzePuffer;
    property Ergebnis:string read WelchesErgebnis write SetzeErgebnis;
    function Wertlisteschreiben:TStringlist;override;
    procedure Wertlistelesen;override;
  end;

  TNetzkante = class(TInhaltskante)
  private
    Ergebnis_:string;
    procedure SetzeErgebnis(S:string);
    function WelchesErgebnis:string;
  public
    constructor Create;
    property Ergebnis:string read WelchesErgebnis write SetzeErgebnis;
    function Wertlisteschreiben:TStringlist;override;
    procedure Wertlistelesen;override;
  end;

  TNetzgraph = class(TInhaltsgraph)
    constructor Create;
    procedure BestimmeAnfangszeit(KnoStart:TNetzknoten;Flaeche:TCanvas);
    procedure BestimmeEndzeit(KnoZiel:TNetzknoten;Flaeche:TCanvas);
    procedure BestimmeErgebnisse(var SListe:TStringlist);
    procedure Netz(var G:TInhaltsgraph;var Oberflaeche:TForm;Flaeche:TCanvas;
    Ausgabe:TLabel;var SListe:TStringlist);
  end;
```

TNetzknoten enthält die zusätzlichen Datenfelder Anfang_,Ende_,Puffer_ und Ergebnis_ sowie zugehörige Propertys und Methoden,um auf diese Felder zugreifen zu können.Anfang bedeutet die Zeit Anfang,Ende die Zeit Ende und Puffer den Puffer zu jedem Knoten.Ergebnis kann ein Ergebnis in Form der genannten Zeiten als string aufnehmen,um es z.B. bei Mausklick auf einen Knoten des Graphen in einem Fenster anzuzeigen.

TNetzkante enthält als zusätzliches Feld ein Feld Ergebnis_ vom Datentyp string (sowie die entsprechende Property),das die Ergebnisse in Form der zu der Kanten gehörenden Zeiten (vgl. Definition VI.1) aufnehmen kann,um sie z.B. bei Mausklick auf eine Kante in einem Fenster anzuzeigen.

TNetzgraph enthält außer dem Constructor die Methoden BestimmeAnfangszeit und BestimmeEndzeit,die die Zeiten Anfang(szeit) und End(e)(dzeit) für jeden Knoten erzeugen.Die Methode BestimmeErgebnisse erzeugt die Pufferzeit in jedem Knoten,markiert die kritischen Knoten und Kanten rot gestrichelt und erzeugt alle anzuzeigenden Ergebnisse für Kanten und Knoten (gemäß Definition C VI.1),und die Methode Netz ruft alle bisher genannten Methoden nacheinander auf.

Erläuterung des Algorithmus:

```
constructor TNetzgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TNetzknoten;
  InhaltsKantenclass:=TNetzkante;
end;
```

Der Constructor von TNetzgraph erzeugt eine Instanz des Datentyps und legt die Datentypen für Knoten und Kanten im Graph als TNetzknoten und TNetzkante für eine Erzeugung durch virtuelle Constructoren fest.

```
procedure TNetzgraph.BestimmeAnfangszeit(KnoStart:TNetzknoten;Flaeche:TCanvas);
var Index:Integer;
    Speicherliste:TKnotenliste;
    Kno:TNetzknoten;
    Berechnet:Boolean;
    StartknotenZeit,NeueZeit:Extended;
begin
  LoescheKnotenbesucht;                                                    1)
  Speicherliste:=TKnotenliste.Create;                                      2)
  KnoStart.Besucht:=true;                                                  3)
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do                                4)
      Speicherliste.AmEndeanfuegen(Knotenliste.Knoten(Index));
    while not Speicherliste.Leer do                                        5)
```

```
    begin
      Speicherliste.AmEndeloeschen(TObject(Kno));                                    6)
      if not Kno.Besucht                                                             7)
      then
      begin
        Berechnet:=true;
        if not Kno.EingehendeKantenliste.Leer then                                   8)
          for Index:=0 to Kno.EingehendeKantenliste.Anzahl-1 do
            if not Kno.EingehendeKantenliste.Kante(Index).Zielknoten(Kno).Besucht
            then
              Berechnet:=false;
        if Berechnet
        then
        begin
          Startknotenzeit:=KnoStart.Anfang;
          if not Kno.EingehendeKantenliste.Leer then                                 9)
            for Index:=0 to Kno.EingehendeKantenliste.Anzahl-1 do
            begin
              NeueZeit:=
                StringtoReal(Kno.EingehendeKantenliste.Kante(Index).Wert)+
                TNetzknoten(Kno.EingehendeKantenliste.Kante(Index).Zielknoten(Kno)).Anfang;
              if Startknotenzeit<NeueZeit then Startknotenzeit:=NeueZeit;
            end;
          Kno.Besucht:=true;                                                         10)
          Kno.Anfang:=Startknotenzeit;                                               11)

        end
        else
          Speicherliste.AmAnfanganfuegen(Kno);                                       12)
      end;
    end;
  LoescheKnotenbesucht;
  Speicherliste.Freeall;
  Speicherliste:=nil;
end;
```

Bei 1) wird zunächst die Besucht-Markierung aller Knoten gelöscht,und bei 2) eine leere Knotenliste Speicherliste zur Aufnahme der zu untersuchenden Knoten erzeugt.Der Startknoten wird bei 3) als besucht markiert,und bei 4) werden alle Knoten des Graphen in der Speicherliste gespeichert.

Solange noch Knoten in der Speicherliste vorhanden sind (5),werden die folgenden Anweisungen ausgeführt:

Bei 6) wird jeweils der letzte Knoten aus der Liste entfernt.Wenn der Knoten noch nicht als besucht markiert wurde (7), und auch berechenbar ist,was bei 8) getestet wird,und bedeutet,dass die Anfangsknoten aller eingehenden Kanten schon als besucht markiert wurden,und damit die Zeit Anfang dieser Knoten bestimmt wurden,wird bei 9) die Anfangszeit Anfang des aktuellen Knoten als Maximum der Summen Anfang der Anfangsknoten vergrößert um den Wert der jeweiligen eingehenden Kante (=Vorgangszeit) bestimmt,und der Knoten bei 10) als besucht und damit als berechnet markiert.

Bei 11) wird der Zeit Anfang des Knotens die bei 9) berechnete Zeit zugewiesen.

Wenn der Knoten nicht berechnet werden konnte,wird er bei 12) wieder der Speicherliste hinzugefügt.

```
procedure TNetzgraph.BestimmeEndzeit(KnoZiel:TNetzknoten;Flaeche:TCanvas);
var Index:Integer;
    Speicherliste:TKnotenliste;
    Kno:TNetzknoten;
    Berechnet:Boolean;
    EndknotenZeit,NeueZeit:Extended;
begin
  LoescheKnotenbesucht;
  Speicherliste:=TKnotenliste.Create;
  if KnoZiel.Anfang>KnoZiel.Ende then KnoZiel.Ende:=KnoZiel.Anfang;
  Endknotenzeit:=KnoZiel.Ende;
  KnoZiel.Besucht:=true;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      Speicherliste.amEndeanfuegen(Knotenliste.Knoten(Index));
  while not Speicherliste.leer do
  begin
    Speicherliste.AmAnfangloeschen(TObject(Kno));
```

```
    if not Kno.Besucht then
    begin
      Berechnet:=true;
      if not Kno.AusgehendeKantenliste.leer then
        for Index:=0 to Kno.ausgehendeKantenliste.Anzahl-1 do
          if not Kno.AusgehendeKantenliste.Kante(Index).Zielknoten(Kno).Besucht then
            Berechnet:=false;
      if Berechnet then
      begin

        if not Kno.AusgehendeKantenliste.Leer then
          for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
          begin
            NeueZeit:=                                                        13)
              TNetzknoten(Kno.AusgehendeKantenliste.Kante(Index).Zielknoten(Kno)).Ende
              -StringtoReal(Kno.AusgehendeKantenliste.Kante(Index).Wert);
            if Endknotenzeit>NeueZeit then Endknotenzeit:=NeueZeit;
          end;
        Kno.Besucht:=true;
        Kno.Ende:=Endknotenzeit;
      end
      else
        Speicherliste.AmEndeanfuegen(Kno);
    end;
  end;
  LoescheKnotenbesucht;
  Speicherliste.Freeall;
  Speicherlite:=nil;
end;
```

Die Methode BestimmeEndzeit entspricht der Methode BestimmeAnfangszeit.Sie wird mit dem Zielknoten statt dem Anfangsknoten gestartet,und es wird jeweils die Endzeit Ende statt der Zeit Anfang der Knoten bestimmt.Bei 13) wird deshalb statt des Maximums der Summen das Minimum der Differenzen aus der Zeit Ende der Zielknoten der vom aktuellen Knoten ausgehenden Kanten vermindert um den Wert der Kante (Vorgangszeit) bestimmt,und als Zeit Ende im aktuellen Knoten gespeichert.

```
procedure TNetzgraph.BestimmeErgebnisse(var SListe:TStringlist);
var Index:Integer;
    Kno,Kno1,Kno2:TNetzknoten;
    Ka:TNetzkante;
    Q,S:string;
    T,A1,A2,E1,E2,P:Real;

begin
  SListe:=TStringlist.Create;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do                                   15)
    begin
      Kno:=TNetzknoten(Knotenliste.Knoten(Index));
      Kno.Puffer:=Kno.Ende-Kno.Anfang;
      S:=Kno.Wert+': '                                                        16)
        +' Anfang: ' + RundeStringtoString(Kno.Wertliste[1],Knotengenauigkeit)
        +' Ende: '+Rundestringtostring(Kno.Wertliste[2],Knotengenauigkeit)+
        ' Puffer: '+
        RundeStringtoString(Kno.Wertliste[3],Knotengenauigkeit);
      if Kno.Puffer=0 then                                                    17)
      begin
        Kno.Farbe:=clred;
        Kno.Stil:=psdot;
        S:=S+' kritisch';
      end;
      SListe.Add(S);
      with Kno do                                                             18)
      begin
        Q:=Wert+chr(13)+'Anfangszeit: '+
          RundeZahltoString(Anfang,Knotengenauigkeit)+chr(13)+'Endzeit: '
          +RundeZahltoString(Ende,Knotengenauigkeit)+chr(13)+
          'Puffer: '+RundeZahltoString(Puffer,Knotengenauigkeit);
        Ergebnis:=Q;
      end;
    end;
  SListe.Add('Kanten:');
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do                                   19)
    begin
      Ka:=TNetzkante(Kantenliste.Kante(Index));                               20)
      T:=StringtoReal(Ka.Wert);
      Kno1:=TNetzknoten(Ka.Anfangsknoten);
      Kno2:=TNetzknoten(Ka.Endknoten);
      A1:=Kno1.Anfang;
```

```
      E1:=A1+T;
      E2:=Kno2.Ende;
      A2:=E2-T;
      P:=E2-E1;
      S:= Kno1.Wert+'-'+Kno2.Wert+' '+
        ' Früh. Anfg: '+RundeZahltoString(A1,Kantengenauigkeit)+
        ' Früh. Ende:'+RundeZahltoString(E1,Kantengenauigkeit)+
        ' Spät. Anfg: '+RundeZahltoString(A2,Kantengenauigkeit)+
        ' Spät. Ende: '+RundeZahltoString(E2,Kantengenauigkeit)+
        ' Puffer: '+RundeZahltoString(P,Kantengenauigkeit);
       if P=0 then                                                            21)
       begin
         S:=S+' kritisch ';
         Ka.Farbe:=clred;
         Ka.Stil:=psdot;
       end;
       SListe.Add(S);
       with Ka do
       begin
         Q:= Wert                                                             22)
           +chr(13)+
           'Frühstmöglicher Anfang '+RundeZahltoString(A1,Kantengenauigkeit)+
           chr(13)+
           'Frühstmögliches Ende:'+RundeZahltoString(E1,Kantengenauigkeit)+
           chr(13)+
           'Spätmöglichster Anfang: '+RundeZahltoString(A2,Kantengenauigkeit)+
           chr(13)+
           'Spätmöglichstes Ende: '+RundeZahltoString(E2,Kantengenauigkeit)+
           chr(13)+
           'Puffer: '+RundeZahltoString(P,Kantengenauigkeit);
         if P=0 then S:=S+chr(13)+'Kritische Kante';
         Ergebnis:=Q;
       end;
    end;
  Endproc:
end;
```

Die Methode BestimmeErgebnisse bestimmt bei 15),16) und 18) zunächst die Pufferzeit und einen string,bestehend aus den Zeiten Anfang,Ende und Puffer (sowie dem Knoteninhalt und dessen Koordinaten) für jeden Knoten des Graphen als Eintrag in der Stringliste SListe für die Ausgabe im Ausgabefenster sowie für das Feld Ergebnis jeden Knotens.Bei 17) werden die kritische Knoten rot markiert.

Bei 19) und 20) werden für jede Kante des Graphen aus den Daten des Anfangs-und Endknoten der Kante mittels der Zeiten Anfang und Ende die entsprechenden Zeiten für jede Kante berechnet (d.h. frühestmöglicher Anfang usw.) und die berechneten Resultate der Kante (einschließlich Bezeichnung der Kante mit Hilfe von Anfangs-und Endknoten) als Einträge in der Stringliste SListe zur Anzeige im Ausgabefenster sowie im Feld Ergebnis der Kante gespeichert.

Die Speicherung im Feld Ergebnis dient dazu,um die Ergebnisse für Knoten und Kanten bei Mausklick mit der linken Maustaste auf die Knoten und Kanten des Ergebnisgraphen jeweils anzeigen zu können.

Bei 21) werden die kritischen Kanten rot markiert,und bei 22) das Ergebnis für die Kanten festgelegt.

Falls nicht alle genanten Ergebnisse sondern nur Teilergebnisse benötigt werden,läßt sich die Methode BestimmeErgebnisse bedeutend vereinfachen. Z.B.könnte man sich nur auf die Zeilen 15) bis 17) oder 18) oder 19) bis 22) beschränken.

```
procedure TNetzgraph.Netz(var G:TInhaltsgraph;var Oberflaeche:TForm;Flaeche:TCanvas; Ausgabe:TLabel;Var SListe:TStringlist);
label Endproc;
var Index:Integer;
    Kno1,Kno2:TNetzknoten;
    Str1,Str2,Anfangszeit,Endzeit:string;
    Anfang,Ende:Extended;
    Zaehl:Integer;
    Eingabe:Boolean;
begin
```

```
if not Leer then
if GraphhatKreise or (AnzahlungerichteteKanten>0) then                          22)
begin
  ShowMessage('Der Graph enthält ungerichtete Kanten oder Kreise.');
  exit;
end;
if not Leer then
  if (not GraphhatKreise)and (AnzahlungerichteteKanten=0) then                  23)
  begin
    Zaehl:=0;
    for Index:=0 to Knotenliste.Anzahl-1 do                                       24)
      if Knotenliste.Knoten(Index).EingehendeKantenliste.leer then
      begin
        Kno1:=TNetzknoten(Knotenliste.Knoten(Index));
        Zaehl:=Zaehl+1;
      end;
    if Zaehl<>1 then
    begin
      ShowMessage('Mehrere Anfangsknoten');
      exit;
    end;
    Anfangszeit:='0';
    Eingabe:=Inputquery('Eingabe Anfangszeit: ','Startknoten: '
      +Kno1.Wert,Anfangszeit);                                                    25)
    if (not StringistRealZahl(Anfangszeit)) or (StringistRealZahl(Anfangszeit)
      and (Abs(StringtoReal(Anfangszeit))<1.0E30))
    then
      begin
        if Anfangszeit='' then   Anfang:=0 else Anfang:=StringtoReal(Anfangszeit);
        Kno1.Anfang:=Anfang;
      end
    else
    begin
      ShowMessage('Fehler! Eingabe nicht im zulässigen numerischen Bereich!');
      Eingabe:=false;
    end;
    if Eingabe=false then goto Endproc;
    Zaehl:=0;
    for Index:=0 to Knotenliste.Anzahl-1 do                                       26)
      if Knotenliste.Knoten(Index).AusgehendeKantenliste.leer then
      begin
        Kno2:=TNetzknoten(Knotenliste.Knoten(Index));
        Zaehl:=Zaehl+1;
      end;
    if Zaehl<>1 then
    begin
      ShowMessage('Mehrere Endknoten');
      exit;
    end;
    Endzeit:='0';
    Eingabe:=Inputquery('Eingabe Endzeit: ','Endknoten: '+Kno2.Wert,Endzeit);     27)
    if (not StringistRealZahl(Endzeit)) or (StringistRealZahl(Endzeit)
      and (Abs(StringtoReal(Endzeit))<1.0E30))
    then
    begin
      if Endzeit='' then   Ende:=0 else Ende:=StringtoReal(Endzeit);
      Kno2.Ende:=Ende;
    end
    else
    begin
      ShowMessage('Fehler! Eingabe nicht im zulässigen numerischen Bereich!');
      Eingabe:=false;
    end;
    if Eingabe=false then goto Endproc;
    if (Kno1<>nil) and (Kno2<>nil) then                                       (*****)
    begin
      LoescheBild(G,TForm(Oberflaeche));
      ZeichneGraph(Flaeche);
      Ausgabe.Refresh;
      Ausgabe.Caption:='Berechnung läuft...'
      BestimmeAnfangszeit(Kno1,Flaeche);                                          28)
      LoescheBild(G,TForm(Oberflaeche));
      BestimmeEndzeit(Kno2,Flaeche);                                              29)
      LoescheBild(G,TForm(Oberflaeche));
      BestimmeErgebnisse(SListe);                                                 30)
      LoescheBild(G,TForm(Oberflaeche));
      ZeichneGraph(Flaeche);
      Ausgabe.Caption:='Kritische Knoten und Kanten markiert. Projektzeit: '
        +RundeZahltoString(Kno2.Ende-Kno1.Anfang,Knotengenauigkeit);
        ShowMessage('Projektzeit: '+RundeZahltoString(Kno2.Ende-
        Kno1.Anfang,Knotengenauigkeit));
      Endproc:
      Ausgabe.Caption:='';
```

```
        Ausgabe.Refresh;
      end;
    end;
end;
```

Der erste Teil der Methode dient lediglich der automatischen Auswahl eines Start- und Zielknoten (bis zur Markierung (*****)),die auch auf andere Art und Weise (z.B. durch direkte Vorgabe oder durch Mausklick) festgelegt werden könnten.Danach werden (ab der Markierung (*****)) die zuvor beschriebenen Methoden aufgerufen.

Die Methode Netz überprüft bei 22) und 23) zunächst,ob die Voraussetzungen für das Vorliegen eines Graphen,der als Netzplan aufgefaßt werden kann,gegeben sind.

Bei 24) wird ein Startknoten bestimmt,der nur ausgehende Kanten hat und bei 25) wird die Anfangszeit eingegeben,und als Zeit Anfang im Startknoten gespeichert.

Bei 26) wird ein Zielknoten gesucht,der nur eingehende Kanten hat,und bei 27) wird die Endzeit als Zeit Ende im Zielknoten gespeichert.

Die Anweisungen 28) bis 30) rufen dann die Methoden BestimmeAnfangszeit, BestimmeEndzeit und BestimmeErgebnisse auf und erzeugen so einen kompletten Ablauf des Algorithmus.

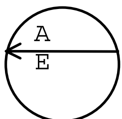
Aufgabe C VI.3:

Gegeben sind die folgenden Netzplangraphen.Bestimme als Übung jeweils die minimale Projektzeit sowie alle in Definition C V.I.1 genannten Zeiten für die Knoten und Kanten nach dem obigen Netzplan-Algorithmus,und kennzeichne den kritischen Pfad.

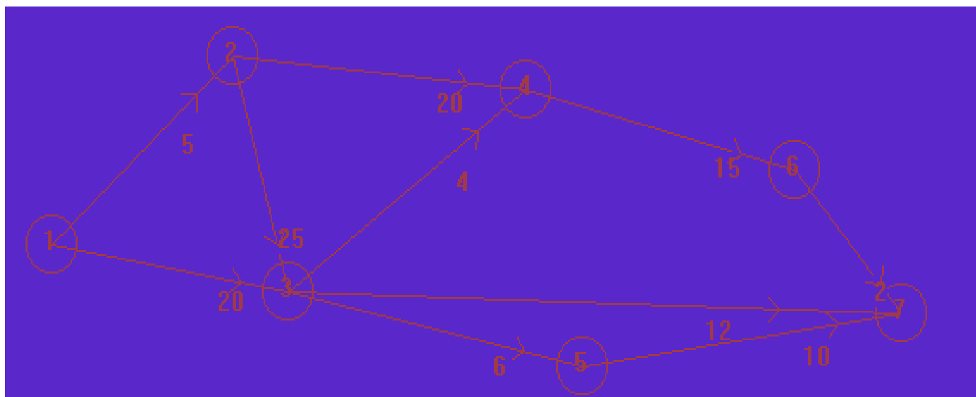
(Ermittle die Ergebnisse per Hand,mit dem Programm Knotengraph Menü Netzwerkzeitplan beziehungsweise mit dem als Programmierungsprojekt mittels der Entwicklungsumgebung selber erstellten Quellcode,je nach Konzeption).

Die Anfangszeit im Startknoten ist 0,die Endzeit im Zielknoten ist gleich der minimalen Projektzeit.

Benutze zur manuellen Berechnung einen Graph mit zweigeteilten Knoten,in die in den oberen Teil die Zeit Anfang und in den unteren Teil die Zeit Ende eingetragen werden kann.Alle anderen Resultate ergeben sich aus diesen Werten.

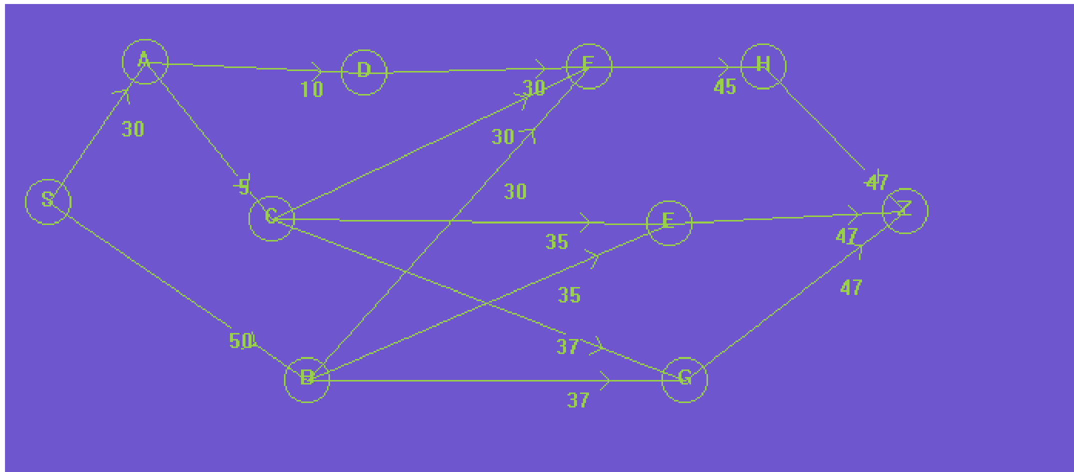


a)



G035.gra

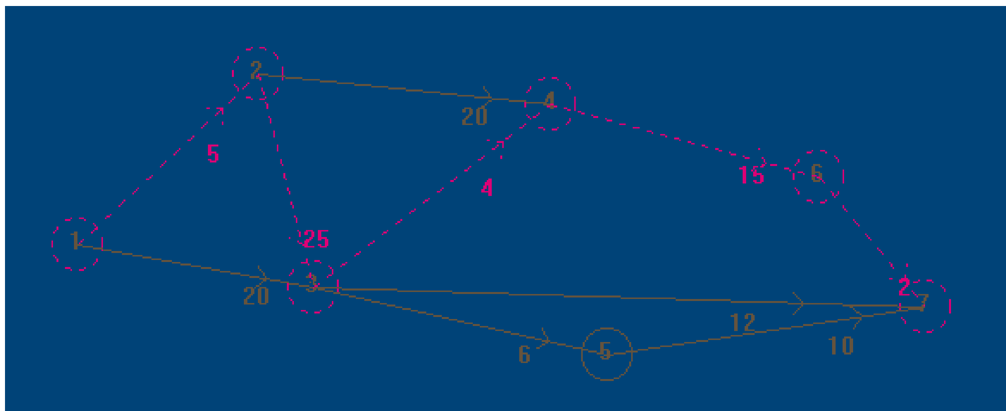
b)



G036.gra

Lösung:

a)**Minimale Projektzeit:51**



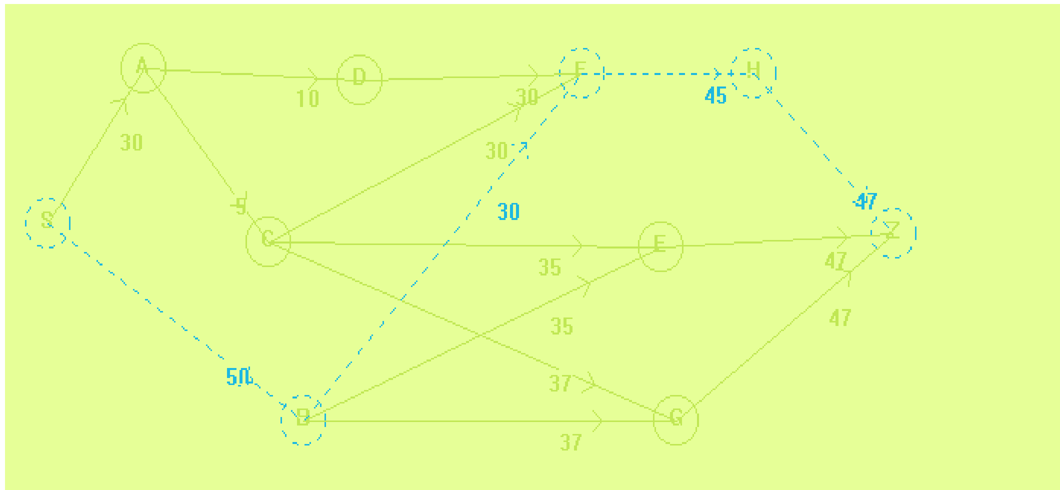
Knoten:

1: Anfang: 0 Ende: 0 Puffer: 0 kritisch
2: Anfang: 5 Ende: 5 Puffer: 0 kritisch
3: Anfang: 30 Ende: 30 Puffer: 0 kritisch
4: Anfang: 34 Ende: 34 Puffer: 0 kritisch
5: Anfang: 36 Ende: 41 Puffer: 5
6: Anfang: 49 Ende: 49 Puffer: 0 kritisch
7: Anfang: 51 Ende: 51 Puffer: 0 kritisch

Kanten:

1-2 Früh. Anfg: 0 Früh. Ende:5 Spät. Anfg: 0 Spät. Ende: 5 Puffer: 0 kritisch
1-3 Früh. Anfg: 0 Früh. Ende:20 Spät. Anfg: 10 Spät. Ende: 30 Puffer: 10
2-4 Früh. Anfg: 5 Früh. Ende:25 Spät. Anfg: 14 Spät. Ende: 34 Puffer: 9
2-3 Früh. Anfg: 5 Früh. Ende:30 Spät. Anfg: 5 Spät. Ende: 30 Puffer: 0 kritisch
3-7 Früh. Anfg: 30 Früh. Ende:42 Spät. Anfg: 39 Spät. Ende: 51 Puffer: 9
3-5 Früh. Anfg: 30 Früh. Ende:36 Spät. Anfg: 35 Spät. Ende: 41 Puffer: 5
3-4 Früh. Anfg: 30 Früh. Ende:34 Spät. Anfg: 30 Spät. Ende: 34 Puffer: 0 kritisch
4-6 Früh. Anfg: 34 Früh. Ende:49 Spät. Anfg: 34 Spät. Ende: 49 Puffer: 0 kritisch
5-7 Früh. Anfg: 36 Früh. Ende:46 Spät. Anfg: 41 Spät. Ende: 51 Puffer: 5
6-7 Früh. Anfg: 49 Früh. Ende:51 Spät. Anfg: 49 Spät. Ende: 51 Puffer: 0 kritisch

b)Minimale Projektzeit:172



Knoten:

S: Anfang: 0 Ende: 0 Puffer: 0 kritisch
A: Anfang: 30 Ende: 40 Puffer: 10
D: Anfang: 40 Ende: 50 Puffer: 10
F: Anfang: 80 Ende: 80 Puffer: 0 kritisch
H: Anfang: 125 Ende: 125 Puffer: 0 kritisch
C: Anfang: 35 Ende: 50 Puffer: 15
E: Anfang: 85 Ende: 125 Puffer: 40
Z: Anfang: 172 Ende: 172 Puffer: 0 kritisch
B: Anfang: 50 Ende: 50 Puffer: 0 kritisch
G: Anfang: 87 Ende: 125 Puffer: 38

Kanten:

S-A Früh. Anfg: 0 Früh. Ende:30 Spät. Anfg: 10 Spät. Ende: 40 Puffer: 10
S-B Früh. Anfg: 0 Früh. Ende:50 Spät. Anfg: 0 Spät. Ende: 50 Puffer: 0 kritisch
A-D Früh. Anfg: 30 Früh. Ende:40 Spät. Anfg: 40 Spät. Ende: 50 Puffer: 10
A-C Früh. Anfg: 30 Früh. Ende:35 Spät. Anfg: 45 Spät. Ende: 50 Puffer: 15
D-F Früh. Anfg: 40 Früh. Ende:70 Spät. Anfg: 50 Spät. Ende: 80 Puffer: 10
F-H Früh. Anfg: 80 Früh. Ende:125 Spät. Anfg: 80 Spät. Ende:125 Puffer:0 kritisch
H-Z Früh. Anfg: 125 Früh. Ende:172 Spät. Anfg:125 Spät.Ende:172 Puffer:0 kritisch
C-E Früh. Anfg: 35 Früh. Ende:70 Spät. Anfg: 90 Spät. Ende: 125 Puffer: 55
C-G Früh. Anfg: 35 Früh. Ende:72 Spät. Anfg: 88 Spät. Ende: 125 Puffer: 53
C-F Früh. Anfg: 35 Früh. Ende:65 Spät. Anfg: 50 Spät. Ende: 80 Puffer: 15
E-Z Früh. Anfg: 85 Früh. Ende:132 Spät. Anfg: 125 Spät. Ende: 172 Puffer: 40
B-G Früh. Anfg: 50 Früh. Ende:87 Spät. Anfg: 88 Spät. Ende: 125 Puffer: 38
B-F Früh. Anfg: 50 Früh. Ende:80 Spät. Anfg: 50 Spät. Ende: 80 Puffer: 0 kritisch
B-E Früh. Anfg: 50 Früh. Ende:85 Spät. Anfg: 90 Spät. Ende: 125 Puffer: 40
G-Z Früh. Anfg: 87 Früh. Ende:134 Spät. Anfg: 125 Spät. Ende: 172 Puffer: 38

C VII Graph als endlicher Automat

Der endliche Automat ist ein wichtiges Modell der theoretischen Informatik, um die regulären Sprachen zu beschreiben.Neben der abstrakten Beschreibung durch Zustandsmenge,Menge von Eingabezeichen,Übergangsfunktion,Anfangs-und Endzustand läßt er sich anschaulich als gerichteter Graph darstellen,wobei die Knoten die Zustände und die Kanten die Übergangsfunktion darstellen.

Um den Begriff und die Bedeutung des endlichen Automaten didaktisch-methodisch zu veranschaulichen,liegt es nahe,den Ablauf bei der Eingabe von Zeichen des Eingabalphabets mit Hilfe eines Graphen zu simulieren.So kann z.B. das Erkennen von Wörtern einer (regulären) Sprache vom Schüler experimentell nachvollzogen werden.

Außerdem regt ein solches Modell dazu an,selbst einen Automaten als Graph zu entwerfen,der geeignet ist,eine bestimmte Sprache zu erkennen.

Ist ein Graphenmodell eines solchen Automaten erst einmal aufgestellt,läßt sich aus ihm auch (durch rein formale Übersetzung) eine Grammatik (Grammatikregeln) der Sprache entwickeln.

Die Existenz von Sprachen,deren Grammatik sich nicht durch endliche Automaten beschreiben lassen,läßt sich durch die Unzulänglichkeit von verschiedenen Automaten-Graphen-Modellen beim Erkennen von Wörtern dieser Sprachen anschaulich demonstrieren (nicht beweisen),und zeigt die Grenzen des Automatenmodells.

Auf diese Weise läßt sich der endliche Automat in den Mittelpunkt einer Unterichtsreihe stellen,die zum Ziel hat,bei den Schülern Verständnis für die Arbeitsweise von Compilern oder Interpretern zu wecken und darüber hinaus eine Klassifikation von (Programmier-)Sprachen zu entwickeln.Bei der Entwicklung der Automatenbegriffs kann anschaulich vom intuitiven Alltags-Begriff des Automaten ausgegangen werden.

Obwohl der Quellcode zur Simulation des endlichen Automaten weiter unten ausführlich besprochen wird,ist es in einer Unterrichtsreihe im Fach Informatik zu diesem Thema nicht unbedingt notwendig,die Simulation neu zu programmieren.Es reicht auch die Demonstration des vorgegebenen Algorithmus bzw. das selbstständige Experimentieren mit Hilfe des **fertigen** Programms Knotengraph,Menü Anwendungen/Endlicher Automat,um ein verinnerlichtes Verständnis der Funktionsweise und der Möglichkeiten des endlichen Automaten bei den Schülern zu erreichen.

In diesem Fall genügt es also das Programm Knotengraph als fertiges Werkzeug einzusetzen.Interessant ist das eigenständige Nachvollziehen des Quellcodes allerdings dann,wenn man die Wirkungsweise von komplexeren Ereignisproceduren thematisieren will.

Aufgabe C VII.1 (Einstiegsproblem):

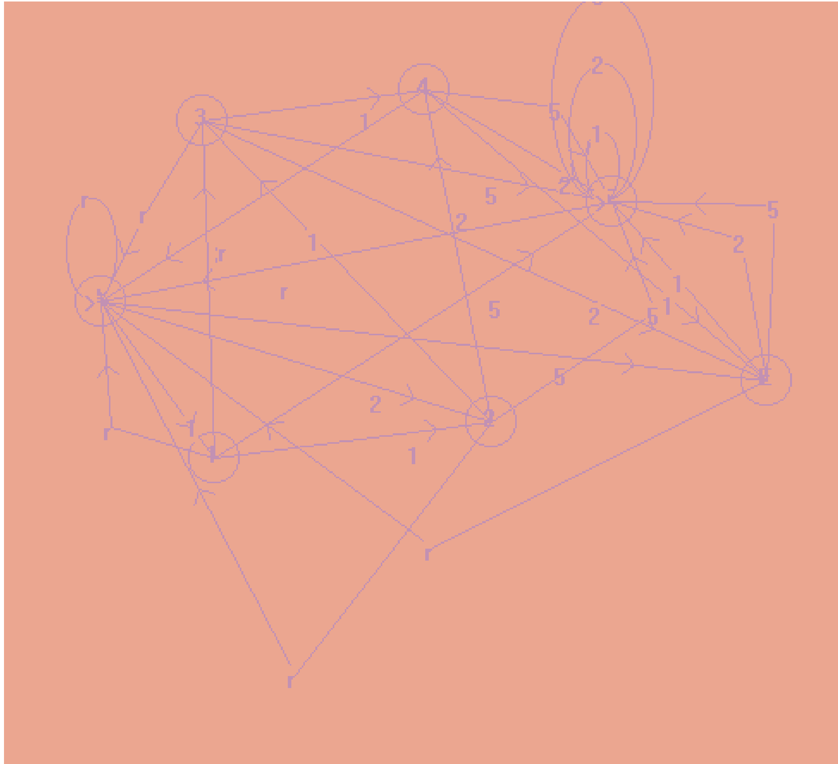
Die Schachtel Zigaretten in einem Zigarettenautomat kostet 5 DM.

Man kann Münzen im Wert von 1 DM,2 DM und 5 DM einwerfen.Wenn die Summe von 5 DM eingeworfen wurden,kann eine Schachtel Zigaretten gezogen werden,und der Vorgang ist beendet.

Es besteht außerdem jederzeit die Möglichkeit - insbesondere auch bei Überzahlung - mit dem Rückgabeknopf r den Vorgang abzubrechen und das Geld zurückzuerhalten.Danach kann neues Geld eingeworfen werden.

a)Beschreibe das Verhalten des Automaten durch einen Graphen.

Lösung:



G037.gra

Es bedeutet:

Knoten:

A: Anfangszustand

1: Zustand 1 DM eingeworfen

2: Zustand 2 DM eingeworfen

3,4: Zustand 3 bzw. 4 DM eingeworfen

E: Endzustand,5 DM eingeworfen,Zigarettenausgabe

F: Fehlerzustand

Kanten:

1,2,5 : Dieser Geldbetrag wurde eingeworfen.
r: Rückgabeknopf wurde betätigt.

b) Interpretiere das Verhalten des Automaten an Hand des Graphen:

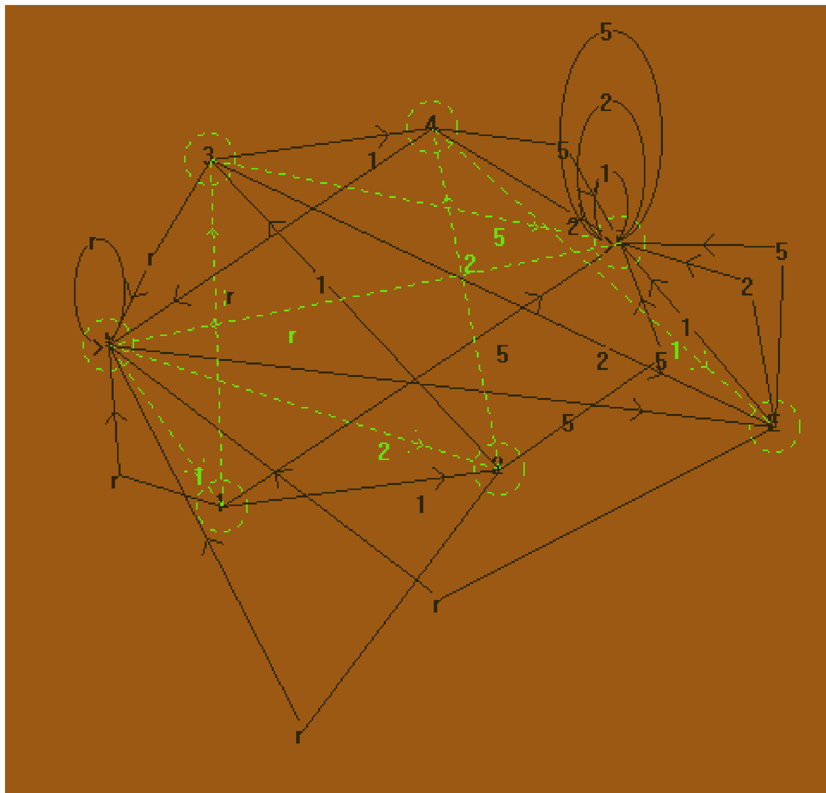
Erzeuge dazu den Graph mit Hilfe von Knotengraph,wähle den Algorithmus Anwendungen/Endlicher Automat und simuliere verschiedene Bedienungsreihenfolge mit Hilfe des Programms.

Wähle A als Anfangszustand und E als einzigen Endzustand.

Gib jeweils die Folge der durchlaufenden Zustände an.

Lösung:

z.B.:



Zustandsfolge: A 1 3 F A 2 4 E

Eingabefolge: 1 2 5 r 2 2 1

Dieses Beispiel legt folgende Definition nahe:

Definition C VII.1:

Ein endlicher,determinierter Automat T besteht aus

1)einer endlichen Menge von Zuständen K (im Graph:Knoten)
2)einem Anfangszustand A aus der Zustandsmenge K
3)einer Endzustandsmenge M,die Teilmenge von K ist
4)einer endlichen Menge von Eingabezeichen Z
5)einer Übergangsfunktion f:KxZ→K

oder T=(K,A,E,Z,f)

Für das obige Beispiel gilt:

K={A,1,2,3,4,E,F}

Anfangszustand A

Endzustandmenge:M={E}

Eingabezeichen:Z={1,2,5,r}

Die Übergangsfunktion f kann in Tabellenform dargestellt werden.Für das obige Beispiel lautet die Tabelle:

f	1	2	5	r
A	1	2	E	A
1	2	3	F	A

2	3	4	F	A
3	4	E	F	A
4	E	F	F	A
E	F	F	F	A
F	F	F	F	A

Die Übergangsfunktion wird durch die Kanten des Graphen dargestellt,die Zustände sind die Knoten des Graphen.Das Eingabealphabet wird durch die Kantenbewertungen vorgegeben.

Der Anfangszustand und die Endzustandsmenge müssen durch Mausklick vom Benutzer gewählt werden.

Die folgenden Aufgaben C VII.2 bis C VII.4 sind weitere Unterrichtsbeispiele zum Einsatz der Automatensimulation:

Das Automatenmodell hat in der Informatik deshalb eine große Bedeutung,weil es zur Syntaxanalyse von regulären Sprachen benutzt werden kann.

Definition C VII.2:

Vorgegeben sei eine endliche Menge von Eingabezeichen Z.Unter der Sprache von Z,die mit Z^* bezeichnet wird,versteht man die Menge aller nur denkbaren endlichen Symbolfolgen,die nur Zeichen aus Z enthalten einschließlich des leeren Wortes,das mit ε bezeichnet wird.

Eine reguläre Sprache R über Z ist dann eine Menge von Wörtern,für die gilt:

(Mit L(w) sei die Menge der durch das Wort w erzeugten regulären Wörter bezeichnet.)

$L(\{\})=\{\}$

Das leere Wort ε gehört zu R,und es ist $L(\varepsilon)=\{\varepsilon\}$.

Für jedes $z\in Z$ ist $L(z)=\{z\}$

Sind w und v reguläre Ausdrücke über Z so sind auch w^*,(wv),$w\cup v$ reguläre Ausdrücke.

Es gilt $L(w^*)=L(w)^*$,$L(((wv)))=L(w)L(v)$ und $L((w\cup v))=L(w)\cup L(v)$.

Bemerkung:

Es läßt sich zeigen,dass die regulären Sprachen gerade auch die Sprachen vom Typ 3 der Chomski-Grammatik-Typisierung (bzw. die Sprachen mit rechtslinearen Grammatik) sind.

Aufgabe C VII.2:

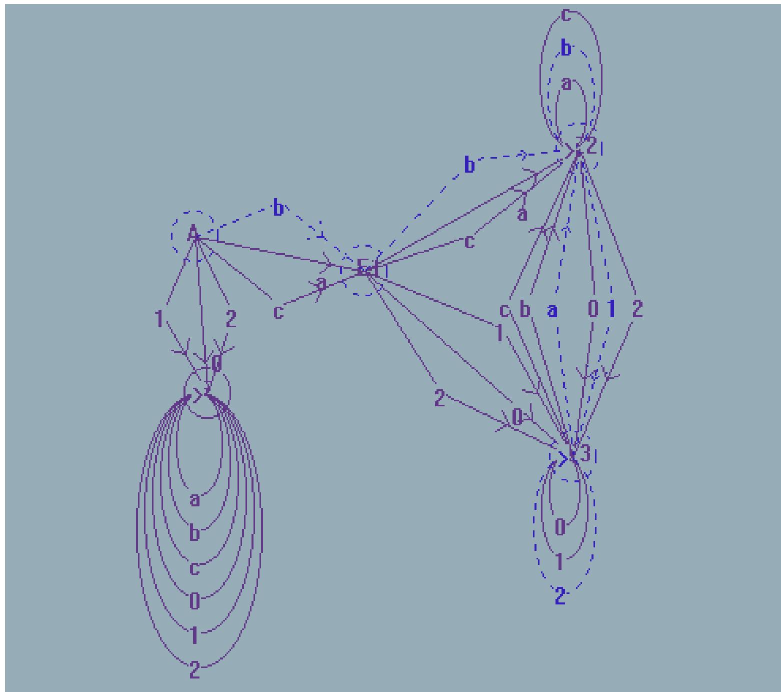
Ein Bezeichner darf in Pascal nur mit einem Buchstaben beginnen,und kann danach beliebige Buchstaben oder Ziffern enthalten.

Gesucht ist ein Automat,der die Syntax von Bezeichnern überprüfen kann.

Aus Vereinfachungsgründen wird die Ziffernmenge auf die Ziffern von 0 bis 2 und die Buchstabenmenge auf die Buchstaben a,b,c beschränkt.

Lösung:

A wird als Anfangszustand markiert und E1,E2 sowie E3 als Endzustände.F ist der Fehlerzustand.



G038.gra

Zustandsfolge: A E1 E2 E3 E3 E2 E2
Eingabefolge: b b 1 2 a b

Wenn die Eingabefolge in einem Endzustand endet,ist die Syntax korrekt.

Es soll nun zunächst gezeigt werden,wie ein endlicher Automat simuliert werden kann.Das Verfahren ergibt sich unmittelbar aus der Einstiegsaufgabe und der Definition des Automaten gemäß C VII.1.

Algorithmus zur Simulation eines endlichen Automaten:

Verbale Beschreibung:

1)Nachdem im Graphen ein Anfangsknoten als Anfangszustand und ein oder mehrere Endknoten als Endzustände gewählt wurden,wird der Startknoten als aktueller Knoten gewählt.

2)Es wird ein Zeichen aus dem Eingabealphabet eingegeben.

3)Die vom aktuellen Knoten ausgehenden Kanten werden daraufhin untersucht,ob der Kantenwert gleich dem eingegebenen Zeichen ist.Wenn dies der Fall ist,wird der Zielknoten dieser Kante bestimmt.Wenn dies nicht der Fall ist,wird eine Fehlermeldung ausgegeben,und der Ablauf wird bei 2) fortgesetzt.

4)Wenn der Zielknoten gleich einem der gewählten Endknoten ist,kann der Algorithmus (durch Auswahl einer Option in einem Eingabefenster) durch den Benutzer beendet werden.Andernfalls oder wenn der aktuelle Knoten nicht einer der Endknoten war,wird der Zielknoten als aktueller Knoten gesetzt, und der Ablauf wird bei 2) fortgesetzt.

Mit dem folgenden Quellcode wird auch die Benutzung von Ereignis-Methoden in Delphi demonstriert:

Quellcode:

Der Quellcode wird hier nur dargestellt,soweit er den Graphtyp TAutomatengraph verwendet.(TAutomatengraph wird nicht durch die entsprechenden Menüs der Form gespeichert oder geladen,wohl aber TInhaltsgraph.)Ein erweiterter zweiter Algorithmus,der die Verwendung des Graphtypen TAutomatenneugraph demonstriert,auf den durch die visuelle Vererbung des Formulars Knotenform an die Hauptform Knotenformular die Menümethoden der Form Knotenform angewendet werden können (insbesondere die Menüs Graph Laden und Graph Speichern),wird als Musterbeispiel für diese Art der Objekt-Graphtypdefinition im Anhang F I Beschreibung der Datenstruktur besprochen.Gemäß dieses Musterbeispiels können alle Anwendungen programmiert werden.Allerdings ist das Laden und Speichern eines Graphen dann gebunden an dessen spezifischen Objektyp und dessen Verwendung an die entsprechende spezifischen Aufgabenstellung.

Die Datenstruktur von TAutomatengraph besteht aus den Objekten TAutomatenknoten und TAutomatengraph,die sich jeweils von TInhaltsknoten und TInhaltsgraph durch Vererbung ableiten:

```
TKnotenart = 0..3;

  TAutomatenknoten = class(TInhaltsknoten)
    ArtdesKnoten:TKnotenart;
    constructor Create;
    procedure SetzeKnotenart(Ka:TKnotenart);
    function WelcheKnotenart:TKnotenart;
    property KnotenArt:TKnotenart read WelcheKnotenart write SetzeKnotenart;
    function Wertlisteschreiben:Tstringlist;override;
    procedure Wertlistelesen;override;
  end;

  TAutomatengraph = class(TInhaltsgraph)
    AktuellerKnoten:TAutomatenknoten;
    constructor Create;
    procedure SetzeaktuellenKnoten(kno:TAutomatenknoten);
    function WelcheraktuelleKnoten:TAutomatenknoten;
    property MomentanerKnoten:TAutomatenknoten read WelcheraktuelleKnoten
    write SetzeaktuellenKnoten;
    procedure SetzeAnfangswertknotenart;
    function BestimmeAnfangsknoten:TAutomatenknoten;
  end;
```

TAutomatenknoten enthält das Feld Knotenart_,das Werte von 0 bis 3 annehmen kann.Die Zahlen bedeuten:

0:Knoten,der weder Start- noch Endknoten ist.
1:Startknoten
2:Endknoten
3:Start-und Endknoten zugleich

TAutomatenknoten enthält das Feld AktuellerKnoten_,das den aktuellen Knoten speichert.

Auf die beiden Felder kann jeweils durch Propertys zugegriffen werden.

Die Methode SetzeAnfangswertknotenart setzt zur Initialisierung bei allen Knoten das Feld Knotenart_ auf 0.

```
procedure TAutomatengraph.SetzeAnfangswertknotenart;
var Index:Integer;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      TAutomatenknoten(Knotenliste.Knoten(Index)).Knotenart:=0;
end;
```

Die Methode BestimmeAnfangsknoten bestimmt den Anfangsknoten des Graphen,indem sie den Knoten sucht,bei dem die Knotenart gleich 1 oder 3 ist.

```
function TAutomatengraph.BestimmeAnfangsknoten:TAutomatenknoten;
var Index:Integer;
begin
  BestimmeAnfangsknoten:=nil;
  if not Leer then
  for Index:=0 to Knotenliste.Anzahl-1 do
    if (TAutomatenknoten(Knotenliste.Knoten(Index)).Knotenart=1)or
      (TAutomatenknoten(Knotenliste.Knoten(Index)).Knotenart=3)
    then
    begin
      BestimmeAnfangsknoten:=TAutomatenknoten(Knotenliste.Knoten(Index));
      exit;
    end;
end;
```

Die (Menü-)Methode EndlicherAutomatClick (von TKnotenform) bestimmt durch Mausklick den Anfangsknoten und die Endknoten,setzt den aktuellenKnoten auf den Anfangsknoten und öffnet das Eingabefenster Eingabe,in das Zeichen des Eingabealphabets eingegeben werden können.

```
procedure TKnotenform.endlicherAutomatClick(Sender: TObject);
var Sliste:TStringList;
Procedure AutomatmitTAutomatengraph;
    Automatengraph:TAutomatengraph;
    Ende:Boolean;
begin
  if Graph.Leer then exit;
  if Graph.AnzahlKomponenten>1 then
  begin
    ShowMessage('Mehrere Komponenten!');                                    1)
    exit;
  end;
  Bildloeschen;
  Graph.ZeichneGraph(Paintbox.Canvas);
  if Graph.AnzahlungerichteteKanten>0 then
  begin
    ShowMessage('Der Graph hat ungerichtete Kanten!');                      2)
    exit;
  end;
  if Graph.AnzahlKnoten<2 then                                              3)
  begin
    ShowMessage('Graph hat nur einen Knoten!');
     exit;
  end;
  Automatengraph:=TAutomatengraph(Graph.InhaltskopiedesGraphen(TAutomatengraph,
  TAutomatenknoten,TInhaltskante,false));                                   4)
  Automatengraph.SetzeAnfangswertknotenart;                                 5)
  GraphH:=Automatengraph;
  GraphH.Zustand:=false;                                                    6)
  ShowMessage('Anfangszustand mit Mausklick bestimmen');
  Paintbox.ONmousemove:=nil;
  Paintbox.OnDblclick:=nil;
  Paintbox.OnMouseDown:=Automaten1MouseDown;                                7)
  repeat                                                                    8)
    Application.Processmessages;
  until GraphH.Zustand;
  repeat
    Ende:=false;
    GraphH.Zustand:=false;                                                  9)
    ShowMessage('Endzustand mit Mausklick bestimmen');
    Paintbox.OnMouseDown:=Automaten2MouseDown;                             10)
    repeat
      Application.Processmessages;
    until GraphH.Zustand;                                                  11)
    if MessageDlg('Alle Endzustände markiert',
      mtConfirmation, [mbYes, mbNo], 0) = mrYes then Ende:=true
  until Ende;                                                              12)
  Automatengraph.MomentanerKnoten:=Automatengraph.BestimmeAnfangsknoten;   13)
  Automatengraph.ZeichneGraph(Paintbox.Canvas);
  Eingabe.Visible:=true;                                                   14)
  Button.Visible:=true;
  Ausgabe1.Caption:='Kante eingeben:';
  Automatengraph.MomentaneKantenliste:=TKantenliste.Create;                15)
  Eingabe.Setfocus;                                                        16)
  GraphH:=Automatengraph;
  Aktiv:=false;
end;
```

Bei 1),2) und 3) wird zunächst abgefragt,ob der Graph nur aus einer Komponente besteht,nur gerichtete Kanten enthält und ob der Graph mehr als einen Knoten enthält.
Bei 4) wird dann ein Graph vom Typ TAutomatengraph mit dem Knotentyp TAutomatenknoten und dem Kantentyp TAutomatenkante als Kopie auf der Variablen Automatengraph erzeugt,und bei 5) wird die Methode SetzeAnfangswertknotenart aufgerufen.

Bei 6),7) und 8) wird einem durch den Benutzer auszuwählenden Knoten des Graphen die Zahl 1 als Knotenart zugewiesen (Anfangsknoten).Dazu wird zunächst die Eigenschaft (Property) Zustand des Graphen (Feldes) GraphH vom Typ TInhaltsgraph,dessen Zeiger auf den Graphen Automatengraph zeigt,auf false gesetzt.Die Ereignismethode Automaten1MouseDown setzt Abbruch von GraphH genau dann auf true,wenn ein Knoten des Graphen durch Mausklick mit der linken Maustaste vom Benutzer ausgewählt wurde und die Knotenart dieses Knotens auf 1 gesetzt wurde.

Dasselbe Verfahren wird dann bei 9),10,11) und 12) mit der Ereignismethode Automaten2MouseDown wiederholt,um einen oder mehrere Endknoten mit der Knotenart 2 oder 3 zu markieren.Die zuvor beschriebenen Verfahrensweisen demonstrieren das parallele Ablaufen von Ereignismethoden.Unter der 16-bit Version Delphi Version 1.x ist hier die Anweisung Application.-Processmessages nötig,damit die Programmkontrolle an die verschiedenen Ereignismethoden abgegeben werden kann (ab Delphi 2.0 unter Windows 95/98 nicht mehr).Das Flag Abbruch bestimmt,wann die Repeat-Schleifen jeweils verlassen werden und die Programmausführung fortgesetzt wird.

Bei 14),15) und 16) wird das Eingabefenster (Methode:Eingabekeypressed) (zur Eingabe von Kanteninhalten) als auch der Abbruch-Button sichtbar gemacht,das Eingabefenster erhält den Focus und die Kantenliste zu Aufnahme der durchlaufenden Kantenfolge wird initialisiert.

Die Methode EingabeKeyPress liest ein Eingabezeichen von der Tastatur und vergleicht es mit den Zeichen,die in den ausgehenden Kanten des aktuellen Knotens gespeichert sind.Falls sich dort ein gleiches Zeichen findet,wird der Zielknoten dieser Kante ausgewählt,und es wird überprüft,ob der Zielknoten einer der Endzustände ist.Wenn dies der Fall ist,wird eine Meldung mit der Frage ausgegeben,ob das Verfahren abgebrochen oder weiter fortgesetzt werden soll.
Bei Fortsetzung des Algorithmus oder wenn der Zielknoten nicht einer der Endknoten ist,wird der Zielknoten zum aktuellen Knoten gemacht.
Die Kantenfolge wird in einer Kantenliste gespeichert.Diese wird ausgewertet und die Knotenfolge sowie die Kantenfolge dieser Liste werden zu der Stringliste Sliste hinzugefügt,um im Ausgabefenster angezeigt werden zu können.

```
procedure TKnotenform.EingabeKeyPress(Sender: TObject; var Key: Char);
label Endproc;
var Ka1,Ka2:TInhaltskante;
    Str,St,S:string;
    Automatengraph:TAutomatengraph;
    Index:Integer;
    ZKno:TAutomatenknoten;
    T:TInhaltsgraph;
begin
  if ord(key)=13 then
  begin
    Eingabe.Setfocus;
    Automatengraph:=TAutomatengraph(GraphH);
    Str:=Eingabe.Text;                                                      17)
    Ka2:=nil;
    if not Automatengraph.MomentanerKnoten.AusgehendeKantenliste.Leer  then
    for Index:=0 to Automatengraph.MomentanerKnoten.AusgehendeKantenliste.Anzahl-1 do
    begin
      Ka1:=TInhaltskante(Automatengraph.MomentanerKnoten.AusgehendeKantenliste.Kante(Index));
      if (Ka1.Wert=Str) or (Ka1.Wert=' '+Str) then Ka2:=Ka1;               18)
    end;
    if (Ka2<>nil) and ((Ka2.Wert=Str) or (Ka2.Wert=' '+Str)) then
    begin
      TKnoten(ZKno):=Ka2.Zielknoten(Automatengraph.MomentanerKnoten);       19)
      if Automatengraph.MomentanerKnoten.Knotenart<>1 then
```

```
      begin
        Automatengraph.MomentanerKnoten.Farbe:=clblack;
        Automatengraph.MomentanerKnoten.Stil:=pssolid;
      end;
      Ka2.Farbe:=clred;
      Ka2.Stil:=psdot;
      Automatengraph.MomentaneKantenliste.AmEndeanfuegen(Ka2);                          20)
      Automatengraph.ZeichneGraph(Paintbox.Canvas);
      Demopause;
      Ka2.Farbe:=clblack;
      Ka2.Stil:=pssolid;
      Ka2.Pfadrichtung:=ZKno;
      Automatengraph.MomentanerKnoten:=ZKno;                                             21)
      Automatengraph.MomentanerKnoten.Farbe:=clred;
      Automatengraph.MomentanerKnoten.Stil:=psdot;
      T:=TInhaltsgraph(Automatengraph.MomentaneKantenliste.Kopie.Graph);
      Ausgabe2.Caption:='Zustandsfolge: '+T.InhaltallerKnoten(ErzeugeKnotenstring);     22)
      Ausgabe2.Refresh;
      S:='Eingabefolge: '+T.InhaltallerKantenoderKnoten(ErzeugeKantenstring);
      Automatengraph.ZeichneGraph(Paintbox.Canvas);
      Eingabe.Text:='';
      if ZKno.Knotenart in [2,3] then                                                    23)
      begin
        if MessageDlg('Endzustand erreicht! Weiter?',
          mtConfirmation, [mbYes, mbNo], 0) = mrYes
        then goto Endproc;
        Eingabe.Visible:=false;                                                          24)
        Button.Visible:=false;
        Listbox.Clear;
        Listbox.Items.Add(Ausgabe2.Caption);
        Listbox.Items.Add(S);
        Ausgabeloeschen(false);
        Graph.ZeichneGraph(Paintbox.Canvas);
        Aktiv:=true;
        T.Faerbegraph(clred,psdot);
        T.ZeichneGraph(Paintbox.Canvas);
        GraphH:=Automatengraph.InhaltskopiedesGraphen(TInhaltsgraph,TAutomatenknoten,
        TInhaltsKante,false);;
        Automatengraph.Freeall;
        Automatengraph:=nil;
      end
    end
    else
    begin
      ShowMessage('ungültige Eingabe!');
       Eingabe.Text:='';
     end;
    end;
    Endproc:
    GraphH.ZeichneGraph(Paintbox.Canvas)
end;
```

Bei 17) wird der aktuelle Text des Eingabefenster bestimmt und bei 18) mit den Knoteninhalten der vom momentanen Knoten ausgehenden Kanten verglichen.Bei Übereinstimmung wird bei 19) der Zielknoten dieser Kante bestimmt, und bei 20) wird diese Kante in die Kantenliste aufgenommen.Bei 21) wird der Zielknoten zum momentanen Knoten gemacht und anschließend rot gefärbt markiert.Bei 22) wird die bisherigen Knotenfolge erzeugt und ausgegeben.

Bei 23) wird geprüft,ob der Zielknoten ein Endknoten ist.Wenn das der Fall ist,wird gefragt,ob der Algorithmus fortgesetzt werde soll oder nicht.Wenn nicht,wird das Eingabefenster und der Abbruch-Button wieder unsichtbar gemacht (24).Wenn ja,wartet die Ereignismethode von neuem,wie auch im Fall,dass der Endknoten noch nicht erreicht worden ist,auf die Eingabe eines Kanteninhalts.Außerdem wird anschließend die Knotenliste der durchlaufenden Knoten der Listbox für das Ausgabefenster hinzugefügt.

Die Methode ButtonClick macht das Eingabefenster (für die Zeichen des Eingabealphabets) unsichtbar und löscht die Labels Ausgabe1 und Ausgabe2.

```
procedure TKnotenform.ButtonClick(Sender: TObject);
begin
  Eingabe.Visible:=false;
  Button.Visible:=false;
  Ausgabeloeschen(false);
  Graph.ZeichneGraph(Paintbox.Canvas);
  Aktiv:=true;
end;
```

Die Ereignismethode Automaten1Mousedown setzt bei Mausklick mit der linken Maustaste auf einen Knoten die Knotenart dieses Knoten auf 1 (Startknoten).Wird der Mausklick auf eine Stelle gesetzt,an der sich kein Knoten befindet,wird die Meldung „Knoten mit der Maus auswählen" angezeigt.

```
procedure TKnotenform.Automaten1Mousedown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var Kno:TAutomatenknoten;
begin
  if GraphH.FindezuKoordinatendenKnoten(X,Y,TInhaltsknoten(Kno))          25)
  then
  begin
    Kno.Knotenart:=1;                                                   26)
    Kno.Stil:=psdash;
    Kno.Farbe:=clgreen;
    Paintbox.OnMouseDown:=PaintboxMouseDown;                            27)
    GraphH.ZeichneGraph(Paintbox.Canvas);
    ShowMessage('Anfangszustand: '+Kno.Wert);
    GraphH.Zustand:=true;
  end
  else
    ShowMessage('Knoten mit  Maus auswählen!');
  GraphH.ZeichneGraph(Paintbox.Canvas);
  Aktiv:=false;
end;
```

Bei 25) wird ein Knoten bestimmt,der in der Nähe der Koordinaten X und Y des Mausklicks liegt,und bei 26) wird die Knotenart dieses Knotens auf 1 gesetzt.Bei 27) wird dann die Ereignismethode für Mausklicks wieder auf die Standardmethode PaintboxMouseDown zurückgesetzt.

Die Ereignismethode Automaten2Mousedown setzt bei Mausklick mit der linken Maustaste auf einen Knoten die Knotenart dieses Knoten auf 2 oder 3 (2 bei Endknoten,wenn die Knotenart 0 war und 3 wenn der Knoten schon die Knotenart 1 hatte).Wird der Mausklick auf eine Stelle gesetzt,an der sich kein Knoten befindet,wird die Meldung „Knoten mit der Maus auswählen" angezeigt.

```
procedure TKnotenform.Automaten2Mousedown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
  var Kno:TAutomatenknoten;
begin
  if GraphH.FindezuKoordinatendenKnoten(X,Y,TInhaltsknoten(Kno))          28)
  then
  begin
    if Kno.Knotenart<> 1 then
    begin
      Kno.Knotenart:=2;                                                 29)
      Kno.stil:=psdashdot;
      Kno.Farbe:=clblue;
      GraphH.ZeichneGraph(Paintbox.Canvas);
    end
    else
    begin
      ShowMessage('Knoten ist auch Anfangszustand!');
      Kno.Knotenart:=3;                                                 30)
    end;
    Paintbox.OnMouseDown:=PaintboxMouseDown;                            31)
    ShowMessage('Endzustand: '+Kno.Wert);
    GraphH.Zustand:=true;
  end
    else
      ShowMessage('Knoten mit Maus auswählen!');
  GraphH.ZeichneGraph(Paintbox.Canvas);
  Aktiv:=false;
end;
```

Bei 28) wird ein Knoten bestimmt,der in der Nähe der Koordinaten X und Y des Mausklicks liegt,und bei 29) oder 30) wird der Knotenart dieses Knotens 2 oder 3 zugewiesen.Bei 31) wird dann die Ereignismethode für Mausklicks wieder auf die Standardmethode PaintboxMouseDown zurückgesetzt.

Aufgabe C VII.3:

Die Wörter der Sprache Dualzahlen mit Vorzeichen sind von der Form +1011 bzw. -1000011 bzw. 11000.
Die Syntax wird durch die folgende rechtsreguläre Grammatik beschrieben:

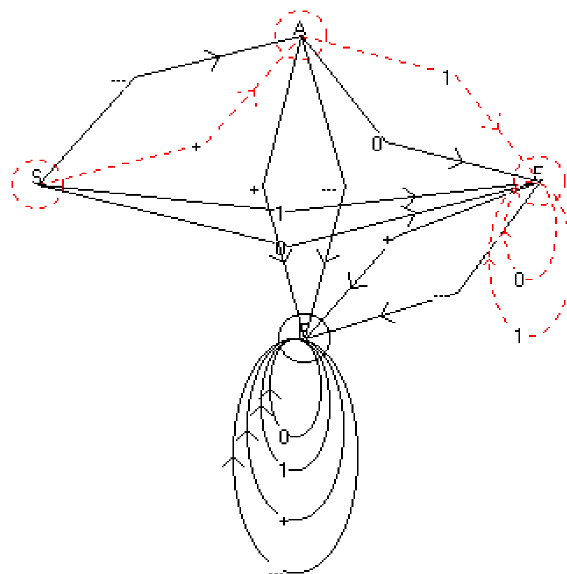
```
S--->+A|-A
S--->0E|1E
E--->0E|1E
E---->0|1
A--->+F|-F
E--->+F|-F
F-->+F|-F|0F|1F
```

Konstruiere mit Hilfe von Knotengraph einen endlichen Automaten(graph),der die Wörter der durch die Grammatik beschriebenen Sprache erkennt,und teste den Automaten bezüglich der von ihm erkannten Sprache.
(vgl. Lit 43, S.131)

Lösung:

Das Nichtterminalzeichen A,E,F und das Startsymbol S lassen sich als Knoten des Graphen,die Pfeile ---> als Kanten zwischen entsprechenden Knoten und die Terminalzeichen +,-,0,1 als Kantenwerte interpretieren.

Das Regelsystem (Überführungsfunktion) der Grammatik läßt sich dann durch einen Graph darstellen:



G039.gra

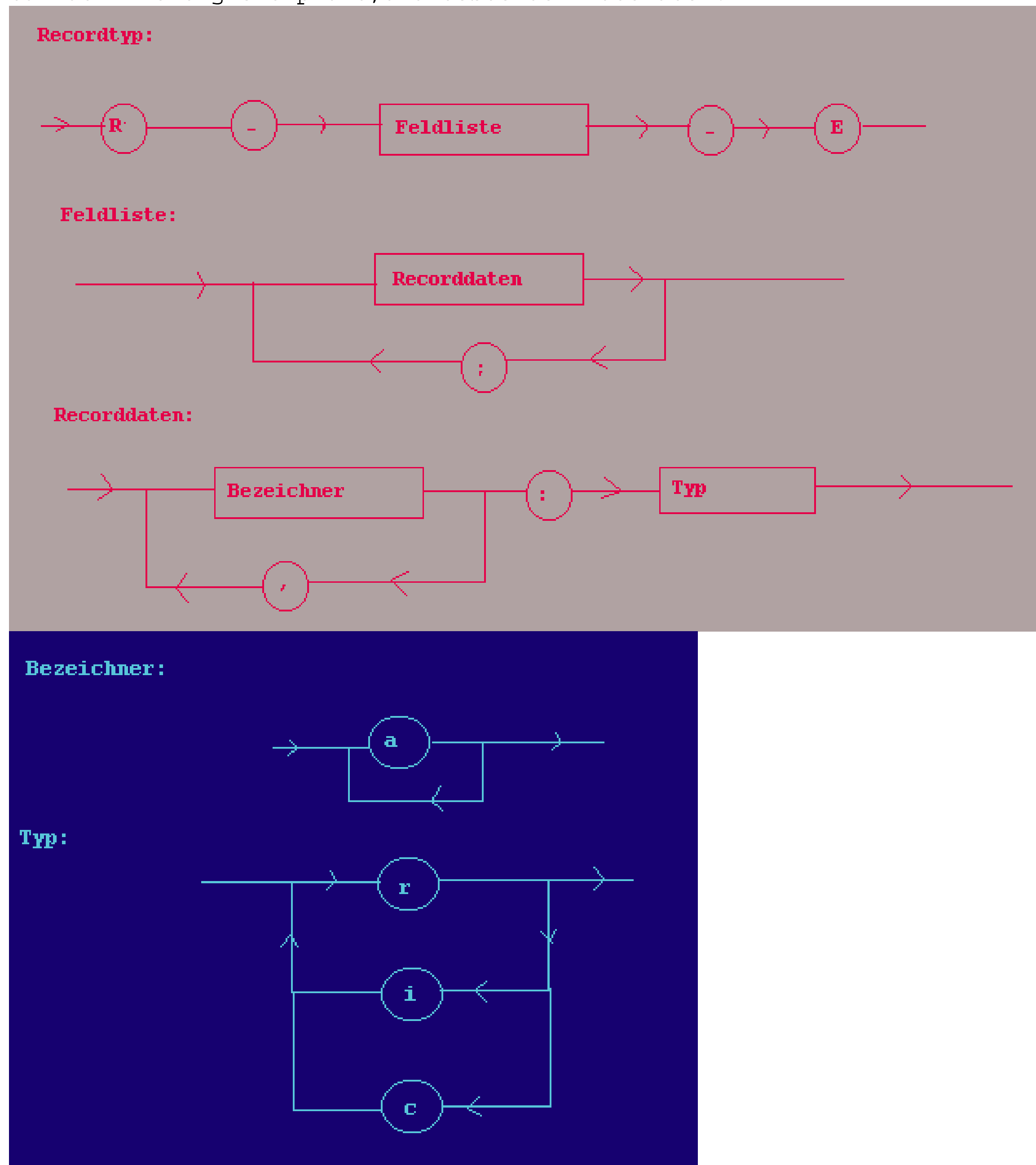
z.B.:

Zustandsfolge: S A E E E
Eingabefolge: + 1 0 1 (erkanntes Wort)

Aufgabe C VII.4:

Gegeben ist das folgende Syntaxdiagramm,das die vereinfachte Definition eines Pascal/Delphi-Records darstellt.

Erzeuge dazu mit Knotengraph einen endlichen Automaten(graph),der die Syntax auf Richtigkeit prüft,und teste den Automaten.



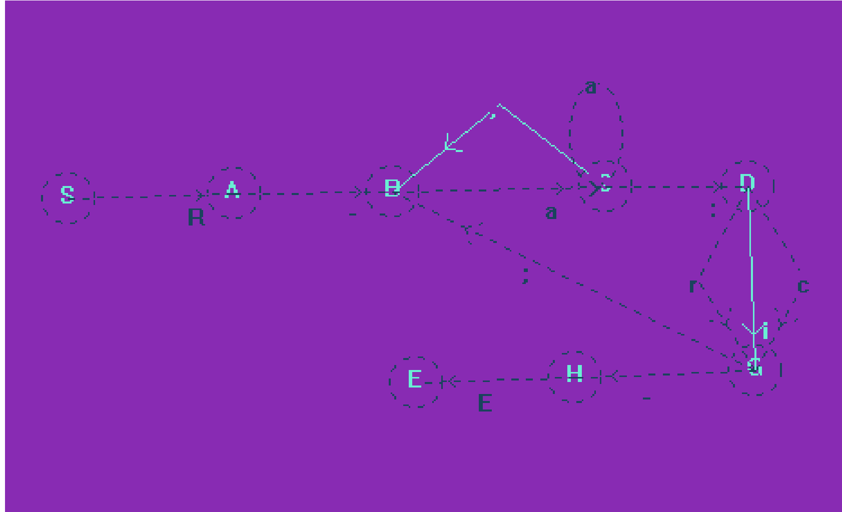
Das Zeichen _ steht für das Leerzeichen,R für Record,E für end,und als Bezeichner sollen nur Wörter,die aus dem Buchstaben a bestehen,zugelassen sein.

Als Typ steht r für Real,i für Integer und c für char.

(vgl. Lit 43, S.133)

Lösung:

Werden die Grundwörter R,_,; usw. als Kantenwerte aufgefaßt,die Linien ——— als Kanten und die Verzweigungsstellen der Linien als Knoten,so läßt sich ein äquivalenter endlicher Automat konstruieren,der die Syntax überprüft:



G040.gra

Zustandsfolge: S A B C C D G B C D G H E
Eingabefolge: R - a a : r ; a : c - E (erkanntes Wort)

Auf die Einführung eines Fehlerzustandes wurde verzichtet,damit der Graph nicht zu unübersichtlich wird,da sonst noch zusätzlich von jedem Knoten aus noch Kanten zu einem Fehlerzustand F erzeugt werden müßten,die jeweils alle (zulässigen) Zeichen enthalten,die nicht als Zeichen in den vom jeweiligen Knoten ausgehenden Kanten enthalten sind.
Stattdessen ist der Fehlerzustand immer dann erreicht,wenn durch das Programm „unzulässige Eingabe" angezeigt wird.

Andere Anwendungsaufgaben zum Thema endlicher Automat sind z.B. das Erkennen von Mustern (Pattern-Matching-Probleme, vgl.Lit 47, S.70 ff.),die mit dem in diesem Kapitel besprochenen Algorithmus ebenfalls leicht simuliert werden können.

C VIII Graphen als Relationen

Der Begriff der Funktion steht im herkömmlichen Mathematikunterricht im Hinblick auf den Analysisunterricht im Vordergrund,der allgemeinere Begriff der Relation wird dagegen seltener benutzt und dient oft nur dazu,den Begriff der Funktion von ihm abzuheben und zu spezialisieren.Dieses liegt sicher zum Teil auch darin begründet,dass sich Funktionen (meistens) graphisch übersichtlich in einem Koordinatensystem zeichnen lassen, während die Darstellung einer Relation dort wesentlich unübersichtlicher erscheint.

Bedeutend besser geeignet ist die zeicherische Veranschaulichung einer Relation als Graph (aus Knoten und Kanten).Dabei werden zwei Elemente,die in Relation zueinander stehen,durch zwei Knoten symbolisiert,die durch eine Kante verbunden werden.Wichtige Begriffe der Strukturmathematik wie Reflexivität,Transitivität,Symmetrie und Antisymmetrie sowie der Begriff der Äquivalenzrelation lassen sich mit Hilfe der Darstellung der Relation als Graphen gut veranschaulichen.

Auch der Begriff der transitiven bzw. reflexiven Hülle läßt sich durch die Vervollständigung der Kantenmengen entsprechender Graphen leicht verständlich machen.Oft impliziert eine Relation eine Rangfolge zwischen einzelnen Elementenpaaren (lokale Ordnung).Es stellt sich die Frage,wann und wie sich aus dieser lokalen Ordnung eine globale Ordnung,d.h. eine Rangfolge zwischen allen vorgegebenen Elementen herstellen läßt.

Das Erzeugen einer transitiven Hülle beruht auf dem Algorithmus von Warshall,der hier an die objektorientierte Datenstruktur angepasst wurde (und sonst auf einer Darstellung des Graphen als Matrix aufbaut),und das Erzeugen einer globalen Ordnungsrelation leitet sich von dem TiefenBaumDurchlauf ab,so dass sich eine entsprechende Unterrichtsreihe zum Thema Bäume und Pfade mit diesem Thema fortsetzen läßt.Mit Hilfe der im folgenden beschriebenen Algorithmen lassen sich jeweils die oben genannten Relationseigenschaften eines vorgegebenen Graphen leicht bestimmen.

So eigenen sich das Programmsystem Knotengraph dazu,um dieses wichtige (aber ansonsten trockene) Kapitel der Strukturmathematik einer anschaulichen,experimentellen Behandlung im Schulunterricht zugänglich zu machen. Wiederum läßt sich ein Unterrichtsgang konzipieren,der den fertig codierten Algorithmus des Programm Knotengraphs im Menü Anwendungen/Relation als Graph lediglich als Werkzeug zur Veranschaulichung entsprechender Graphenprobleme (im Mathematikunterricht) benutzt.Oder aber die Algorithmen werden (z.B. im Informatikunterricht) an Hand des Demomodus des Programm Knotengraphs zunächst nachvollzogen und dann selber neu codiert.

In beiden Fällen ist es erforderlich mit einem konkreten Einführungsbeispiel zu beginnen,das je nach Unterrichtskonzeption entweder sofort manuell und mit dem Programm Knotengraph oder aber noch zusätzlich später mittels der eigenen Codierung gelöst wird.

Aufgabe C VIII.1 (1.Teil Einstiegsproblem):

In einer Firma sind B und F die Vorgesetzten von A,E der Vorgesetzte von B,B und F die Vorgesetzten von D,D der Vorgesetzte von C,G der Vorgesetzte von F und E der Vorgesetzte von G.

Ein Vorgesetzter ist jeweils weisungsbefugt zu allen seinen „Untergebenen“.

Die Relation „ist Vorgesetzter“ läßt sich als Graph darstellen.Dabei stellen die Knoten die Personen A,B,C,D,E,F und G dar.Eine gerichtete Kante von X nach Y besagt,dass Y der Vorgesetzte von X ist.Die Relation Y ist weisungsbefugt zu X soll in entsprechender Weise durch eine gerichtete Kante von X nach Y dargestellt werden.

a)Erstelle mittels des Programms Knotengraph (oder auch zunächst manuell) einen Graphen,der die Relation „ist Vorgesetzter von“ darstellt.

b)Ermittle mit Hilfe des Algorithmus des Menüs Anwendungen/Relation als Graph des Programmm Knotengraph (oder manuell),welche Personen jeweils welchen anderen Personen Anweisungen geben dürfen sowie die Rangfolge der Personen.(Eine Person steht in der Rangfolge weiter oben als eine andere Person,wenn sie dieser Person Anweisungen erteilen darf.)

D.h. erstelle aus dem obigen Graphen einen Graphen,der die Relation ist „weisungsbefugt zu“ darstellt.

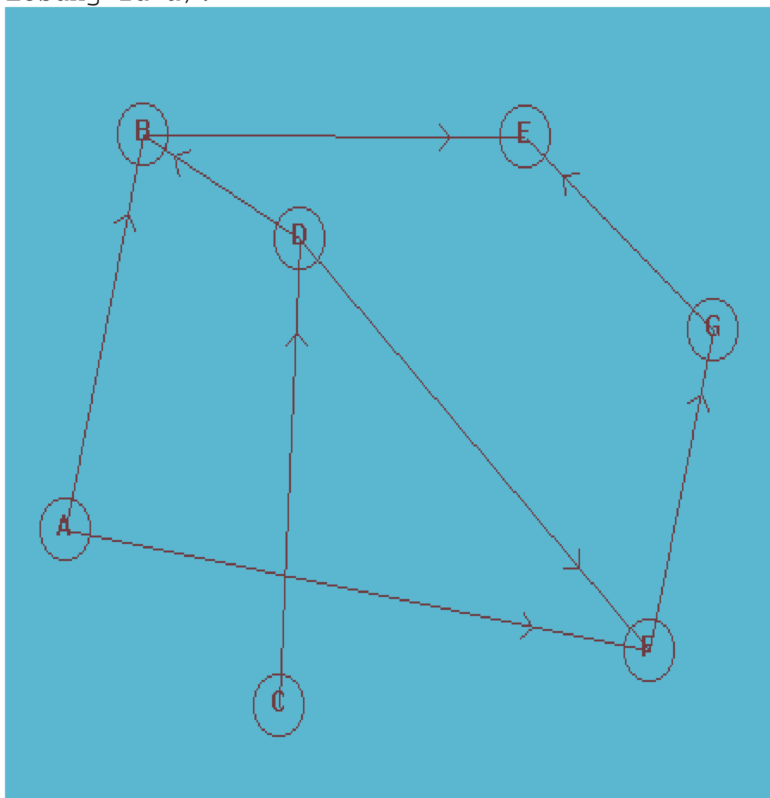
Berücksichtige bei der Erstellung des Graphen noch,dass jede Person sich selber Anweisungen erteilen kann.

c)Wähle beim dem Relations-Algorithmus des Programms Knotengraph die Option,dass der ursprüngliche Graph nicht wiederhergestellt werden soll,lasse den Algorithmus erneu (d.h. zweimal) ablaufen,und ermittle die Ausgabe im Ausgabefenster.

d)In der oben genannten Firma ist die Demokratie ausgebrochen,so dass jede Person jeder anderen Anweisungen geben darf (eventuell Chaos!).Erstelle auch für diesen Fall einen Graphen,und ermittle die Ausgabe im Ausgabefenster mit Hilfe des Programms Knotengraph auch für diesen Fall.Alternativ:Suche manuell nach einer Lösung.

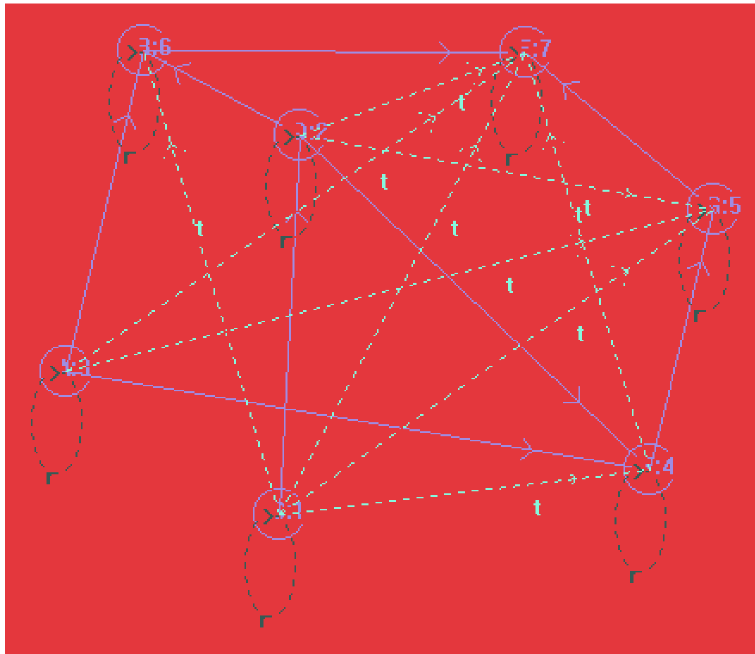
Lösung:

Lösung zu a):



G041.gra

Lösung zu b):



G041.gra

Erzeuge dazu die reflexive und transitive Hülle des Graphen.
Die grün markierten Schlingen besagen, dass jede Person zu sich selber weisungsbefugt ist.

Wenn B weisungsbefugt zu A ist und E weisungsbefugt zu B ist, dann ist auch E weisungsbefugt zu A. Deshalb muß die Kante AE zusätzlich eingezeichnet werden. Dasselbe gilt für die übrigen rot markierten Kanten.

Die Rangfolge wird im Ausgabefenster unter Ordnung angezeigt:

Ordnung:
A:3
B:6
C:1
D:2
E:7
F:4
G:5

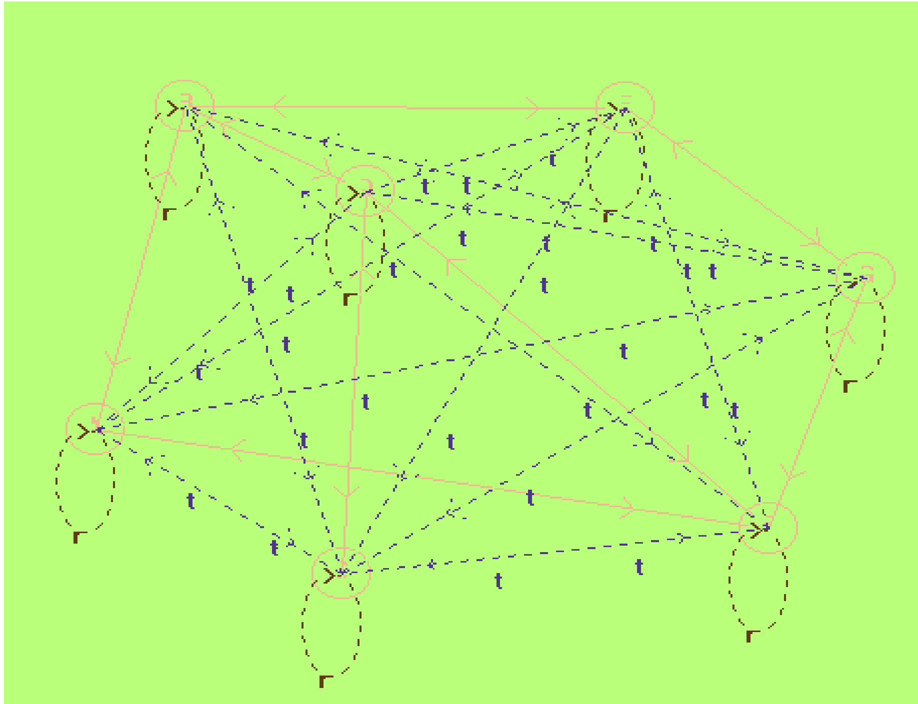
Die reflexive und transitive Hülle der Relation ist antisymmetrisch!
Die ursprüngliche Relation ist nicht reflexiv!
Die ursprüngliche Relation ist nicht transitiv!
Die ursprüngliche Relation ist nicht symmetrisch!

c) (Graph wie bei b))

Ordnung:
A:3
B:6
C:1
D:2
E:7
F:4
G:5

Die reflexive und transitive Hülle der Relation ist antisymmetrisch!
Die ursprüngliche Relation ist reflexiv!
Die ursprüngliche Relation ist transitiv!
Die ursprüngliche Relation ist nicht symmetrisch!

d)



G042.gra

Die reflexive und transitive Hülle der Relation ist nicht antisymmetrisch!
Die ursprüngliche Relation ist nicht reflexiv!
Die ursprüngliche Relation ist nicht transitiv!
Die ursprüngliche Relation ist symmetrisch!

Das Enstiegsproblem macht folgende Definitionen plausibel:

Definition C VIII.1:

Unter einer Relation R über einer Menge M versteht man eine Teilmenge R von MxM.

Eine Relation R heißt reflexiv,wenn für alle $x \in M$ $(x,x) \in R$ gilt.

Eine Relation R heißt symmetrisch,wenn für für alle $(x,y) \in R$ $(y,x) \in R$ gilt mit $x,y \in M$.

Eine Relation heißt transitiv,wenn für alle $(x,y) \in R$ und $(y,z) \in R$ gilt $(x,z) \in R$ mit $x,y,z \in M$.

Eine Relation heißt antisymmetrisch,wenn für alle $(x,y) \in R$ und $(y,x) \in R$ gilt $x=y$ mit $x,y \in M$.

Eine Relation R heißt Ordnungsrelation,wenn R antisymmetrisch und transitiv ist.

Eine Relation R heißt Äquivalenzrelation,wenn R reflexiv,symmetrisch und transitiv ist.

Wenn man zu einer Relation R alle die Paare (x,x),für die gilt $(x,x) \notin R$ zusätzlich hinzufügt,entsteht die reflexive Hülle der Relation.

Wenn man zu einer Relation für alle $(x,y) \in R$ und $(y,z) \in R$ mit $(x,z) \notin R$ die Paare (x,z) solange hinzufügt,bis sich keine neuen Paare mehr hinzufügen lassen,entsteht die transitive Hülle von R.

Bemerkungen:

1)Wenn (x,y) Element der transitiven und reflexiven Hülle von R ist,dann gibt es einen Kantenzug in dem die Relation darstellenden Graphen zwischen den Knoten x und y.

2)Wenn ein Graph eine Ordnungsrelation ist,dann lassen sich die Elemente der Menge M mittels einer Größergleichrelation >= so der Größe nach anordnen,dass mit (x,y)∈R x<=y gilt.Das Gleichheitszeichen gilt genau dann,wenn x=y ist.

3)Die reflexive und transitive Hülle einer Relation ist genau dann eine Ordnungsrelation,wenn der zugehörige schlichte Graph keine Kreise (auch nicht der Länge 2,d.h. keine antiparallelen Kanten) enthält.

Überprüfe die Bemerkungen an Hand geeigneter Graphen mit Hilfe des Programms Knotengraph,Menüs Pfade/Minimaler Pfad zwischen zwei Knoten, Eigenschaften/Kreise,Eigenschaften Parallelkanten und Schlingen und Anwendungen/Graph als Relation.

Wie lassen sich nun reflexive Hülle,transitive Hülle und eine Ordnung zu einer Ordnungsrelation zu einer als Graph dargestellten Relation algorithmisch ermitteln?Wie lassen sich Symmetrie,Transitivität und Reflexivität eines Graphen überprüfen?

Aufgabe C VIII.1 (2.Teil Einstiegsproblem):

Ermittle an Hand der der Lösung zu Aufgabe b) auch unter Einsatz des Demomodus oder Einzelschrittmodus den Ablauf des Algorithmus des Menüs Anwendungen/Graph als Relation.Benutze auch Graph G004a.gra.

Beobachtung und problemorientierte Erarbeitung:

An Hand der grünen und roten Zusatzkanten kann sofort (ohne Demo-oder Einzelschrittmodus) erkannt werden,dass zur Erzeugung der reflexiven Hülle zu jedem Knoten des Graphen ohne Schlinge eine Schlinge hinzugefügt wird.Um die transitive Hülle zu erzeugen,ist jeweils,falls nicht vorhanden,bei jedem „Zweier-Pfad" von Kanten eine rote Zusatzkante vom Startknoten der ersten Kante zum Endknoten der zweiten Kante hinzuzufügen.

Das Erzeugen der globalen Ordnung sollte dann mittels des Demomodus oder Einzelschrittmodus beobachtet werden.Wendet man den Algorithmus zunächst auf den Graphen G004a.ga der Aufgabe C II 1a an,zeigt sich dass der bekannte Tiefebaumdurchlauf in Postorder-Reihenfolge abläuft.

Zu beobachten ist dann bei Graph G041.gra allerdings,dass dort nacheinander zweimal ein Postorder-Tiefenbaumdurchlauf stattfindet,nämlich mit den Startknoten A und C.Von A aus erfolgt die Nummerierung der Knoten E 7,B 6,G 5,F 4 und A 3.Von C aus erfolgt die Nummerierung D 2 und C 1.Also erzeugt der Algorithmus jeweils von allen (bis dahin) noch nicht besuchten Knoten aus einen tiefen Baumdurchlauf in Postorderfolge,wobei die am weitesten enfernten Knoten die höchsten noch verfügbaren Ordnungszahlen bekommen.

Das Verfahren wird dann zu folgendem Algorithmus präzisiert:

Algorithmus Graph als Relation

Verbale Beschreibung:

Reflexivität:

Das Erzeugen der reflexiven Hülle geschieht dadurch,dass jeder Knoten auf die Existenz von Schlingen untersucht wird.Wenn keine Schlinge zu einem Knoten existiert,wird zu diesem Knoten eine zusätzliche Schlinge erzeugt

und dem Graph hinzugefügt.Dann war die ursprüngliche Relation nicht reflexiv.

Transitivität:

Das Erzeugen der transitiven Hülle geschieht nach dem Algorithmus von Warshall:

Führe für jeden Knoten Kno des Graphen die folgende Operation durch:

Für alle Anfangsknoten Knoa der einlaufenden Kanten zum Knoten Kno führe jeweils folgende Operation durch:

Für alle Endknoten Knoe der auslaufenden Kanten zum Knoten Kno führe jeweils folgende Operation durch:

Wenn es noch keine Kanten zwischen Knoa und Knoe im Graphen gibt,erzeuge diese Kante und füge sie in den Graphen ein.

Wenn eine Kante zusätzlich eingefügt wurde,ist die ursprüngliche Relation nicht transitiv.

Symmetrie:

Zu jeder Kante des Graphen (d.h. der Kantenliste des Graphen) wird überprüft,ob eine Kante vom Endknoten zum Anfangsknoten dieser Kante existiert.Wenn dies für alle Kanten des Graphen zutrifft,ist die Relation symmetrisch,sonst nicht.

Topologisches Sortieren/Ordnung erzeugen:

Das topologische Sortieren beruht auf der Tiefensuche (vgl. Paragraph C II):

Jeder Knoten erhält ein zusätzliches Feld vom Typ Integer,in dem die Knotenordnung gespeichert wird.

1)Bei jedem Knoten wird die Knotenordnung auf Null gesetzt.Die Besucht-Markierung jedes Knoten wird gelöscht.Eine Boolesche Variable Kreis wird auf false gesetzt.Setze eine globale Integer-Variable AktuelleOrdnung auf Null.

2)Für alle noch nicht markierten Knoten Kno der Knotenliste des Graphen rufe mit diesem Knoten als Parameter die rekursive Methode Topsort auf:

Rekursive Methode Topsort:

3)Prüfe,ob die Boolesche Variable Kreis true ist.

Wenn ja,verlasse alle Rekursionsebenen und beende den Algorithmus.Dann ist die Relation nicht antisymmetrisch,und es existiert ein Kreiskantenzug im Graphen.(Abbruchbedingung)

4)Wenn nicht markiere den aktuellen Knoten Kno als besucht.Wähle den Zielknoten K der in der Kantenreihenfolge nächsten ausgehenden Kante vom Knoten Kno aus (bzw. beim ersten Aufruf den Zielknoten der ersten Kante).

5)Wenn die in der Reihenfolge letzte Kante schon ausgewählt wurde,verlasse diese Rekursionsebene von Topsort (Abbruchbedingung),und setze den Ablauf in der vorigen Rekursionsebene bei Schritt 8) fort.

5)Überprüfe,ob K schon als besucht markiert wurde.

6)Wenn K noch nicht als besucht markiert wurde,dann rufe die Methode Topsort mit dem Parameter K (für Kno) rekursiv erneut auf (Fortsetzung in der nächsten Rekursionsebene bei Schritt 3)).

7)Wenn K schon als besucht markiert wurde,dann überprüfe,ob die Knotenordnung dieses Knotens gleich Null ist.Wenn dies der Fall ist,setze die Variable Kreis auf true.(Dann existiert ein geschlossener Kantenzug im Graph.)

8)Erhöhe die Variable aktuelleOrdnung um Eins.Speichere die Differenz Knotenanzahl des Graphen minus aktuelle Ordnung vergrößert um 1 als Knotenordnung im entsprechenden Feld des Knotens Kno.

8)Setze den Ablauf bei Schritt 3) fort.

Natürlich kann man sich sowohl bei der verbalen Beschreibung als auch im folgenden Quellcode nur auf die Besprechung bzw. Programmierung von einem bzw. auf eine Teilmenge der beschriebenen (unabhängigen) Algorithmen beschränken.

Die Datenstruktur:

TRelationknoten leitet sich durch Vererbung von TInhaltsknoten ab und enthält die zusätzlichen Felder Ordnung_ (Integer) und Ergebnis_(string) sowie die entsprechenden Propertys.Ordnung_ dient zum Speichern der Ordnung des Knotens.Im Feld Ergebnis_ wird das Ergebnis bestehend aus Knoteninhalt und Knotenordnung (als string) zwecks Anzeige gespeichert.Auf die Knotenordnung kann durch die Property Ordnung und auf das Ergebnis mittels der Property Ergebnis (bzw. mittels der entsprechenden Methoden) zugegriffen werden.

```
TRelationsknoten = class(TInhaltsknoten)
  private
    Ordnung_:Integer;
    Ergebnis_:string;
    procedure SetzeOrdnung(O:Integer);
    function WelcheOrdnung:Integer;
    procedure SetzeErgebnis(S:string);
    function WelchesErgebnis:string;
  public
    constructor Create;
    property Ordnung:Integer read WelcheOrdnung write setzeOrdnung;
    property Ergebnis:string read WelchesErgebnis write SetzeErgebnis;
    function Wertlisteschreiben:TStringlist;override;
    procedure Wertlistelesen;override;
  end;
```

TRelationsgraph leitet sich von TInhaltsgraph durch Vererbung ab,und enthält keine neuen Datenfelder.Die Methoden von TRelationsgraph werden im folgenden beschrieben.

```
TRelationsgraph = class(TInhaltsgraph)
    constructor Create;
    procedure SetzebeiallenKnotenAnfangsOrdnung;
    procedure Schlingenerzeugen(var SListe:TStringlist);
    procedure ErzeugeOrdnung(Flaeche:TCanvas);
    procedure Warshall(var SListe:TStringlist);
    procedure ErzeugeErgebnis(var SListe:TStringlist);
    function Relationistsymmetrisch:Boolean;
  end;
```

Der Quellcode:

```
constructor TRelationsknoten.Create;
begin
  inherited Create;
  Inhaltsknotenclass:=TRelationsknoten;
  Inhaltskanteclass:=TInhaltskante;
end;
```

Der Constructor setzt die Datentypen für die Knoten des Graphen auf TRelationsknoten und für die Kanten des Graphen auf TInhaltskante.

```
procedure TRelationsgraph.SetzebeiallenKnotenAnfangsOrdnung;
var Index:Integer;
```

```
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      TRelationsknoten(Knotenliste.Knoten(Index)).Ordnung:=0;
end;
```

Die Methode SetzebeiallenKnotenAnfangsOrdnung setzt bei allen Knoten der Knotenliste des Graphen die Knotenordnung auf Null.

```
procedure TRelationsgraph.Schlingenerzeugen(var SListe:TStringlist);
var Index:Integer;
    Kno:TRelationsKnoten;
    Ka:TInhaltskante;
    Graphistreflexiv:Boolean;
begin
  Graphistreflexiv:=true;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TRelationsknoten(Knotenliste.Knoten(Index));
      if Kno.AnzahlSchlingen=0 then                                         1)
      begin
        Ka:=TInhaltskante.Create;
        Ka.Weite:=30;
        Ka.Wert:='r';
        Ka.Farbe:=clgreen;
        Ka.Stil:=psdot;
        FuegeKanteein(Kno,Kno,true,Ka);                                     2)
        Graphistreflexiv:=false;                                            3)
      end;
    end;
  if Graphistreflexiv
  then
    SListe.Add('Die ursprüngliche Relation ist reflexiv!')                 4)
  else
    SListe.Add('Die ursprüngliche Relation ist nicht reflexiv!');
end;
```

Wenn die Anzahl der Schlingen eines Knotens gleich Null ist (1),wird bei 2) eine neue grün-markierte Schlinge mit dem Inhalt r (reflexiv) erzeugt und in den Graph eingefügt.Wenn dies der Fall,wird die Variable Graphistreflexiv auf false gesetzt (3),und bei (4) wird eine entsprechende Aussage der Stringliste Sliste zur Anzeige im Ausgabefenster hinzugefügt (4).

```
procedure TRelationsgraph.ErzeugeOrdnung(Flaeche:TCanvas);
var AktuelleOrdnung:Integer;
    Kreis:Boolean;
    Index:Integer;

  procedure Topsort(Kno:TRelationsknoten);
  var Zaehl:Integer;

    procedure TopsortNachbarknoten(Kno:TKnoten;Ka:TKante);
    var K:TRelationsknoten;
    begin
      K:=TRelationsknoten(Ka.Zielknoten(Kno));                            12)
      TInhaltsknoten(K).Farbe:=clblack;
      TInhaltsknoten(K).Zeichneknoten(Flaeche);
      if not K.Besucht
      then                                                                 13)
        Topsort(K);                                                        14)
      else
        if K.Ordnung=0 then  Kreis:=true;                                  15)
    end;

  begin
    if not Kreis and (not Kno.Besucht) then                                7)
    begin
      Kno.Besucht:=true;                                                    8)
      if not Kno.AusgehendeKantenliste.Leer then
        for  Zaehl:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do

          if not Kno.AusgehendeKantenliste.Kante(Zaehl).KanteistSchlinge
          then
            TopsortNachbarknoten(Kno,Kno.ausgehendeKantenliste.Kante(Zaehl));  9)

      AktuelleOrdnung:=AktuelleOrdnung+1;                                  10)
      Kno.Ordnung:=Self.AnzahlKnoten-AktuelleOrdnung+1;                    11)
      end;
  end;
```

```
begin
  if not Leer then
  begin
    LoescheKnotenbesucht;                                                  5)
    SetzebeiallenKnotenAnfangsOrdnung;
    AktuelleOrdnung:=0;
    Kreis:=false;
    for Index:=0 to Knotenliste.Anzahl-1 do
      Topsort(TRelationsknoten(Knotenliste.Knoten(Index)));                6)
  end;
end;
```

Bei 5) wird die Besucht-Markierung der Knoten gelöscht,die Ordnung der Knoten auf Null gesetzt,die Boolesche Variable Kreis auf false gesetzt,und die globale Variable AktuelleOrdnung auf Null gesetzt.Bei 6) wird dann für jeden Knoten der Knotenliste als Parameter nacheinander die rekursive Methode Topsort aufgerufen.

Diese Methode wird ausgeführt,wenn der Knoten noch nicht als besucht markiert wurde und die Variable Kreis false ist (7).Dann wird bei 8) der aktuelle Knoten als besucht markiert und bei 9) nacheinander mit allen ausgehenden Kanten dieses Knotens die rekursive Methode TopsortNachbarknoten aufgerufen.
Bei 10) wird die globale Variable AktuelleOrdnung um 1 erhöht,und bei 11) wird die Differenz aus Knotenzahl des Graphen minus AktuelleOrdnung vergrößert um eins als Ordnung des aktuellen Knoten gespeichert.

Bei 12) wird der Zielknoten,der in die Methode TopsortNachbarknoten übergebenden Kante bestimmt,und bei 13) wird geprüft,ob dieser Zielknoten schon als besucht markiert wurde.Wenn nein,wird die Methode Topsort bei 14) rekursiv mit diesem Zielknoten erneut als aktuellem Knoten aufgerufen.Ansonsten wird bei 15),falls die Ordnung des Knotens Null ist,die Boolesche Variable Kreis auf false gesetzt.

```
procedure TRelationsgraph.Warshall(var SListe:TStringlist);
var Index,Index1,Index2:Integer;
    Graphisttransitiv:Boolean;
    Kno,Kno1,Kno2:TKnoten;
*
    procedure ErzeugeTransitiveKante(Kno1,Kno2:TKnoten);
    var Ka:TInhaltskante;
    begin
      if not KanteverbindetKnotenvonnach(Kno1,Kno2) then                   20)
      begin
        Ka:=Tinhaltskante.Create;
        Ka.Wert:='t';
        Ka.Farbe:=clred;
        Ka.Stil:=psdot;
        FuegeKanteein(TInhaltsknoten(Kno1),TInhaltsknoten(Kno2),true,Ka);  21)
        Graphisttransitiv:=false;                                          22)
      end;
    end;

begin
  Graphisttransitiv:=true;
  if not self.Knotenliste.Leer then
    for Index:=0 to self.Knotenliste.Anzahl-1 do                           16)
    begin
      Kno:=Self.Knotenliste.Knoten(Index);
      if not Kno.EingehendeKantenliste.leer then
 *       for Index1:=0 to Kno.EingehendeKantenliste.Anzahl-1 do             17)
        begin
          Kno1:=Kno.EingehendeKantenliste.Kante(index1).Anfangsknoten;
          if not Kno.AusgehendeKantenliste.leer then
            for Index2:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do         18)
            begin
              Kno2:=Kno.AusgehendeKantenliste.Kante(Index2).Endknoten;
              ErzeugeTransitiveKante(Kno1,Kno2);
            end;
        end;
    end;
  if not Graphisttransitiv                                                 23)
  then
    SListe.Add('Die ursprüngliche Relation ist nicht transitiv!')
  else
    SListe.Add('Die ursprüngliche Relation ist transitiv!');
end;
```

Bei 16) werden alle Knoten Kno der Knotenliste nacheinander ausgewählt.Bei 17) werden nacheinander alle Anfangsknoten Kno1 der eingehenden Kanten des Knotens Kno ausgewählt,und bei 18) alle Endknoten Kno2 der ausgehenden Kanten des Knotens Kno.

Mit den Knoten Kno1 und Kno2 wird dann bei 19) die Methode ErzeugetransitiveKante aufgerufen.Wenn es noch keine Kante vom Knoten Kno1 zum Knoten Kno2 gibt (20),wird eine solche Kante bei 21) erzeugt und in den Graph eingefügt (mit dem Inhalt t für transitiv).

Bei 22) wird in diesem Fall die Variable Graphisttransitiv auf false gesetzt.

Bei 23) wird der Wert der Variablen Graphisttransitiv geprüft und eine entsprechende Meldung ausgegeben,sowie die Meldung der Stringliste SListe zur Anzeige im Ausgabefenster hinzugefügt.

```
procedure TRelationsgraph.ErzeugeErgebnis(var SListe:TStringlist);
va*r Index:Integer;
    Kno:TRelationsknoten;
begin
  SListe.Add('Ordnung:');
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TRelationsknoten(self.Knotenliste.Knoten(Index));
      Kno.Ergebnis:=Kno.Wert+':'+Integertostring(Kno.Ordnung);                    24)
      SListe.Add(Kno.Ergebnis);
    end;
end;
```

Die Methode ErzeugeErgebnis erzeugt bei 24) für alle Knoten im Feld Ergebnis als string eine Ergebnis,das aus dem Knoteninhalt und der Ordnungszahl des Knotens besteht.

```
function TRelationsgraph.Relationistsymmetrisch:Boolean;
var Index:Integer;
    Ka:TInhaltskante;
    Knoa,Knoe:TRelationsknoten;
    Symmetrisch:Boolean;
begin
  Symmetrisch:=true;
  if not Kantenliste.leer then
    for Index:=0 to Kantenliste.Anzahl-1 do                                       25)
    begin
      Ka:=TInhaltskante(Kantenliste.Kante(Index));
      Knoa:=TRelationsknoten(Ka.Anfangsknoten);                                   26)
      Knoe:=TRelationsknoten(Ka.Endknoten);                                       27)
      if not self.KanteverbindetKnotenvonnach(Knoe,Knoa)                          28)
      then
        Symmetrisch:=false;
    end;
  Relationistsymmetrisch:=Symmetrisch;                                            29)
end;
```

Die Methode Relationistsymmetrisch überprüft bei 25) alle Kanten der Kantenliste des Graphen,ob auch eine (Rückwärts-) Kante vom Endknoten der Kante zum Anfangsknoten existiert (26 und 27).

Falls das nicht der Fall ist,wird bei 28) die Boolesche Variable symmetrisch auf false gesetzt und bei 29) als Resultat der Function zurückgegeben.

Aufgabe C VIII.2:

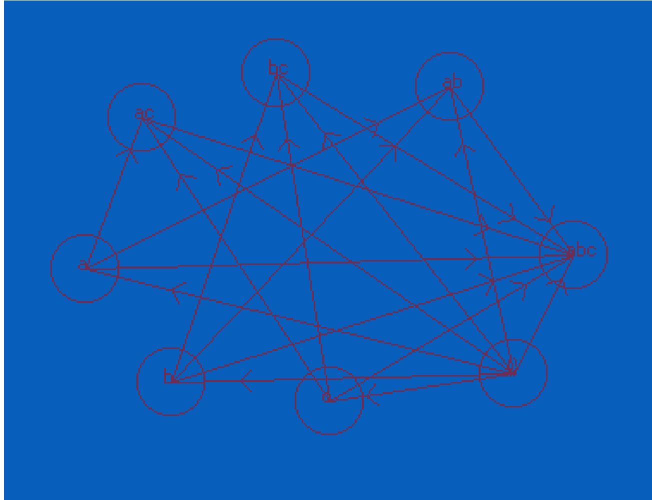
Gegeben ist die Menge M={a,b,c}.Bilde die Menge P aller Untermengen von M (d.h. die Potenzmenge).In P ist zwischen zwei Mengen die (echte) Teilmengenrelation definiert.Stelle diese Teilmengenrelation durch einen Graph mit Knotengraph dar.Die Knoten stellen dabei die Mengen von P dar.

Zwischen zwei Knoten wird genau dann eine gerichtete Kante gezeichnet,wenn die Menge des ersten Knoten eine echte Teilmenge der Menge des zweiten Knotens ist.

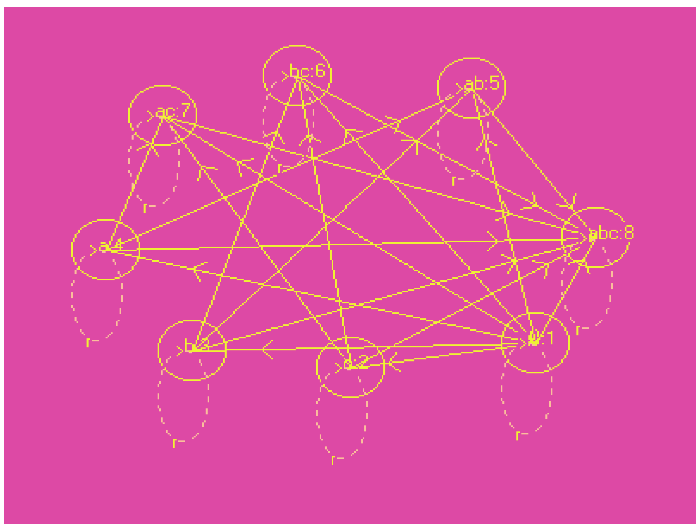
Untersuche die Eigenschaft der Relation durch des Algorithmus des Menüs Anwendungen/Graph als Relation des Programms Knotengraph (benutze dabei den Demo-Modus) oder mittels des eigenen Quellcodes in der Entwicklungsumgebung.

Was bedeutet das zusätzliche Einfügen von Schlingen in jedem Knoten?

Lösung:



G043.gra



G043.gra

Das Einfügen von Schlingen bedeutet,dass die unechte Teilmengenrelation betrachtet wird.

Die Knoten können der Reihe nach geordnet werden,so dass jeweils Kanten nur von Knoten kleinerer Ordnung zu Knoten größerer Ordnung führen:

Ordnung:

abc:8
ac: 7
bc: 6
ab: 5
a: 4
b: 3
c: 2
{}: 1

Die reflexive und transitive Hülle der Relation ist antisymmetrisch!
Die ursprüngliche Relation ist nicht reflexiv!
Die ursprüngliche Relation ist transitiv!
Die ursprüngliche Relation ist nicht symmetrisch!

<u>Aufgabe C VIII.3:</u>

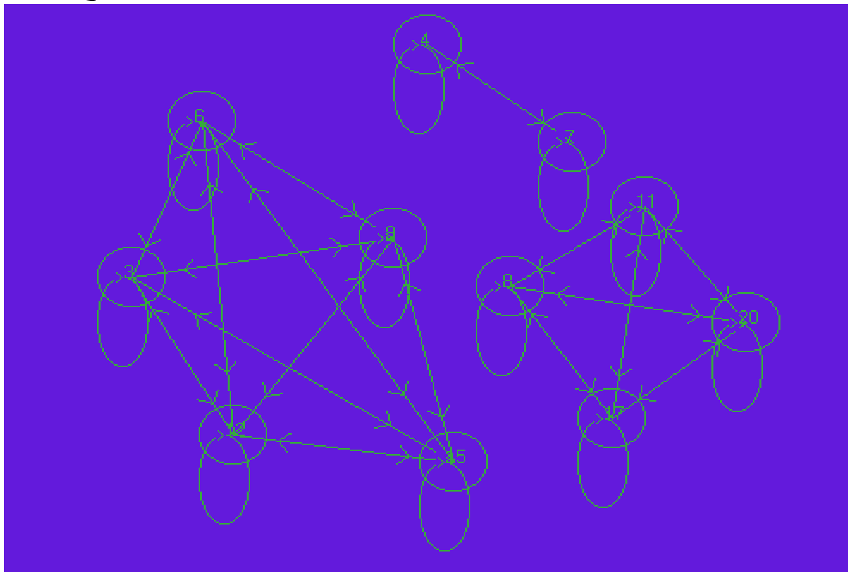
Gegeben ist die Relation R mit $(x,y)\in R \Leftrightarrow (x-y)$ Mod 3 = 0,wobei der Mod-Operator auch für negative Zahlen definiert sei,über der Menge M.

Die Menge M sei M={3;4;6;7;8;9;11;12;15;17;20}

Erzeuge mit Knotengraph einen Graph zu dieser Relation und ermittle die Eigenschaften mittels des Menüs Anwendungen/Graph als Relation oder mittels des eigenen Quellcodes in der Entwicklungsumgebung.

Bestimme die Anzahl der Komponenten des Graphen mittels des Algorithmus des Men*üs Eigenschaften/Anzahl Kanten und Knoten.

Lösung:



G044.gra

Die reflexive und transitive Hülle der Relation ist nicht antisymmetrisch!
Die ursprüngliche Relation ist reflexiv!
Die ursprüngliche Relation ist transitiv!
Die ursprüngliche Relation ist symmetrisch!

Also ist die Relation eine Äquivalenzrelation.

Das Menü Eigenschaften/Anzahl Kanten und Knoten zeigt,dass die Anzahl der Komponenten 3 ist.

Folgerung:

Eine Äquivalenzrelation besteht aus verschiedenen Partitionen.

Hier sind es die Mengen:

M_1={4;7} M_2={8;11;17;20} M_3={3;6;9;12;15}

Zwischen den Partitionen des Graphen bestehen keine Kanten.
Innerhalb der Partitionen (Komponenten) ist jeder Knoten mit jedem anderen Knoten durch eine (Vorwärts-und Rückwärts-)Kante verbunden. Außerdem besitzt jeder Knoten eine Schlinge.

C IX Maximaler Netzfluss

Das Problem des maximalen Netzflusses von einem Quellenknoten zu einem Senkenknoten in einem Netzgraphen ist ebenfalls ein Optimierungsproblem des Operations Research.Es gilt hier wieder das schon im Kapitel C VI Graph als Netzplan gesagte,dass es sich nämlich um ein Optimierungsproblem der diskreten endlichen Mathematik handelt,dessen Lösungsverfahren deshalb von grundsätzlicher didaktischer Bedeutung ist,weil der herkömmliche Mathematikunterricht fast nur Optimierungsprobleme kennt,deren Lösung mit Hilfe infiniter Verfahren der Analysis erfolgen.

Im Gegensatz zu der Aufgabe des Entwurfs eines Netzzeitplans,tritt bei diesem Problem die Berücksichtigung der Existenz von Nebenbedingungen (in Form der Kantenschranken,die z.B. als Rohrkapazitäten gedeutet werden können) auf.

Zur Lösung wird der Algorithmus von Ford-Fulkerson benutzt.Die Weiterführung der Lösungsmöglichkeiten dieses Algorithmus führt zum (einfachen) Transportproblem sowie zur Lösung des Maximalen Matching-Problems in paaren Graphen,das im nächsten Kapitel dargestellt wird und sich didaktisch-methodisch als geeignete Fortsetzung anbietet.Das einfache Transportproblem kann wiederum durch die Themen (allgemeines) Transportproblem,Hitchcockproblem und Optimales Matchingproblem erweitert werden,die im Kapitel C XIII erörtert werden (vgl. dazu auch die Aufgaben C IX.4 und C XIII.4).

Auch bei dem Thema maximaler Netzfluss ist sowohl eine Programmierung (eigene Quellcodeerstellung) des Algorithmus z.B. als Projekt im Informatikunterricht oder aber eine Behandlung des Algorithmus als Verbalalgorithmus (wie weiter unten dargestellt) im Mathematikunterricht unter Demonstration des Verfahrens mit Hilfe des Demomodus des Programms Knotengraph mittels des Algorithmus des Menüs Anwendungen/Maximaler Fluss möglich.

In beiden Fällen sollte von den folgenden Beispiel-oder Einstiegsproblemen Gebrauch gemacht werden.

Siehe zu diesem Thema auch den zugehörigen Unterrichtsplan im Anhang F IV.

Aufgabe C IX.1 (Einstiegsproblem):

Gegeben ist ein Wasser-Rohrleitungssystem.Das Wasser wird bei 1 in das Leitungssystem eingespeist und kann bei 7 wieder herausfließen.Es gibt nur eine Einspeisestelle (Quelle) und nur eine Ausflussstelle (Senke).Die Rohre haben je nach Dicke eine eingeschränkte Kapazität (z.B. Liter pro Minute), um Wasser transportieren zu können.Durch jede Rohrleitung kann das Wasser nur in eine Richtung fließen.

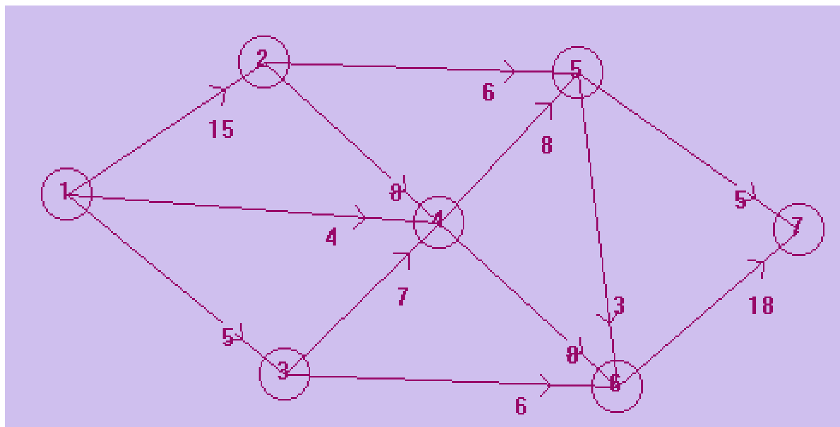
Das Problem kann als Graph dargestellt werden.Die Kanten stellen dabei jeweils die Wasserrohre dar.Der Kanteninhalt bedeutet jeweils die maximale Wasserkapazität (pro Zeiteinheit).

Die Knoten stellen die Rohrverzweigungen dar.Bei jedem Knoten ist die Wassermenge (pro Zeiteinheit),die in ihn hineinfließt gleich der Wassermenge,die aus ihm herausfließt.Die Knoten 1 und 7 sind der Quell-und Senkenknoten.

Wenn eine (Rück-)Rohrleitung von 7 nach 1 gezeichnet wäre,gilt die Gleichheit der Flüsse auch für diese beiden Knoten.

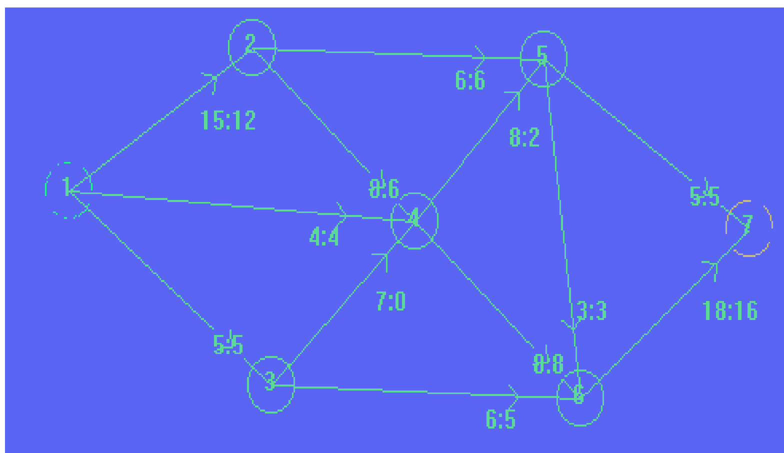
Wieviel Wasser kann maximal pro Minute von 1 nach 7 transportiert werden.Wieviel Liter pro Minute fließen dann in welchem Rohr?

Löse das Problem zunächst per Hand (durch Probieren) und danach mit dem Programm Knotengraph und dem Algorithmus des Menüs Anwendungen/Maximaler Netzfluss.Benutze auch den Demomodus.



G045.gra

Lösung:



G045.gra

Zu jeder Kante ist angegeben (zweite Zahl),wieviel Liter pro Minute durch sie fließen.Die maximale Kapazität wird nirgendswo überschritten.

Der maximale Fluss beträgt 21 Liter pro Minute.

Kanten,Schranken und Fluss:
1->3 Schranke: 5 Fluss: 5
1->4 Schranke: 4 Fluss: 4
1->2 Schranke: 15 Fluss: 12
2->5 Schranke: 6 Fluss: 6
2->4 Schranke: 8 Fluss: 6
3->4 Schranke: 7 Fluss: 0
3->6 Schranke: 6 Fluss: 5
4->5 Schranke: 8 Fluss: 2
4->6 Schranke: 8 Fluss: 8
5->7 Schranke: 5 Fluss: 5
5->6 Schranke: 3 Fluss: 3
6->7 Schranke: 18 Fluss: 16
maximaler GesamtFluss: 21

Definition C IX.1:

Unter einem Fluss auf einem zusammenhängenden,gerichteten,schlichten Graphen ohne Schlingen versteht man eine nichtnegative Zahl,die jeder Kante zugeordnet wird.Die Summe der Flüsse aller einlaufenden Kanten eines Knoten muß stets gleich der Summe der Flüsse aller auslaufenden Kanten sein außer beim Quellen-und Senkenknoten,die jeweils nur auslaufende bzw. einlaufende Kanten haben.Im Flussgraphen muß es jeweils genau einen Quellen- und einen

Senkenknoten geben.Wenn man sich zusätzlich eine Kante vom Senken-zum Quellenknoten eingefügt denkt,die jeweils die Flusssumme der ein-bzw. auslaufenden Kanten trägt,gilt die zuvor genannte Bedingung auch für diese beiden Knoten.Der Fluss durch diese Kante ist der Gesamtfluss.

Wenn den Kanten des Graphen noch eine weitere nichtnegative Zahl als Schranke zugeordnet wird,muß der Fluss durch jede Kante noch zusätzlich der Bedingung genügen,dass der Fluss kleiner oder gleich dieser Schranke ist.(Nebenbedingung)

Die Bestimmung des maximalen Flusses kann durch den Algorithmus von Ford-Fulkerson erfolgen:

Eine Möglichkeit der problemorientierten Erarbeitung des Algorithmus wird im zugehörigen Unterichtsplan im Anhang F IV ausführlich dargestellt.

Der Algorithmus von Ford-Fulkerson

Vorgegeben sei ein zusammenhängender,schlichter Graph ohne Schlingen mit einem Quellen- und einem Senkenknoten,die jeweils nur auslaufende bzw. nur einlaufende Kanten haben.

Allen Kanten des Graphen werden eine nichtnegative Zahl,genannt Fluss,die zu Beginn des Algorithmus auf Null gesetzt wird als auch eine zweite nichtnegative Zahl als Schranke des Flusses zugeordnet.

Unter einem Flusspfad vom Quellen- zum Zielknoten sei ein Pfad mit folgenden Eigenschaften verstanden:

Der Pfad kann sowohl Vorwärtskanten (die in Pfeilrichtung durchlaufen werden) als auch Rückwärtskanten,die entgegen der Pfeilrichtung durchlaufen werden) enthalten.

Unter der Distanz versteht man bei Vorwärtskanten die Differenz zwischen Schranke und Fluss,die immer größer oder gleich Null sein muß,da der Fluss nicht größer als die Schranke werden darf.Bei Rückwärtskanten ist die Distanz der (nichtnegative) Fluss der Kante selber.

Jedem Flusspfad vom Quellen- zum Zielknoten wird als Pfaddistanz die kleinste der Distanzen der einzelnen Kanten zugeordnet.Ein Flusspfad heißt zulässig,wenn die Pfaddistanz größer als Null ist.

Verbale Beschreibung:

Der Algorithmus von Ford-Fulkerson läuft dann folgendermaßen ab:

1)Setze den Fluss in allen Kanten gleich Null.Die Kanteninhalte werden als Schranken aufgefaßt.

2)Setze eine Variable Gesamtfluss auf Null.

3)Bestimme einen möglichen zulässigen Flusspfad zwischen Quellen- und Senkenknoten,falls ein solcher Pfad existiert.

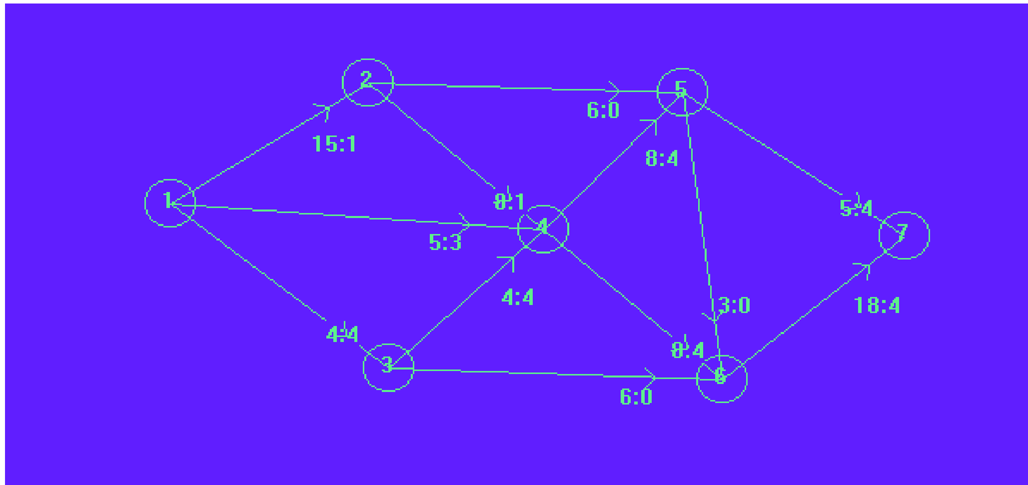
4)Wenn kein solcher zulässiger Pfad existiert,beende den Algorithmus.Der Wert der Variable Gesamtfluss ist dann der maximale Gesamtfluss.

5)Wenn ein zulässiger Pfad existiert,erhöhe den Fluss bei allen Vorwärtskanten auf diesem Flusspfad um die Pfaddistanz und vermindere den Fluss bei allen Rückwärtskanten auf diesem Pfad um die Pfaddistanz.

6)Erhöhe die Variable Gesamtfluss um die Pfaddistanz.

7)Setze den Algorithmus bei Schritt 3) fort.

Schritt 5) des Algorithmus wird durch den folgenden Graphen veranschaulicht:



G046.gra

Der Quellenknoten ist Knoten 1,der Senkenknoten Knoten 7.Die erste Zahl der Kanten des Graphen bedeutet die Kantenschranke und die zweite Zahl den Kantenfluss.Die Bedingung für einen Fluss,dass die Summe der Flüsse der einlaufenden Kanten eines jeden Knotens gleich der Summe der Flüsse der auslaufenden Kanten ist,ist erfüllt.Der Gesamtfluss beträgt momentan 8.

Der Gesamtfluss kann z.B. durch folgende zulässigen Flusspfade erhöht werden:

1)Pfad längs der Knoten 1 2 5 7.Dieser Pfad enthält nur Vorwärtskanten.Die Distanzen der Kanten sind 14,6 und 1.Die Pfaddistanz ist also 1.Also kann der Fluss längs des Pfades um 1 erhöht werden.Der Fluss durch die Kanten ist nach der Erhöhung: 2,1 und 5.Der Gesamtfluss ist dann 9.

2)Pfad längs der Knoten 1 4 3 6 7.Dieser Pfad enthält eine Rückwärtskante zwischen 4 und 3.Die Distanzen der Kanten sind 2,4,6 und 14.Also ist die Pfaddistanz 2.Bei der Kante zwischen 4 und 3 wird der Fluss um 2 verringert,bei den anderen Kanten um 2 erhöht.Der Fluss ist nach der Änderung:5,2,2 und 6.Mit der Erhöhung von 1) ist der Gesamtfluss dann 11.

Um die jeweilige Pfaddistanz auf einem zulässigen Pfad bestimmen zu können,wird jedem Knoten ein Datenfeld Knotendistanz zugeordnet.Diese Felder werden zu Beginn des Suchen der zulässigen Flusspfade auf den Wert 1E32 gesetzt,was unendlich bedeuten soll und mit i auf der Zeichenfläche angezeigt wird.

Beim Durchlaufen des Pfades vom Quellen- zum Senkenknoten wird dem Endknoten einer Vorwärtskante als Knotendistanz jeweils das Minimum der Knotendistanz des Anfangsknoten und der Differenz aus Schranke und Fluss der Kante (Kantendistanz) und dem Anfangsknoten einer Rückwärtskante als Knotendistanz das Minimum aus der Knotendistanz des Endknotens und dem Fluss der Kante (Kantendistanz) zugewiesen (jeweils nur wenn diese Knotendistanzen größer als Null sind).

Auf diese Weise wird im Senkenknoten jeweils die Pfaddistanz gespeichert und beim Rückwärtsdurchlaufen entlang des Pfades wird der Fluss jeder Kante um diesen Betrag erhöht bzw. vermindert.
Um das Vorwärts-und Rückwärtsdurchlaufen des Pfades auf einfache Weise realisieren zu können,ist es sinnvoll die entsprechende Methode mit Namen Fluss rekursiv zu programmieren.

Der Algorithmus im Detail:

1)Bestimme den Quellen- und Senkenknoten des Graphen.

2)Setze den Fluss jeder Kante auf Null und den Gesamtfluss auf Null.Fasse den Kanteninhalt als Schranke der Kanten auf.

3)Wiederhole Schritt 4 (und die von 4 eingeschlossene rekursive Methode Fluss) bis gefunden false ist.

4)Setze die Knotendistanz jedes Knoten auf 1E32 (unendlich).Lösche die Besucht-Markierung der Knoten.Setze die Boolesche Variable gefunden auf false.Rufe die rekursive Methode Fluss mit dem Quellenknoten als aktuellem Knoten und dem Senkenknoten als Parameter auf.

Die rekursive Methode Fluss:

5)Wähle die in der Reihenfolge nächste (evtl. die erste) ausgehende oder eingehende Kante des aktuellen Knotens.(Zuerst die ausgehenden Kanten in der Reihenfolge der ausgehenden Kantenliste und danach die eingehenden Kanten in der Reihenfolge der eingehenden Kantenliste) Rufe mit diesen Kanten als Parameter jeweils die Proceduren BesucheausgehendeKante 6) und BesucheeingehendeKante 7) auf.Wenn alle Kanten besucht worden sind,verlasse diese Rekursionsebene und kehre zur vorigen zurück.

6)a)Ermittle Anfangs-und Endknoten der aktuellen Kante sowie Fluss und Schranke.Wenn die Kante noch nicht als besucht markiert wurde und der Fluss kleiner als die Schranke ist,dann markiere die Kante als besucht und speichere das Minimum aus der Knotendistanz des Anfangsknoten und der Differenz Schranke minus Fluss als Knotendistanz des Endknotens.Wenn der Endknoten der Senkenknoten ist,dann setze gefunden auf true und erhöhe den Gesamtfluss um die Knotendistanz vom Endknoten.Speichere die Knotendistanz des Senkenknotens in einer globalen Variablen Distanz.Andernfalls rufe die Methode Fluss rekursiv erneut mit dem Endknoten und dem Senkenknoten als Parameter auf.

6)b)Bei der Rückkehr aus der vorigen Rekursionsebene erhöhe den Fluss der aktuellen Kante um den in der globalen Variablen Distanz gespeicherten Wert.Lösche die Besucht-Markierung der Kante.Verlasse dann die Procedure BesucheAusgehendeKante.

7)a)Ermittle Anfangs-und Endknoten der aktuellen Kante sowie Fluss und Schranke.Wenn die Kante noch nicht als besucht markiert wurde und der Fluss größer als Null ist,dann markiere die Kante als besucht und speichere das Minimum aus der Knotendistanz des Endknoten und dem Fluss als Knotendistanz des Anfangsknotens.Rufe dann die Methode Fluss rekusiv erneut mit dem Anfangsknoten und dem Senkenknoten als Parameter auf.

7)b)Bei der Rückkehr aus der vorigen Rekursionsebene vermindere den Fluss der aktuellen Kante um den in der globalen Variablen Distanz gespeicherten Wert.Verlasse dann die Procedure BesucheEingehendeKante.

Der Quellcode:

Die Datenstruktur:

```
TMaxflussknoten =  class(TInhaltsknoten)
  private
    Distanz_:Extended;
    Ergebnis_:string;
    procedure SetzeDistanz(Di:Extended);
    function WelcheDistanz:Extended;
    procedure SetzeErgebnis(S:string);
    function WelchesErgebnis:string;
  public
    constructor Create;
    property Distanz:Extended read WelcheDistanz write SetzeDistanz;
    property Ergebnis:string read WelchesErgebnis write SetzeErgebnis;
    function Wertlisteschreiben:TStringlist;override;
    procedure Wertlistelesen;override;
    procedure ErzeugeErgebnis;
  end;

  TMaxflusskante = class(TInhaltskante)
  private
```

```
    Fluss_:Extended;
    Ergebnis_:string;
    procedure  SetzeFluss(Fl:Extended);
    function WelcherFluss:Extended;
    procedure SetzeErgebnis(S:string);
    function WelchesErgebnis:string;
  public
    constructor Create;
    property Fluss:Extended read WelcherFluss write setzeFluss;
    property Ergebnis:string read WelchesErgebnis write SetzeErgebnis;
    function Wertlisteschreiben:TStringlist;override;
    procedure Wertlistelesen;override;
    procedure ErzeugeErgebnis;
  end;

  TMaxflussgraph = class(TInhaltsgraph)
  private
    Distanz_:Extended;
    procedure SetzeDistanz(Di:Extended);
    function WelcheDistanz:Extended;
  public
    constructor Create;
    property Distanz:Extended read WelcheDistanz write SetzeDistanz;
    function Wertlisteschreiben:TStringlist;override;{entbehrlich}
    procedure Wertlistelesen;override;{entbehrlich}
    procedure LoescheFluss;
    procedure SetzeKnotenDistanz(Flaeche:TCanvas);
    procedure Fluss(Kno,Endknoten:TKnoten;var Gefunden:Boolean;
      var Gesamtfluss:Extended;Flaeche:TCanvas);
    procedure StartFluss(Flaeche:TCanvas;var Gesamtfluss:Extended);
    procedure BestimmeErgebnis(var SListe:TStringlist);
  end;
```

TMaxflussknoten und TMaxflusskante sowie TMaxflussgraph leiten sich durch Vererbung von TInhaltsknoten,TInhaltskante und TInhaltsgraph ab.

TMaxflussknoten enthält die zusätzlichen Felder Distanz_ und Ergebnis_ zur Aufnahme der Knotendistanz und eines Ergebnisses als String,das aus dem Knoteninhalt und der Knotendistanz besteht.

TMaxflusskante enthält die zusätzlichen Felder Fluss_ und Ergebnis_ zur Aufnahme des Flusses durch die Kante und eines Ergebnisses als String,das aus dem Kanteninhalt,d.h. der Schranke und dem Kantenfluss besteht.

TMaxflussgraph enthält als zusätzliches Feld das Feld Distanz_ zur Aufnahme einer Pfaddistanz,um die jeweils der Fluss der Kanten des Pfades vergrößert oder verkleinert werden muß (je nach Vorwärts-oder Rückwärtskante).

Auf die Felder wird jeweils mittels der entsprechenden Propertys zugegriffen.

```
procedure TMaxflussknoten.ErzeugeErgebnis;
var S:string;
    P:Integer;
begin
  P:=Position;
  Position:=0;
  if Distanz<1E32
  then
   S:=RundeZahltoString(Distanz,TInhaltsgraph(Graph).Knotengenauigkeit) else S:='i';
  SetzeErgebnis(Wert+':'+S+'     ');
  Position:=P;
end;
```

Die Methode TMaxflussknoten.ErzeugeErgebnis erzeugt einen String im Ergebnisfeld eines Knotens,der aus dem Knoteninhalt und der Knotendistanz besteht (Werte größer gleich 1E32 werden als unendlich dargestellt (i).).

```
procedure TMaxflusskante.ErzeugeErgebnis;
var P:Integer;
begin
  P:=Position;
  Position:=0;
  SetzeErgebnis(RundeStringtoString(Wert,
  TInhaltsgraph(Anfangsknoten.Graph).Kantengenauigkeit)+':'
  +RundeZahltoString(Fluss,TInhaltsgraph(Anfangsknoten.Graph).Kantengenauigkeit));
  Position:=P;
end;
```

Die Methode TMaxflusskante.ErzeugeErgebnis erzeugt einen String im Ergebnisfeld einer Kante,der aus dem Kanteninhalt,d.h. der Kantenschranke und dem Kantenfluss der Kante besteht.

```
constructor TMaxflussgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TMaxflussknoten;
  InhaltsKanteclass:=TMaxflusskante;
  Distanz_:=0;
end;
```

Der Constructor von TMaxflussgraph setzt die Datentypen für Knoten bzw. Kanten im Graph TMaxflussgraph auf TMaxflussknoten bzw. auf TMaxflusskante.

```
procedure TMaxflussgraph.LoescheFluss;
var Index:Integer;
begin
  if not  Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
      TMaxflussKante(Kantenliste.Kante(Index)).Fluss:=0;
end;
```

Die Methode LoescheFluss setzt bei allen Kanten des Graphen das Datenfeld Fluss auf Null.

```
procedure TMaxflussgraph.SetzeKnotenDistanz;
var Index:Integer;
begin
  if not Leer then
  for Index:=0 to Knotenliste.Anzahl-1 do
  begin
    TMaxflussknoten(Knotenliste.Knoten(Index)).Distanz:=1E32;
    TMaxflussknoten(Knotenliste.Knoten(Index)).ErzeugeErgebnis;
  end;
end;
```

Die Methode SetzeKnotendistanz setzt bei allen Knoten des Graphen das Feld (Fluss-)Distanz (Knotendistanz) auf 1E32,was unendlich bedeuten soll.

```
procedure TMaxflussgraph.BestimmeErgebnis(var SListe:TStringlist);
var Index:Integer;
    Ka:TMaxflusskante;
begin
  SListe:=TStringlist.Create;
  if not Kantenliste.Leer then
  for Index:=0 to Kantenliste.Anzahl-1 do
  begin
    Ka:=TMaxflusskante(Kantenliste.Kante(Index));
    SListe.Add(Ka.Anfangsknoten.Wert+'->'+Ka.Endknoten.Wert+
      ' Schranke: '+RundeStringtoString(Ka.Wert,Kantengenauigkeit)
      +' Fluss: '+
      RundeZahltoString(Ka.Fluss,Kantengenauigkeit));
  end;
end;
```

Die Methode BestimmeErgebnis erzeugt eine Stringliste SListe,deren Strings jeweils für jede Kante den Anfangs-und Endknoten der Kante,deren Schranke sowie deren Fluss enthält.Die Liste soll im Ausgabefenster angezeigt werden.

```
procedure TMaxflussGraph.StartFluss(Flaeche:TCanvas;var Gesamtfluss:Extended);
var Gefunden:Boolean;
    Index,Zaehle:Integer;
    Startknoten,Zielknoten:TKnoten;

  procedure Ergebniserzeugen;
  var Zaehl:Integer;
  begin
    Knotenwertposition:=0;
    Kantenwertposition:=0;
    if not Knotenliste.leer then
      for Zaehl:=0 to Knotenliste.Anzahl-1 do
        TMaxflussknoten(Knotenliste.Knoten(Zaehl)).Ergebnis:=
          TMaxflussknoten(Knotenliste.Knoten(Zaehl)).Wert;
    if not Kantenliste.leer then
```

```
      for zaehl:=0 to Kantenliste.Anzahl-1 do
        TMaxflusskante(Kantenliste.Kante(Zaehl)).Ergebnis:=
        TMaxflusskante(Kantenliste.Kante(Zaehl)).Wert;
  end;

begin
  Gesamtfluss:=0;                                                          1)
  LoescheFluss;
  Ergebniserzeugen;
  Zaehle:=0;
  for Index:=0 to Knotenliste.Anzahl-1 do
    if Knotenliste.Knoten(Index).EingehendeKantenliste.Leer then          2)
    begin
      Startknoten:=TMaxflussknoten(Knotenliste.Knoten(Index));
      Zaehle:=Zaehle+1;
    end;
  If Zaehle<>1 then
  begin
    ShowMessage('mehrere Anfangsknoten');
    exit;
  end;
  TInhaltsknoten(Startknoten).Farbe:=clblue;
  TInhaltsknoten(Startknoten).Stil:=psDashDot;
  ZeichneGraph(Flaeche);
  ShowMessage('Startknoten: '+Startknoten.Wert);
  Zaehle:=0;
  for Index:=0 to Knotenliste.Anzahl-1 do
    if Knotenliste.Knoten(Index).ausgehendeKantenliste.Leer then          3)
    begin
      Zielknoten:=TMaxflussknoten(Knotenliste.Knoten(Index));
      Zaehle:=Zaehle+1;
    end;
  if Zaehle<>1 then
  begin
    ShowMessage('Mehrere Endknoten');
    exit;
  end;
  TInhaltsknoten(Zielknoten).Farbe:=clgreen;
  TInhaltsknoten(Zielknoten).Stil:=psdash;
  ZeichneGraph(Flaeche);
  ShowMessage('Zielknoten: '+Zielknoten.Wert);
  Repeat                                                                   4)
    SetzeKnotenDistanz;                                                    5)
    LoescheKantenbesucht;                                                  6)
    Gefunden:=false;                                                       7)
    Distanz:=0;                                                            8)
    ZeichneGraph(Flaeche);
    Fluss(Startknoten,Zielknoten,Gefunden,Gesamtfluss,Flaeche)             9)
  until not Gefunden;
  TInhaltsknoten(Zielknoten).Farbe:=clgreen;
  TInhaltsknoten(Zielknoten).Stil:=psdash;
  ZeichneGraph(Flaeche);
end;
```

Bei 1) wird zunächst der Gesamtfluss auf Null und der Fluss durch jede Kante auf Null gesetzt (LoescheFluss).Bei 2) wird ein Knoten als Startknoten (Quelle) gesucht,bei dem die eingehende Kantenliste leer ist und bei 3) ein Zielknoten (Senke),bei dem die ausgehende Kantenliste leer ist.

Bei 4) beginnt die Hauptschleife zur Bestimmung von zulässigen Flusspfaden,die solange durchlaufen wird,bis Gefunden false ist,d.h. kein zulässiger Flusspfad mehr existiert.

Dazu wird zunächst die Distanz der Knoten des Graphen auf den Wert 1 E32 (unendlich gesetzt) (5),die Besucht-Markierung der Kanten gelöscht (6) und Gefunden auf false (7) sowie das Datenfeld Distanz zur Speicherung der Pfaddistanz auf Null gesetzt (8).

Danach wird die rekursive Methode Fluss mit den Parametern u.a. Startknoten (Quelle) und Zielknoten (Senke) sowie dem Referenzparameter Gefunden und Gesamtfluss aufgerufen (9).

```
procedure TMaxflussgraph.Fluss(Kno,Endknoten:TKnoten;
var Gefunden:Boolean;Var Gesamtfluss:Extended;Flaeche:TCanvas);
var Index1,Index2:Integer;
    Kaeingehend,Kaausgehend:TKante;

  procedure Graphzeichnen;
```

```
  begin
    Knotenwertposition:=2;
    Kantenwertposition:=2;
    ZeichneGraph(Flaeche);
    Knotenwertposition:=0;
    Kantenwertposition:=0;
  end;

  procedure BesucheeingehendeKante(Ka:TKante);
  var Kno1,Kno2:TMaxflussknoten;
      Schranke,Fluss:Extended;
      Kante:TMaxFlusskante;
  begin
    if Gefunden then exit;
    Kno1:=TMaxflussknoten(Ka.Endknoten);                                    24)
    Kno2:=TMaxflussknoten(Ka.Anfangsknoten);
    Kante:=TMaxflusskante(Ka);
    Schranke:=StringtoReal(Ka.Wert);
    Fluss:=Kante.Fluss;
    if not Kante.Besucht then                                               25)
    begin
      Kante.ErzeugeErgebnis;
      if Fluss>0 then                                                       26)
      begin
        Kante.Besucht:=true;                                                27)
        Kno2.Distanz:=Minimum(Kno1.Distanz,Fluss);                          28)
        Kno2.ErzeugeErgebnis;
        Fluss(Kno2,Endknoten,Gefunden,Gesamtfluss,Flaeche);                 29)
      end;
      Kante.Besucht:=false;                                                 30)
      if Gefunden then Kante.Fluss:=Kante.Fluss-Distanz;                    31)
      Kno2.ErzeugeErgebnis;
      Kante.ErzeugeErgebnis;
    end;
  end;

  procedure BesucheausgehendeKante(Ka:TKante);
  var Kno1,Kno2:TMaxflussknoten;
      Schranke,Fluss:Extended;
      Kante:TMaxFlusskante;
  begin
    if Gefunden then exit;
    Kno1:=TMaxflussknoten(Ka.Anfangsknoten);                                12)
    Kno2:=TMaxflussknoten(Ka.Endknoten);
    Kante:=TMaxflusskante(Ka);
    Schranke:=StringtoReal(Kante.Wert);
    Fluss:=Kante.Fluss;
    if not Kante.Besucht then                                               13)
    begin
      Kante.ErzeugeErgebnis;
      if Fluss<Schranke then                                                14)
      begin
        Kante.Besucht:=true;                                                15)
        Kno2.Distanz:=Minimum(Kno1.Distanz,Schranke-Fluss);                 16)
        Kno2.ErzeugeErgebnis;
        if Kno2=Endknoten then                                              17)
        begin
          Gefunden:=true;                                                   18)
          Gesamtfluss:=Gesamtfluss+Kno2.Distanz;                            19)
          Distanz:=Kno2.Distanz;                                            20)
        end
        else
          self.Fluss(Kno2,Endknoten,Gefunden,Gesamtfluss,Flaeche);          21)
      end;
      Kante.Besucht:=false;                                                 22)
      if Gefunden then Kante.Fluss:=Kante.Fluss+Distanz;                    23)
      Kante.ErzeugeErgebnis;
      Kno2.ErzeugeErgebnis;
    end;
  end;

begin
  if not Gefunden then
  begin
    if not Kno.AusgehendeKantenliste.Leer then
      for Index1:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
      begin
        Kaausgehend:=Kno.AusgehendeKantenliste.Kante(Index1);               10)
        if Kaausgehend.gerichtet then BesucheausgehendeKante(Kaausgehend);
      end;
      if not Kno.EingehendeKantenliste.Leer then
        for Index2:=0 to Kno.EingehendeKantenliste.Anzahl-1 do
        begin
```

```
          Kaeingehend:=Kno.EingehendeKantenliste.Kante(Index2);                    11)
          if Kaeingehend.Gerichtet then BesucheeingehendeKante(Kaeingehend);
        end;
  end;
end;
```

Bei 10) und 11) werden vom aktuellen Knoten Kno aus alle ausgehenden Kanten und alle eingehenden Kanten nacheinander besucht,indem die entsprechenden Proceduren aufgerufen werden.

Procedure BesucheausgehendeKante:

Anfangs-und Endknoten sowie Fluss und Schranke der Kante werden bestimmt (12).Wenn die Kante noch nicht als besucht markiert wurde (13) und der Fluss durch die Kante kleiner als die Schranke der Kante ist (14),wird die Kante als besucht markiert (15) und als Knotendistanz des Endknoten der Kante das Minimum aus der Distanz des Anfangsknoten und der Differenz Schranke minus Fluss gespeichert (16).

Falls der Endknoten der Kante der Senkenknoten des Graphen ist (17),wird Gefunden auf true gesetzt (18) und der Gesamtfluss um die Pfaddistanz vergrößert (19),als auch die Pfaddistanz in dem globalen Datenfeld Distanz gespeichert (20).Andernfalls wird die Methode Fluss erneut rekursiv mit dem Endknoten der Kante als aktuellem Knoten aufgerufen (21).

Bei Rückkehr aus der vorigen Rekursionsebene wird bei Zurücklaufen des gefundenen Pfades die Besucht-Markierung der Kante gelöscht (22) und,falls ein zulässiger Pfad gefunden wurde,der Fluss der Kante um die Pfaddistanz erhöht (23).

Procedure BesucheeingehendeKante:

Anfangs-und Endknoten sowie Fluss und Schranke der Kante werden bestimmt (24).Wenn die Kante noch nicht als besucht markiert wurde (25) und der Fluss durch die Kante größer als Null ist (26),wird die Kante als besucht markiert (27),und als Knotendistanz des Anfangsknoten der Kante das Minimum aus der Distanz des Endknotens und dem Fluss der Kante gespeichert (28).

Danach wird die Methode Fluss erneut rekursiv mit dem Anfangsknoten der Kante als aktuellem Knoten aufgerufen (29).

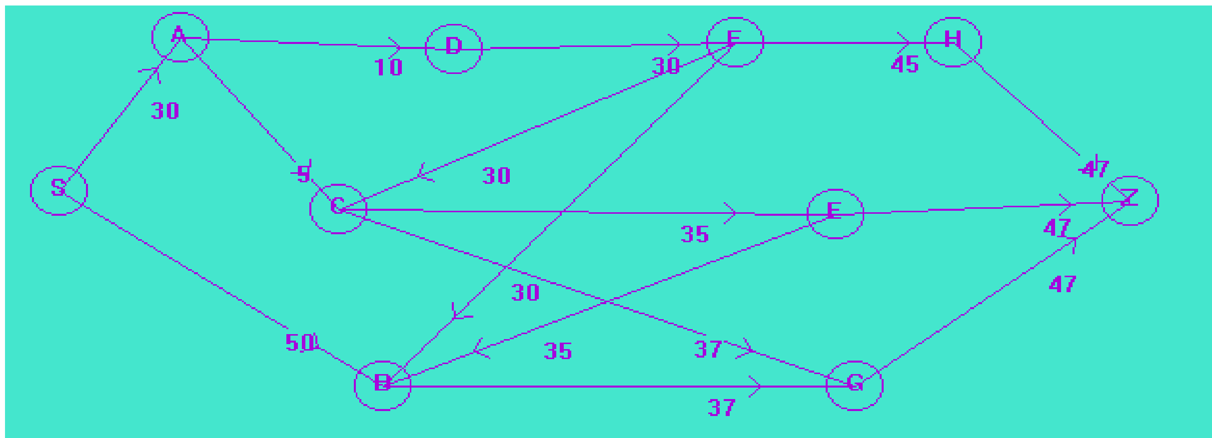
Bei Rückkehr aus der vorigen Rekursionsebene wird bei Zurücklaufen des gefundenen Pfades die Besucht-Markierung der Kante gelöscht (30) und,falls ein zulässiger Pfad gefunden wurde,der Fluss der Kante um die Pfaddistanz vermindert (31).

Bemerkung:

Es ist nicht in jedem Fall gesichert,dass der Algorithmus von Ford-Fulkerson terminiert.Bei ganzzahligen Schranken ist jedoch die Endlichkeit des Verfahrens (nach hier nicht dargestellten Untersuchungen) gegeben.

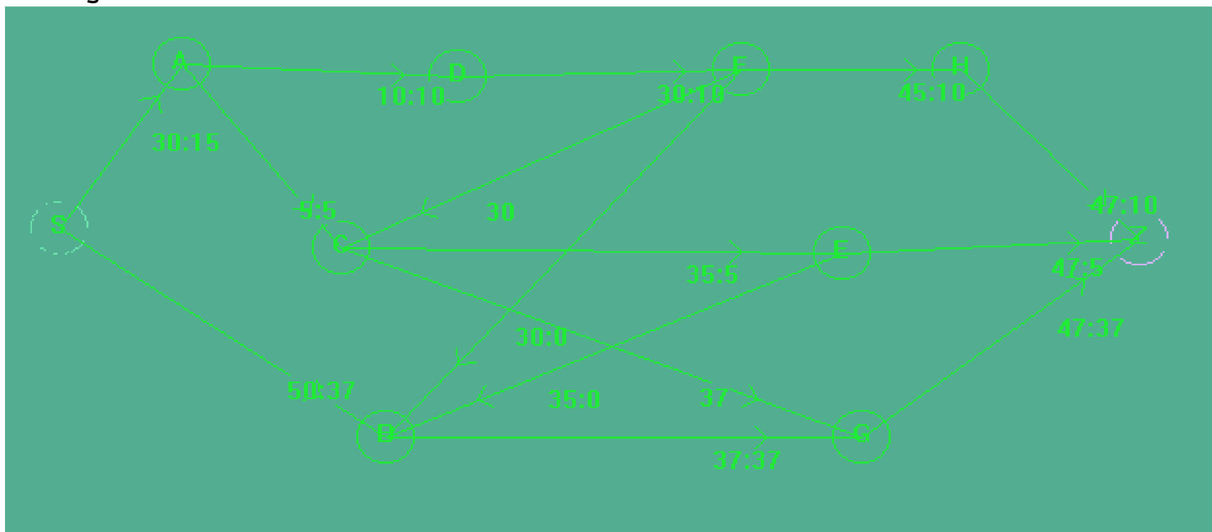
Aufgabe C IX.2:

Gegeben ist der folgende Graph mit Quellen-und Senkenknoten S und Z.Bestimme nach dem Algorithmus von Ford-Fulkerson den maximalen Fluss und den Fluss in den einzelnen Kanten.Bestimme die Ergebnisse (manuell,mit Hilfe des Programms Knotengraph durch den Algorithmus des Menüs Anwendungen/Maximaler Fluss im Demo-Modus oder mittels des eigenen erstellten Quellcodes in der Entwicklungsumgebung,je nach Konzeption).



G047.gra

Lösung:



G047.gra

Kanten,Schranken und Fluss:

S->A Schranke: 30 Fluss: 15
S->B Schranke: 50 Fluss: 37
A->D Schranke: 10 Fluss: 10
A->C Schranke: 5 Fluss: 5
D->F Schranke: 30 Fluss: 10
F->H Schranke: 45 Fluss: 10
F->C Schranke: 30 Fluss: 0
F->B Schranke: 30 Fluss: 0
H->Z Schranke: 47 Fluss: 10
C->E Schranke: 35 Fluss: 5
C->G Schranke: 37 Fluss: 0
E->B Schranke: 35 Fluss: 0
E->Z Schranke: 47 Fluss: 5
B->G Schranke: 37 Fluss: 37
G->Z Schranke: 47 Fluss: 37

maximaler Gesamtfluss: 52

An Hand dieses Beispiels sei noch der **Satz** von Ford-Fulkerson erläutert,der hier ohne Beweis angegeben wird,dessen Aussage aber anschaulich sofort klar ist.

Definition C IX.1:

Wenn man die Menge der Knoten des Graphen in zwei disjunkte Teilmengen X und X′ aufteilt,deren Vereinigungsmenge wieder die Menge aller Knoten des Graphen ergibt,wobei in X der Quellenknoten und in X′ der Senkenknoten liegt,versteht man unter einem Schnitt durch den Graphen die Menge der Kanten,die den Anfangsknoten in X und den Endknoten in X′ haben.Die Summe der maximalen Flüsse durch diese Kanten wird als Kapazität des Schnittes bezeichnet.Wenn der Schnitt so gewählt wird,dass diese Kapazität unter allen möglichen Schnitten minimal ist,heißt der Schnitt ein minimaler Schnitt.

Satz C IX.1 (von Ford-Fulkerson):

a)Die minimale Kapazität eines Schnittes durch den Graphen ist gleich dem maximalen Netzfluss vom Quellenknoten zum Senkenknoten.

b)Genau dann ist ein Fluss ein maximaler Fluss,wenn es keinen Pfad (mit Vorwärts- bzw. Rückwärtskanten) vom Quellenknoten zum Senkenknoten mehr gibt,auf dem der Fluss noch erhöht werden könnte.

(Beweis z.B.:Lit 55, S. 1699)

Erläuterung:

Im obigen Beispiel enthalte die Menge X die Knoten S,A und B und die Menge X′ die restlichen Knoten des Graphen.Der minimale Schnitt besteht dann aus den Kanten A->D,A->C und B->G,deren Kapazität 52 beträgt und gleich dem maximalen Gesamtfluss ist.

Die Schnittkanten sind stets daran zu erkennen,dass bei Ihnen der Fluss gleich der Schranke ist.

Aufgabe C IX.3:

Wie verläuft der minimale Schnitt bei Anwendungsaufgabe C IX.1?

Lösung:

Er besteht aus den Kanten 1->3,4->6,5->6 und 5->7 mit der Kapazität 21.
Die Menge X' besteht aus den Knoten 3,6,7,X aus den übrigen Knoten.

Aufgabe C IX.4:(Einführungsaufgabe:einfaches Transportproblem)

Bemerkung:

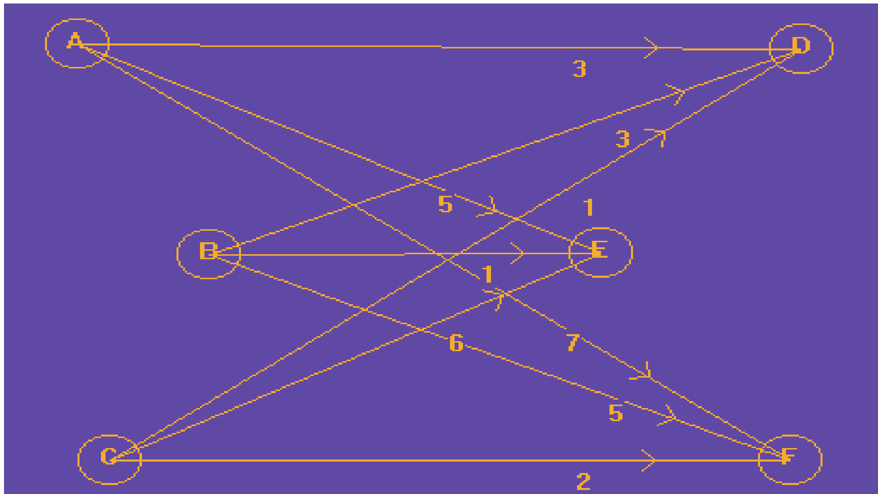
Diese Aufgabe wird im Kapitel C XIII als Aufgabe C XIII.4 wieder aufgegriffen und dort als allgemeines Transportproblem gelöst.

Gegeben ist folgender Graph,dessen Knoten Städte und dessen Kanten Straßen darstellen sollen.In den Fabriken der Städte A,B,C werden Ersatzteile produziert,die von den Fabriken in den Städten D,E und F benötigt werden.

In A werden 5 Tausend,in B 3 Tausend und in C 4 Tausend Ersatzteile pro Tag produziert.In D werden 2 Tausend,in E 4 Tausend und in F 6 Tausend Ersatzteile pro Tag benötigt.Die Abnahmekapazität ist dabei gleich der Produktionskapazität.

Zwischen den Städten A,B und C einerseits und den Städten D,E und F andererseits liegt ein Straßennetz,dessen Straßen jeweils die im folgenden Graphen als Kantenschranken (Kantenwerte) vorgegebenen Transportkapazitäten in Tausend pro Tag haben.

Wie ist der LKW-Verkehr bzw. dessen Ladekapazität zu organisieren,damit die produzierten Ersatzteile bei den Abnehmern in ausreichender Zahl ankommen?



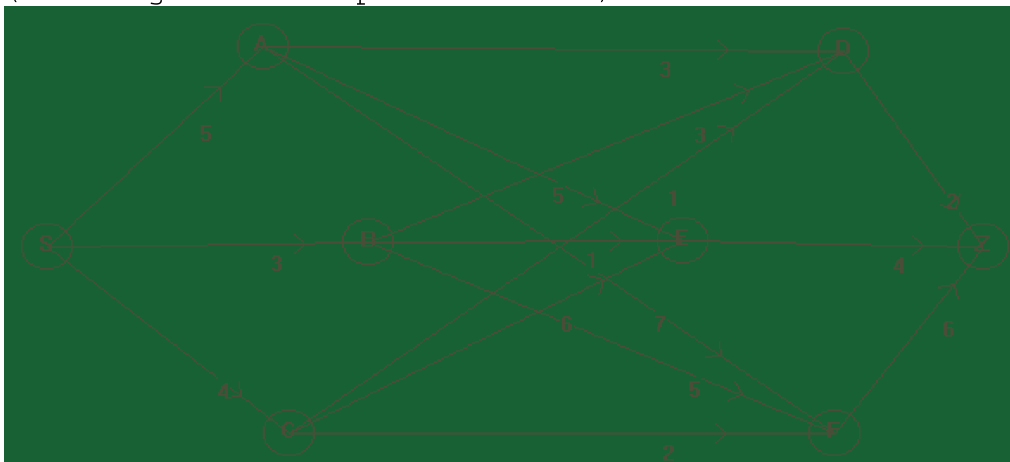
G048.gra

Didakisch-methodische Überlegungen: Es liegt nahe die Transportmenge pro Tag als Fluss aufzufassen und das Problem als Maximalflussproblem zu lösen. Dann müßte durch die Produktionsstädte und durch die Abnahmestädte jeweils ein Fluss in Höhe der Produktionsmenge und der Abnahmemenge fließen. Diese Mengen werden jedoch durch die entsprechenden Kapazitäten beschränkt, so dass man den Produktions- und Abnahmestädten auch wiederum Kanten mit diesen Schranken zuordnen sollte. Zum Einsatz des Maximalflussalgorithmus ist es schließlich noch nötig, diese Kanten dann jeweils mit einem neu zu erzeugenden Quellen- und Senkenknoten zu verbinden.

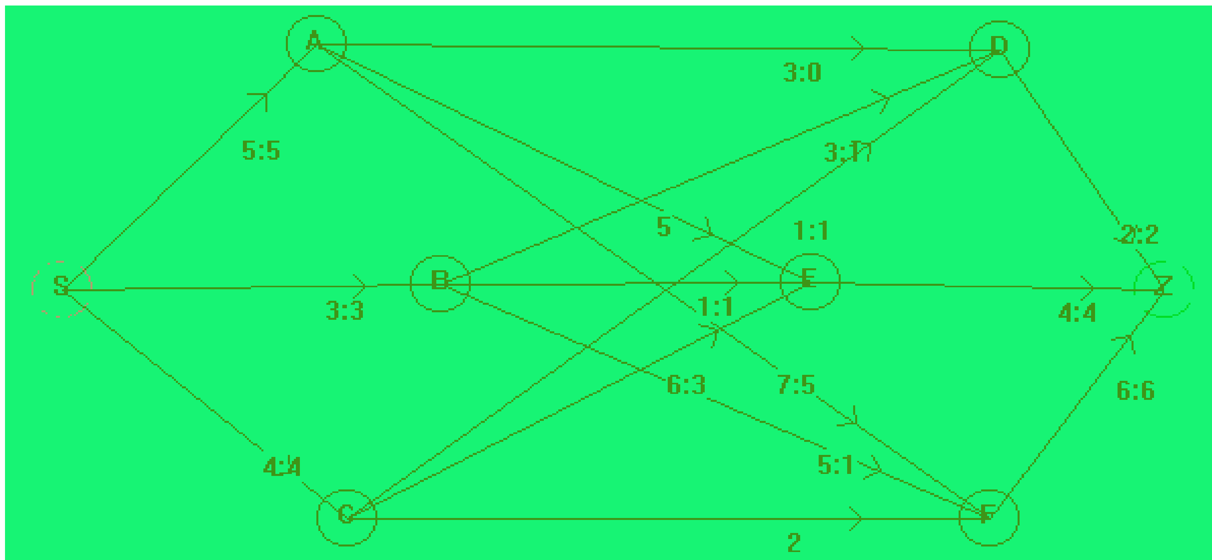
Lösung:

Der Graph wird zu einem Netz durch Startknoten S und Zielknoten Z ergänzt. Diese Knoten werden jeweils durch Kanten mit den Knoten A,B und C bzw D,E und F verbunden, deren Schranken (Kantenwerte) den Produktionskapazitäten bzw. Verbrauchskapazitäten entsprechen.

(Zahlenangaben der Graphen in Tausend)


G049.gra

Anschließend ergibt der Algorithmus Maximaler Netzfluss des Menüs Anwendungen (Programm Knotengraph) die Lösung (bzw. die Ausführung des selbst erstellten Quellcodes in der Entwicklungsumgebung):



G049.gra

Der Fluss durch die einzelnen Kanten gibt die Transportkapazität (Anzahl der Ersatzteile in Tausend) der LKWs pro Tag zwischen den einzelnen Städten an.

Angaben in Tausend:

Kanten,Schranken und Fluss:

A->F Schranke: 7 Fluss: 5
A->E Schranke: 5 Fluss: 0
A->D Schranke: 3 Fluss: 0
B->E Schranke: 1 Fluss: 1
B->D Schranke: 3 Fluss: 1
B->F Schranke: 5 Fluss: 1
C->D Schranke: 1 Fluss: 1
C->E Schranke: 6 Fluss: 3
C->F Schranke: 2 Fluss: 0
S->A Schranke: 5 Fluss: 5
S->B Schranke: 3 Fluss: 3
S->C Schranke: 4 Fluss: 4
D->Z Schranke: 2 Fluss: 2
E->Z Schranke: 4 Fluss: 4
F->Z Schranke: 6 Fluss: 6

maximaler Gesamtfluss: 12 (Tausend)

Auch im nächsten Kapitel wird der Algorithmus von Ford-Fulkerson zur Lösung des Problems Maximales Matching im paaren Graphen benötigt.

C X Maximales Matching

Ein maximales Matching läßt sich in paaren Graphen durch den im vorigen Kapitel besprochenen Algorithmus von Ford-Fulkerson mittels der Lösung eines maximalen Flussproblems lösen,indem jeweils ein zusätzlicher Quellen-und Senkenknoten erzeugt wird,die jeweils mit den Knoten der beiden disjunkten Knotenmengen des paaren Graphen durch Kanten der Schrankenbewertung 1 verbunden werden.Ein maximaler Fluss bestimmt dann eine Auswahl von maximal vielen Kanten zwischen den beiden Knotenmengen, und induziert somit ein maximales Matching.Wenn man will,läßt sich also eine Unterrichtsreihe konzipieren,in der die interessanten Aufgabenstellungen gelöst werden können,die sich auf den Heiratssatz von Hall unter Konstruktion eines maximalen Matchings zurückführen lassen,ohne unbedingt einen neuen Algorithmus einführen,besprechen oder programmieren zu müssen.Erst wenn man ein maximales Matching in einem beliebigen nicht unbedingt paaren Graphen erzeugen will,ist es nötig,den neuen Algorithmus der nach der Methode Suche einen erweiternden Weg arbeitet, einzuführen.Um den Begriff des maximalen Matchings im Mathematikunterricht zu besprechen,ist dies jedoch wie gesagt nicht unbedingt erforderlich und nur in einem (Informatik-)Kurs zu empfehlen,in dem es darum geht,die Idee dieses interessanten Algorithmus zu vermitteln und in Quellcode umzusetzen. Ansonsten genügt es sicherlich wieder zur Demonstration dieses Verfahrens den Demo-Modus des fertigen Programm Knotengraph zu benutzen (Menü Anwendungen/Maximales Matching).Die Aufgaben dieses Kapitels sind auch dazu geeignet,den wichtigen mathematischen Begriff der Relation und Funktion zu veranschaulichen bzw. zu vertiefen.

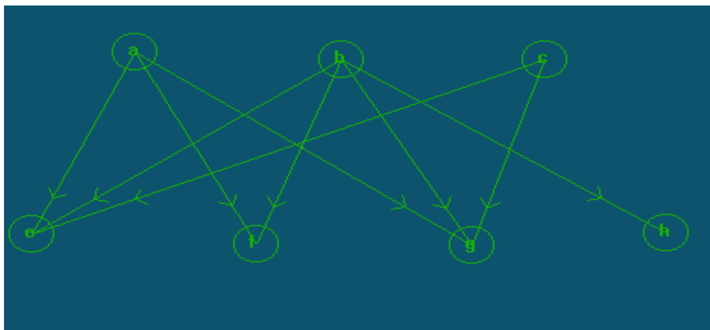
Aufgabe C X.1 (Einstiegsproblem):

Betrachtet wird eine Menge von 3 Jungen a,b und c und 4 Mädchen e,f,g und h.Junge a hat die Freundinnen e,f und g,Junge b hat die Freundinnen e,f,g und h und Junge c die Freundinnen e und g.
Ist es möglich,eine (eindeutige) Zuordnung, (d.h. Funktion jeder Junge zu genau einem Mädchen) so zu finden,dass jeder Junge eine seiner Freundinnen heiraten kann?Unter welchen Bedingungen ist dies stets möglich?

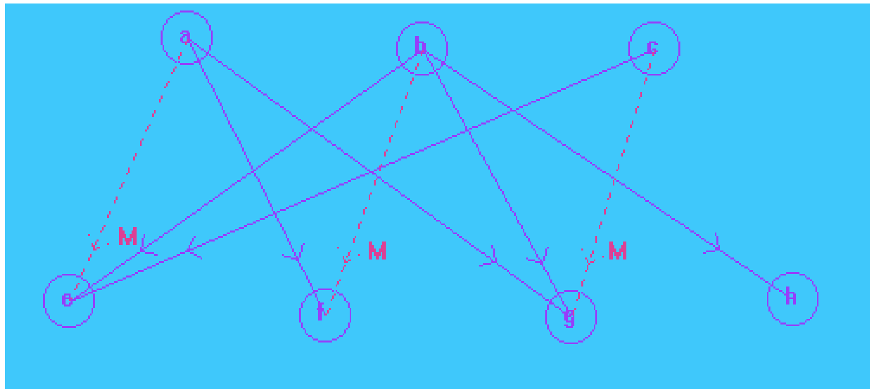
Die Relation Junge-Freundinnen kann durch einen Graph dargestellt werden,wobei den Jungen und Mädchen jeweils ein Knoten des Graphen entspricht,und die Relation durch gerichtete Kanten zwischen entsprechenden Knoten dargestellt wird.

Erzeuge einen entsprechenden Graph mit dem Programm Knotengraph,und ermittle eine Lösung zuerst per Hand.Benutze danach entweder das Menü Anwendungen,Maximales Matching zur Ermittlung einer möglichen Paarung oder wende den Algorithmus von Ford-Fulkerson an und löse dazu zunächst die Aufgabe C X.V weiter unten.

Lösung:



G050.gra



G050.gra

Kanten des maximalen Matchings:

Kante: a-e
Kante: b-f
Kante: c-g

Möglich sind auch noch andere Lösungen.

Das durch die Aufgabe vorgegebene Problem nennt sich das Heiratsproblem.

Es läßt sich allgemein wie folgt formulieren:

Gegeben ist ein paarer,gerichteter Graph G mit der Zweiteilung der Knoten in eine Menge X,von deren Knoten die Kanten ausgehen und der Menge Y,in deren Knoten die Kanten enden.
Unter welchen Bedingungen gibt es ein maximales Matching in G,so dass alle Knoten von X gesättigt sind?

Definition C X.1:

Gegeben sei ein beliebiger Graph G.Eine Untermenge U der Kanten von G heißt Matching,wenn keine zwei Kanten aus U einen gemeinsamen Knoten als Anfangs- oder Endknoten besitzen.

(Ausgeschlossen ist auch,dass derselbe Knoten bei der einen Kante Anfangsknoten und bei der anderen Kante Endknoten ist.)

Ein Knoten von G heißt gesättigt,wenn er Anfangs-oder Endknoten einer Kante von U ist.

Ein Knoten von G,der bei keiner der Kanten des Matchings U Anfangs-oder Endknoten ist,heißt isolierter Knoten.

Ein Matching heißt perfekt,wenn es keine isolierten Knoten im Graph G gibt.

Ein Matching M heißt maximal,wenn der Graph G kein Matching mehr enthält,dass eine größere Kantenzahl als M enthält.

Wenn der Graph G paar ist und in die Knotenmengen X und Y zerfällt,wobei jede Kante von einem Knoten aus X zu einem Knoten aus Y führt,dann heißt die Menge aller Zielknoten (die in Y liegen),der von den Knoten einer beliebigen Teilmenge T in X ausgehenden Kanten die Menge der Nachbarknoten N(T) zu T als Teilmenge in Y.

Eine notwendige Bedingung zur Lösung der obigen Anwendungsaufgabe ist sicher,dass die Anzahl der Freundinnen zu jeder beliebigen Jungen-Teilmenge größer/gleich als die Anzahl der Jungen ist.Diese Bedingung ist für die Lösbarkeit des Problems auch hinreichend.

Dies besagt der Heiratssatz von Hall:

Satz C X.1:

Es sei G ein paarer Graph mit der Teilung der Knoten in die Mengen X und Y.G hat genau dann ein (maximales) Matching,durch das jeder Knoten in X gesättigt wird,wenn für jede Teilmenge T in X bezüglich der Nachbarknoten von N(T) gilt:

$$|N(T)| \geq |T|$$

Mit den Betragstrichen ist hier die Anzahl der Elemente gemeint.

Beweis:

OBdA, um den Beweis zu veranschaulichen, wird die Bezeichnungsweise aus Anwendungsaufgabe X.1 nämlich Jungen,Mädchen,Freundinnen und heiraten verwendet.

Eine beliebige Teilmenge von Jungen kann sicher nur dann heiraten,wenn die Anzahl ihrer Freundinnen mindestens genau so groß ist,wie die Anzahl der Jungen.Damit ist die Notwendigkeit der Bedingung gezeigt.

Durch vollständige Induktion über die Anzahl der Jungen j wird gezeigt,dass die obige Ungleichung zur vollständigen Verheiratung aller Jungen hinreichend ist.

Wenn j=1 ist,besagt die Ungleichung,dass mindestens eine Freundin vorhanden sein muß.Damit ist ein Heiratspartner gefunden.

Annahme:

Für alle Mengen von T Jungen,wobei j=1,2,...n-1 ist,ist das Problem der Verheiratung schon gelöst.

Es sei jetzt $|X|=n$.

Fallunterscheidung:

I)Für alle echten Teilmengen $T \subset X$ gilt $|N(T)| > |T|$.

Der Junge x_1 aus X ist sicher mit einer Freundin y aus Y durch eine Kante verbunden (hat y als Freundin).Durch geeignete Nummerierung kann erreicht werden,dass $y=y_1$ ist.

Es sei nun $X'=X\setminus\{x_1\}$ und $Y'=Y\setminus\{y_1\}$.G' sei der Graph,der aus G entsteht,indem alle Kanten,die von x_1 und y_1 ausgehen oder dort enden aus dem Graphen von G einschließlich der Knoten x_1 und y_1 gelöscht werden.
Dann ist $|X'|=n-1$.

Es sei X'' eine beliebige Teilmenge in X'.Dann gilt in G'

$|N(X')| \geq |X'|$,da in N(X') nur höchstens der Knoten y_1 fehlen kann,zu dem Kanten von X' in G führen könnten und wegen $|N(X')| > |X'|$ in G.

Nach Induktionsvoraussetzung besteht dann aber in G' ein maximales Matching,bei dem jeder Knoten in X' gesättigt wird (wegen $|X'|=n-1$).Mit der Zuordnung x_1 zu y_1 entsteht daraus ein maximales Matching in G,bei dem alle Knoten in X gesättigt sind.

II)Es gibt eine Teilmenge X'' in X mit $|N(X'')| = |X''|$.

Es sei Y''=N(X'') und es sei X' die Komplementmenge zu X'' in X und Y' die Komplementmenge zu Y'' in Y.
Bei leerer Menge X' ist die Behauptung schon gezeigt.Ansonsten sei G' der Graph,der entsteht,wenn man in G alle Kanten, die von Knoten in X'' und

Y'' ausgehen bzw. enden,löscht und die Knoten aus X'' und Y'' löscht.Der Graph G' hat dann die Knotenmengen X' und Y'.

Dann ist $|X'|<n$,so dass für X' die Induktionsannahme gilt.
Dann enthält G' eine maximales Matching.

Wäre dies nicht so,dann gibt es eine Teilmenge in T in X' mit $|N(T)|<|T|$ nach Induktionsvoraussetzung wegen der Äquivalenz der Behauptung.

Es sei $X'''=T\cup X''$.Dann gilt $|N(X''')|<|X'''|$ in G,da alle Kanten,die mit Knoten aus X'' verbunden sind,zu Knoten aus Y'' führen oder von dort ausgehen und nicht mit Knoten aus Y'.

Daher vergrößert sich die Zahl der Knoten auf beiden Seiten der Ungleichung $|N(T)|<|T|$ um dieselbe Zahl.

Die Ungleichung $|N(X''')|<|X'''|$ in G ist aber ein Widerspruch zur Voraussetzung.

Also existiert ein maximales Matching in G',das durch Zunahme der Kanten zwischen X'' und Y'' zu einem maximalen Matching in G,bei dem alle Knoten aus X gesättigt sind,ergänzt werden kann.

Aufgabe C X.2:

Gegeben ist ein paarer Graph.Die Knoten a,b,c,d bedeuten 4 Lehrer und die Knoten e,f,g,h bedeuten 4 Klassen,denen die Lehrer als Klassenlehrer zugeordnet werden sollen.

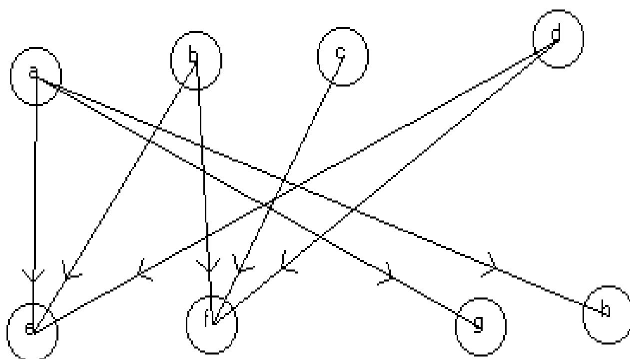
Ein Lehrer kann gemäß seiner Fächerkombination in mehreren Klassen möglicher Klassenlehrer sein.

Diese Möglichkeiten werden durch die Kanten des folgenden Graphen beschrieben.

Das Problem heißt Personalzuteilungsproblem.

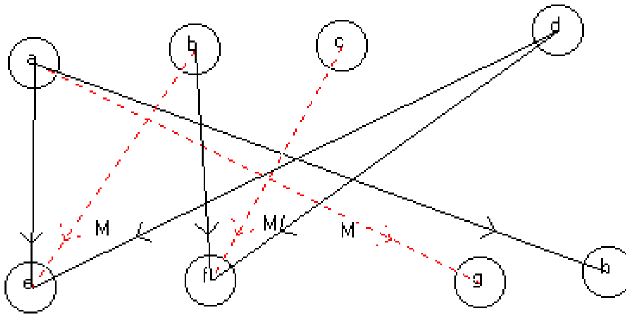
Gibt es eine Zuordnung,so dass jeder Lehrer Klassenlehrer genau einer Klasse unter Beachtung der Fächerkombination ist?

Versuche das Problem zunächst per Hand zu lösen und benutze dann entweder das Programm Knotengraph,Menü Anwendungen/Maximales Matching oder aber analog dem Verfahren in Aufgabe C X.V den Algorithmus von Ford-Fulkerson.



G051.gra

Lösung:



G051.gra

Kanten des maximalen Matchings:
Kante: a-g
Kante: b-e
Kante: c-f

Wie man sieht kann Lehrer d nicht zugeordnet werden.Da es sich um ein maximales Matching handelt,gibt es auch bei anderen Kombinationen nicht mehr als 3 Matchingkanten,so dass immer ein Lehrer und eine Klasse übrigbleibt.

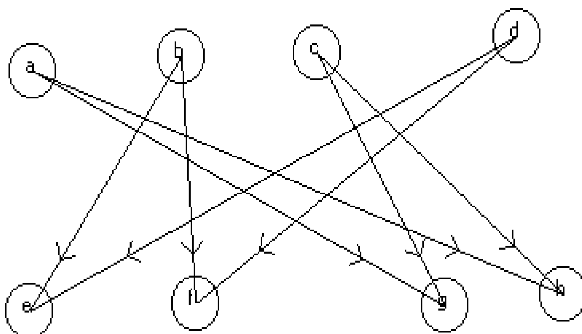
Der Grund dafür ist sofort dem Heiratssatz von Hall zu entnehmen:

Die 3 Knoten (Lehrer) b,c und d haben jeweils nur Kanten,die den 2 Knoten (Klassen) e und f zugeordnet sind.Daher ist $|N(T)|<|T|$ mit T={b;c;d}.Also gibt es kein Matching,das alle Lehrerknoten absättigt,weil die Voraussetzungen des Satzes von Hall nicht gegeben sind.

Aufgabe C X.3:

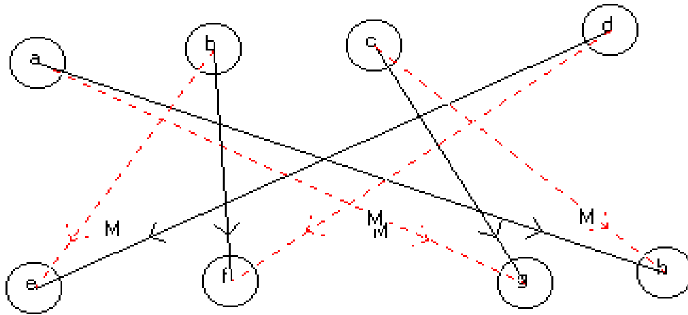
Lösche die Kante a-e im Graph von Aufgabe C X.2 und füge stattdessen eine Kante zwischen c und h ein.Lösche außerdem die Kante c-f und füge dafür die Kante c-g ein.

Lässt sich in dem nun entstandenen Graph das Personalzuteilungsproblem lösen?
Löse per Hand und danach mit dem Programm Knotengraph/Menü Anwendungen,Maximales Matching oder aber analog dem Verfahren in Aufgabe C X.V mit dem Algorithmus von Ford-Fulkerson.



G052.gra

Lösung:



G052.gra

Kanten des maximalen Matchings:
Kante: a-g
Kante: b-e
Kante: d-f
Kante: c-h
Das Matching ist perfekt!

Das Matching ist perfekt,d.h. das Personalzuteilungsproblem läßt sich lösen.

Der Graph hat folgende Besonderheit:

Jeder (ungerichtete) Knotengrad ist 2.Überprüfe diese Tatsache auch mit dem Menü Eigenschaften/Knoten anzeigen im Ausgabefenster.

<u>Definition C X.2:</u>

Ein Graph,bei dem jeder (ungerichtete) Knotengrad gleich einer festen Zahl r ist,heißt regulär der Ordnung r.

<u>Satz C X.2:</u>

Ein paarer,regulärer Graph der Ordnung r hat stets ein maximales perfektes Matching,in dem alle Knoten der Teilung X und Y abgesättigt sind.

Beweis:

Von jedem Knoten einer Teilmenge T aus X gehen r Kanten zu Knoten aus Y weg.Es gilt:

$|N(T)|=r|T|$.

Von den Kanten die zu Knoten aus N(T) gehen wiederum r Kanten aus.Unter diesen müssen die Kanten sein,die von Knoten aus T ausgingen.

Also gilt $r.|N(T)|>r|T|$.Daraus folgt $|N(T)|>|T|$.

Also sind die Voraussetzungen des Satzes von Hall erfüllt.

Wenn jetzt dieselbe Argumentation auf beliebige Teilmengen der Menge Y angewendet wird,ist der Satz bewiesen.

Die Aussage des Satzes läßt sich für r=2 folgendermaßen veranschaulichen:

Wenn je zwei Lehrer einer Klasse und je zwei Klassen jeweils einem Lehrer zugeordnet sind,gibt es jeweils ein perfektes maximales Matching,das eine Zuordnung von genau einem Lehrer zu einer Klasse beschreibt,wobei keine Klasse und kein Lehrer übrigbleibt.

Ein anderes Beispiel ist:

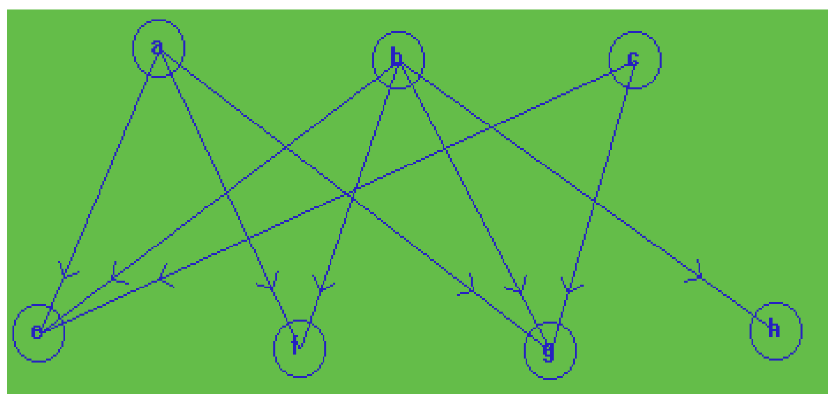
Wenn je 2 von 10 Schüler einer Klasse eine von 10 gestellten Aufgaben lösen,und jede Aufgabe genau von 2 Schülern gelöst wird,dann kann anschließend jede Aufgabe von genau einem Schüler an der Tafel vorgerechnet werden.

Aufgabe C X.4:

Vorgegeben sei der Graph von Aufgabe C X.1.

Ermittle in dem Graphen die Zahl der Kanten eines maximalen Matching.Versuche anschließend in dem Graph solange Knoten zu löschen,bis der Graph keine Kanten mehr besitzt.

Welches ist die minimale Anzahl solcher Knoten,die für einen Kantenlosen Graph gelöscht werden müssen?



G050.gra

Lösung:

Durch Löschen der 3 Knoten a,b und c wird der Graph Kantenlos:
(Menü Knoten löschen/Programm Knotengraph)

Außerdem ist die Kantenzahl eines maximalen Matching ebenfalls 3.

Dieser Sachverhalt gilt ganz allgemein,und ist Inhalt des Satzes von König,der hier ohne Beweis angegeben wird:

Satz C X.3:

In einem paaren Graph ist die Kantenzahl eines maximalen Matching gleich der Minimalzahl der Knoten,die aus dem Graph gelöscht werden müssen,damit er Kantenlos wird.

Wie läßt sich nun ein maximales Matching bestimmen?

Eine Methode für paare Graphen ist es,auf den Algorithmus von Ford-Fulkerson aus dem Kapitel C IX zurückzugreifen und einen maximalen Fluss zu bestimmen.

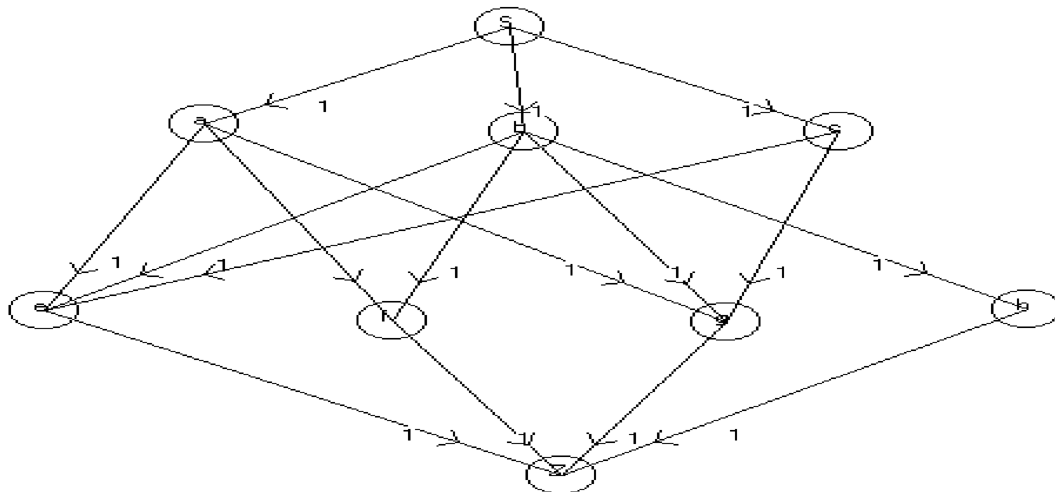
Didaktisch-methodische Vorbemerkungen: Das Bestimmen eines maximalen Matchings in einem paaren Graph läßt sich als Einfaches Transportproblem (siehe Aufgabe C IX.4) auffassen,wobei die Produktions-und Abnahmekapazitäten

jeweils pro Knoten 1 sind,da maximal eine Kante von und zu jedem Knoten führen soll.Aus demselben Grund sind auch alle übrigen Kantenschranken 1,da eine Kante entweder gewählt wird oder nicht (Fluss 1 oder 0,d.h. Interger-werte).

Aufgabe C X.5:

Ergänze den Graph von Anwendungsaufgabe C X.1 um einen Startknoten (Quellenknoten) S und einen Zielknoten (Senkenknoten) Z.Ergänze von S aus jeweils gerichtete Kanten mit dem Wert 1 als Schranke vom Datentyp Integer zu den Knoten a,b und c (der Menge X) und von den Knoten e,f,g und h (der Menge Y) jeweils gerichtete Kante mit dem Wert 1 als Schranke vom Datentyp Integer zum Knoten Z.Ordne jeweils jeder Kante des ursprünglichen (paaren) Graphen einen Integerwert 1 als Schranke zu.

Der Graph sieht dann folgendermaßen aus:

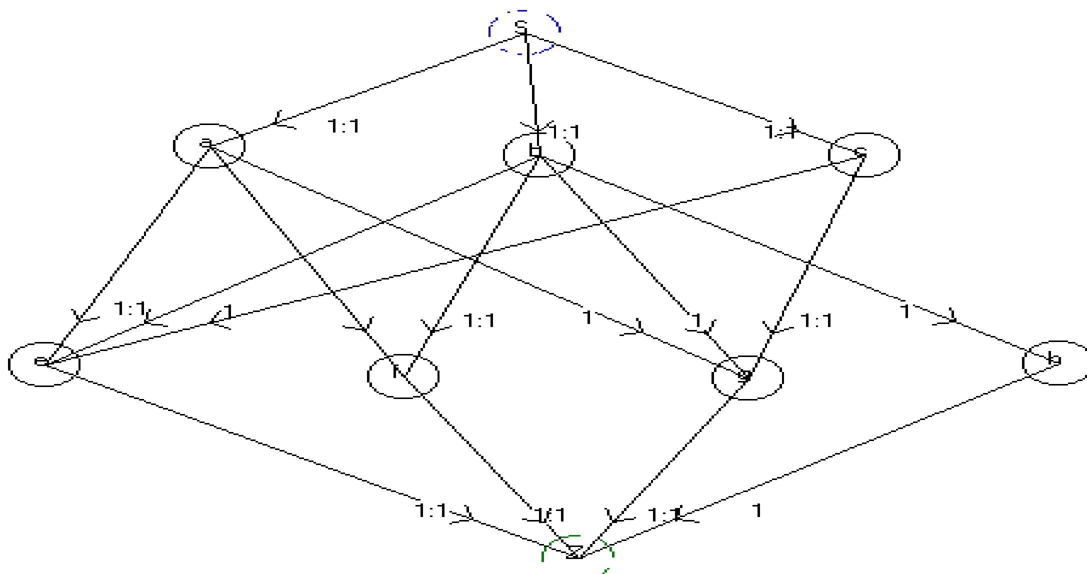


G053.gra

Lasse dann anschließend den Algorithmus des Menüs Anwendungen/Maximaler Netzfluss des Programms Knotengraph ablaufen.Ermittle die Kanten des ursprünglichen paaren Graphen,die den Fluss 1 besitzen.

Überprüfe durch Ablauf des Algorithmus des Menüs Anwendungen/Maximales Matching ,dass gerade diejenigen Kanten den Fluss 1 besitzen,die die Kanten des maximalen Matchings sind.

Lösung:



G053.gra

Kanten,Schranken und Fluss:
a->e Schranke: 1 Fluss: 1 Matching
a->g Schranke: 1 Fluss: 0
a->f Schranke: Fluss: 0
b->f Schranke: 1 Fluss: 1 Matching
b->g Schranke: 1 Fluss: 0
b->h Schranke: 1 Fluss: 0
b->e Schranke: 1 Fluss: 0
c->g Schranke: 1 Fluss: 1 Matching
c->e Schranke: Fluss: 0
S->a Schranke: 1 Fluss: 1
S->b Schranke: 1 Fluss: 1
S->c Schranke: 1 Fluss: 1
e->Z Schranke: 1 Fluss: 1
f->Z Schranke: 1 Fluss: 1
g->Z Schranke: 1 Fluss: 1
h->Z Schranke: 1 Fluss: 0
maximaler GesamtFluss: 3

Das Beispiel zeigt,dass für paare Graphen (mit gerichteten Kanten von X nach Y) ein maximales Matching folgendermaßen erzeugt werden kann:

Erzeuge einen Start-und Zielknoten und verbinde jeweils vom Startknoten aus alle Knoten der Menge X durch gerichtete Kanten mit der Schranke 1 vom Datentyp Integer.Verbinde auch alle Knoten der Menge Y mit dem Zielknoten durch gerichtete Kanten mit der Schranke 1 vom Datentyp Integer.

Lasse dann den Algorithmus von Ford-Fulkerson ablaufen,und ermittle die Kanten des paaren Graphen,die den Fluss 1 besitzen.Diese sind dann die Kanten eines maximalen Matchings.

Begründung:

Durch jeden Knoten der Mengen X und Y kann nur der Fluss 1 fließen,da sie nur durch eine Kante mit dem Start-bzw. Zielknoten verbunden sind.

Daher können niemals zwei Kanten des paaren Graphen,die einen gemeinsamen Anfangs-und Endknoten haben,den Fluss 1 besitzen.

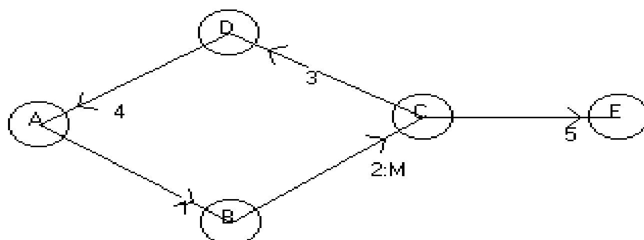
Da ein maximaler Fluss erzeugt wird,hat jede Kante des paaren Graphen den Fluss 1 oder 0 (0 falls kein Fluss durch eine Kante erzeugt wurde),und es wird eine maximal mögliche Anzahl von Kanten mit dem Fluss 1 versehen.

So läßt sich das Problem des maximalen Matchings für paare Graphen auf das Problem des maximalen Netzflusses zurückführen.

Wie aber läßt sich ein maximales Matching für einen nicht paaren Graph erzeugen?

<u>Algorithmus Maximales Matching/Erweiterndernden Weg suchen</u>

Die Idee des Algorithmus sei an dem folgenden Graphen dargestellt.



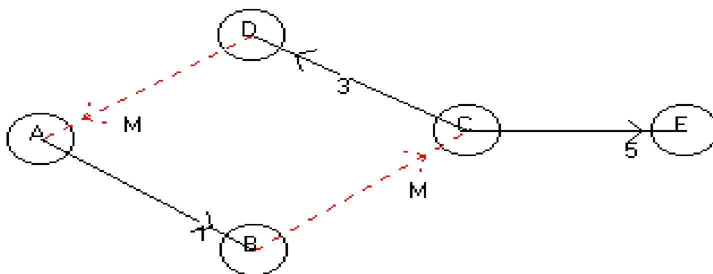
G054.gra

Aufgabe C X.6:(Einstiegsaufgabe)

Erzeuge den obigen Graphen,indem die Knoten in der Reihenfolge B,C,D,E,A eingefügt werden (dadurch wird die Kante 2 zuerst untersucht) mit dem Programm Knotengraph,und beobachte im Demomodus oder Einzelschrittmodus mittels des Menüs Anwendungen/Maximales Matching den Ablauf des Algorithmus.Lasse dabei kein Anfangsmatching erzeugen.Versuche das Prinzip des Verfahrens zu erkennen.

Erzeuge auch andere Graphen und beobachte den Ablauf des Algorithmus im Demomodus.

Lösung:



G055.gra

Beobachtung und problemorientierte Erarbeitung:

Es seien schon einige Kanten des Graphen als zum Matching gehörend bestimmt.Im folgenden werden solche Kanten als Matchingkanten bezeichnet.In dem Graph ist dies die Kante 2.Sie wird mit M markiert.

Um nun weitere Matchingkanten zu bestimmen,wird geprüft,ob es noch isolierte Knoten im Graph gibt.Wenn es keine isolierten Knoten mehr gibt,kann das Matching nicht mehr vergrößert werden,weil jeweils alle Knoten zu Kanten des Matchings gehören.In diesem Fall ist das Matching perfekt.

Wenn es noch isolierte Knoten gibt,wird versucht einen sogenannten erweiternden Weg von diesem Knoten zu finden.

In obigem Graphen ist A ein isolierter Knoten.Von A aus werden nun nacheinander alle ausgehenden und eingehenden Kanten bzw. ungerichtete Kanten (die keine Schlingen sind) darauf überprüft,ob sie als Zielknoten einen Knoten haben,der zu einer Matchingkante des Graphen gehört.

Wenn dies für eine Kante nicht erfüllt ist,ist der Zielknoten ebenfalls ein isolierter Knoten,und diese Kante kann als Anwärter A für eine Matchingkante markiert werden.

Im obigen Beispiel ist dies für die Kante 4 der Fall,die dann mit A markiert würde und dann im nächsten Schritt zur Matchingkante mit der Bezeichnung M umgefärbt würde.

Im obigen Beispielgraph würde jedoch zuerst in der Reihenfolge der Kanten von Knoten A aus die Kante 1 untersucht,so dass ein anderer Weg eingeschlagen wird:

Die Kante 1 besitzt nämlich einen Zielknoten B,von dem die Matchingkante 2 ausgeht.Dann wird der Zielknoten B der Kante 1 mit X bezeichnet,und es wird der Zielknoten C dieser Matchingkante wiederum bestimmt,der mit Z bezeichnet wird.

Im Knoten Y=B wird als letzte besuchte Kante die Kante 1 gespeichert und im Knoten Z=C die Kante 2.

Die Kante 1 erhält die Markierung A (für Anwärter) und die Kante 2 die Markierung L (für Loosing Matching).

Dasselbe Verfahren wird nun mit dem Knoten Z=C,der dann zum Knoten X als Ausgangsknoten wird,in derselben Art und Weise durchgeführt.

Die Kante 3 wird nun untersucht,und es wird festgestellt,dass der Zielknoten D nicht zu einer Matchingkante gehört.Damit wird diese Kante mit A markiert.

Damit ist ein so genannter erweiternder Weg gefunden,da der Pfad in einem isolierten Knoten endet,der nicht der Startknoten A ist.

Die Kanten auf diesem erweiternder Weg 1 2 3 werden jetzt „umgefärbt".Aus der Markierung A wird die Markierung M (Matchingkante),und aus Kante der Markierung L werden nicht markierte Kanten.

Damit sind jetzt zwei Kanten die neuen Matching-Kanten,nämlich die Kanten 1 und 3,und das Matching wurde um eine Kante vergrößert.

Wenn das Suchen des erweiternden Weges nicht erfolgreich sein sollte,dadurch,dass der Weg in keinem isolierten Knoten bzw. im Startknoten endet,wird die Markierung der Kanten wieder rückgängig gemacht:aus L wird M und aus Kanten mit der Markierung A werden wieder unmarkierte Kanten.So wird der ursprüngliche Zustand wieder hergestellt.

In einem neuen Durchgang wird nun untersucht, ob ein weiterer erweiternder Weg gefunden werden kann.

Da E ein isolierter Knoten ist,ist ein neuer erweiternder Weg-Kandidat von E aus der Weg 5 3 4 1 2.Bei 2 wird jedoch als Zielknoten C ein Knoten erreicht,der schon besucht wurde,was daran erkannt wird,dass zu jedem Knoten die letzte durchlaufende Kante gespeichert wird.(Dies ist hier die Kante 5).

Daher gibt es keinen erweiternden Weg von E aus.Andere isolierte Knoten gibt es nicht mehr,so dass jetzt er Algorithmus abgebrochen wird.

Damit ist die maximale Zahl der Kanten des Matchings 2,nämlich die Kanten 1 und 3.

Dies führt zu folgender Definition und verbaler Beschreibung des Algorithmus:

Definition C X.3:

Ein erweiternder Weg ist ein Pfad zwischen zwei isolierten,verschiedenen Knoten des Graphen,auf dem sich Matchingkanten und Nicht-Matchingkanten kontinuierlich abwechseln.Die beiden Endkanten sind dabei Nichtmatchingkanten.Eine etwa vorhandene Kantenrichtung wird bei diesem Pfad nicht berücksichtigt,d.h. der Graph wird als ungerichtet aufgefaßt.

Unter dem Umfärben eines erweiternden Weges versteht man,dass die Nicht-Matchingkanten zu Matchingkanten werden und die Matchingkanten zu Nicht-Matchingkanten.Dadurch wird das Matching um eine Kante erhöht.

Verbale Beschreibung des Algorithmus:

0)Wiederhole die folgenden Schritte,bis entweder kein isolierter Knoten mehr vorhanden ist,oder aber kein weiterer erweiternder Weg von allen noch isolierten Knoten mehr gefunden werden kann:

1)Bestimme,falls existent,den nächsten isolierten Knoten des Graphen (z.B. in der Reihenfolge der Knotenliste).

2)Suche einen erweiternden Weg von diesem Knoten.

3)Falls ein solcher Weg existiert,färbe die Kanten um.

Um den Algorithmus zu beschleunigen,ist es sinnvoll vor Ablauf des Algorithmus willkürlich ein beliebiges Anfangsmatching zu erzeugen,wobei nacheinander für jeden Knoten des Graphen und den Zielknoten seiner ausgehenden und eingehenden Kanten geprüft wird,ob sie schon Anfangs-und Endknoten einer Matchingkante sind.Ansonsten wird die Kante als Matchingkante markiert.

Die Datenstruktur:

```
TMatchknoten = class(TInhaltsknoten)
  private
    Matchkante_:Integer;  {Durch die Verwendung von Datenfelder vom Typ Integer statt Zeigern}
    VorigeKante_:Integer;      {können Instanzen dieses Typs besser gespeichert werden.}
    procedure SetzeMatchkante(Ka:TInhaltskante);
    function WelcheMatchkante:TInhaltskante;
    procedure SetzeVorigeKante(Ka:TInhaltskante);
    function WelcheVorigeKante:TInhaltskante;
    procedure SetzeMindex(M:Integer);
    function WelcherMindex:Integer;
    procedure SetzeVindex(V:Integer);
    function WelcherVindex:Integer;
  public
    constructor Create;
    property Matchkante:TInhaltskante read WelcheMatchkante write SetzeMatchkante;
    property VorigeKante:TInhaltskante read WelchevorigeKante write SetzevorigeKante;
    property Matchkantenindex:Integer read WelcherMindex write SetzeMindex;
    property VorigerKantenindex:Integer read WelcherVindex write SetzeVindex;
    function Wertlisteschreiben:TStringlist;override;
    procedure Wertlistelesen;override;
    function Knotenistexponiert:Boolean;
  end;
```

TMatchknoten leitet sich durch Vererbung von TInhaltsknoten ab und enthält die zusätzlichen Datenfelder Matchkante_ und VorigeKante_,worin jeweils der Index der zugeordneten Matchkante bzw. der Index der vorigen zum Knoten als Zielknoten führenden im (evtl. versuchsweise) erweiternden Weg befindlichen Kante der Kantenliste des Graphen gespeichert wird.Durch die Propertys Matchkante bzw. VorigeKante kann dann auf die zu einem Knoten gehörende Matchkante bzw. die vorige Kante des Knotens (bezüglich des erweiternden Weges) zugegriffen werden.

Der Quellcode:

```
procedure TMatchknoten.SetzeMatchkante(Ka:TInhaltskante);
begin
  if Ka=nil
  then
    Matchkante_:=-1
  else
    begin
      if (Graph<>nil) and (not Graph.Kantenliste.Leer)
      then
        Matchkante_:=Graph.Kantenliste.Position(Ka)
      else
        Matchkante_:=-1;
    end;
end;


function TMatchknoten.WelcheMatchkante:TInhaltskante;
begin
  if (Graph<>nil) and (not Graph.Kantenliste.Leer)
       and (Matchkante_>=0)
```

```
       and (Matchkante_<=Graph.Kantenliste.Anzahl-1)
  then
    WelcheMatchkante:=TInhaltskante(Graph.Kantenliste.Kante(Matchkante_))
  else
    WelcheMatchkante:=nil;
end;

procedure TMatchknoten.SetzeVorigeKante(Ka:TInhaltskante);
begin
  if Ka=nil
  then
    VorigeKante_:=-1
  else
    begin
      if (Graph<>nil) and (not Graph.Kantenliste.Leer)
      then
        Vorigekante_:=Graph.Kantenliste.Position(Ka)
      else
        Vorigekante_:=-1;
    end;
end;

function TMatchknoten.WelcheVorigeKante:TInhaltskante;
begin
  if (Graph<>nil) and (not Graph.Kantenliste.leer)
        and (Vorigekante_>=0)
        and (Vorigekante_<=Graph.Kantenliste.Anzahl-1)
  then
    WelcheVorigeKante:=TInhaltskante(Graph.Kantenliste.Kante(Vorigekante_))
  else
    WelcheVorigeKante:=nil;
end;
```

Die Propertys Matchkantenindex bzw. VorigeKantenindex beziehen sich jeweils direkt auf die Index-Position in der Kantenliste.Auf die Datenfelder Matchkante_ bzw. VorigeKante_ wird mittels der Property Matchkante bzw. VorigeKante vom Typ TInhaltskante zugegriffen,indem jeweils zur Kante der Kantenindex in der Kantenliste des Graphen bzw. zum Kantenindex die Kante bestimmt wird.

```
function TMatchknoten.Knotenistexponiert:Boolean;
begin
  Knotenistexponiert:=(MatchKante=nil);
end;
```

Die Functions(-Methode) Knotenistexponiert bestimmt,ob ein Knoten ein exponierter Knoten ist,indem sie überprüft,ob die Property TMatchkante nil ist.

```
TMatchgraph = class(TInhaltsgraph)
    constructor Create;
    procedure InitialisierealleKnoten;
    procedure AlleKnotenSetzeVorigeKanteaufNil;
    procedure ErzeugeAnfangsMatching;
    function  AnzahlexponierteKnoten:Integer;
    procedure VergroessereMatching(G:TInhaltsgraph);
    procedure BestimmeMaximalesMatching(Flaeche:TCanvas;
      Ausgabe:TLabel;G:TInhaltsgraph);
    procedure ErzeugeListe(Var SListe:TStringlist);
  end;
```

TMatchgraph leitet sich durch Vererbung von TInhaltsgraph ab.Im folgenden werden die Methoden des Objekttyps erläutert.

```
constructor TMatchgraph.Create;
begin
  inherited Create;
  SetzeKnotenclass(TMatchknoten);
  SetzeKantenclass(TInhaltskante);
end;
```

Der Constructor setzt die Datentypen für die Knoten und die Kanten des Graphen auf TMatchknoten und TInhaltskante.

```
procedure TMatchgraph.InitialisierealleKnoten;
var  Index:Integer;
begin
  if not Leer then
```

```
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      TMatchknoten(Knotenliste.Knoten(Index)).Matchkante:=nil;
      TMatchknoten(Knotenliste.Knoten(Index)).VorigeKante:=nil;
    end;
end;
```

Die Methode InitialisierealleKnoten setzt bei allen Knoten des Graphen die Propertys Matchkante und VorigeKante auf nil.

```
procedure TMatchgraph.AlleKnotenSetzeVorigeKanteaufNil;
var  Index:Integer;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      TMatchknoten(Knotenliste.Knoten(Index)).VorigeKante:=nil;
end;
```

Die Methode AlleKnotenSetzeVorigeKanteaufNil setzt bei allen Knoten des Graphen die Property VorigeKante auf nil.Sie wird bei jedem Start der Suche eines erweiternden Weges neu gebraucht.

```
procedure TMatchgraph.ErzeugeAnfangsMatching;
var Index1:Integer;

  procedure SucheMatchkante(Kno1:TMatchknoten);
  var Index2:Integer;
      Kno2:TMatchknoten;
      Ka,Kant:TInhaltskante;
  begin
    if not Kno1.AusgehendeKantenliste.Leer then
      for Index2:=0 to Kno1.AusgehendeKantenliste.Anzahl-1 do
      begin
        Ka:=TInhaltskante(Kno1.AusgehendeKantenliste.Kante(Index2));
        Kno2:=TMatchknoten(Ka.Zielknoten(Kno1));
        if (Kno2.Matchkante=nil) and (Kno1.Matchkante=nil) and                    1)
             (not Ka.KanteistSchlinge) then
        begin
          Kno1.Matchkante:=Ka;                                                     2)
          Kno2.Matchkante:=Ka;
          Ka.Wert:='M';
          Ka.Farbe:=clred;
          Ka.Stil:=psdot;
        end;
      end;
      if not Kno1.EingehendeKantenliste.Leer then
        for Index2:=0 to Kno1.EingehendeKantenliste.Anzahl-1 do
        begin
          Ka:=TInhaltskante(Kno1.EingehendeKantenliste.Kante(index2));
          Kno2:=TMatchknoten(Ka.Zielknoten(Kno1));
          if (Kno2.Matchkante=nil) and (Kno1.Matchkante=nil) and                  3)
            (not Ka.KanteistSchlinge) then
          begin
            Kno1.Matchkante:=Ka;                                                   4)
            Kno2.Matchkante:=Ka;
            Ka.Wert:='M';
            Ka.Farbe:=clred;
            Ka.Stil:=psdot;
          end;
        end;
  end;

begin
  if not Leer then
    for Index1:=0 to Knotenliste.Anzahl-1 do
      SucheMatchkante(TMatchknoten(Knotenliste.Knoten(Index1)));
end;
```

Die Methode ErzeugeAnfangsMatching erzeugt ein zufälliges Anfangsmatching des Graphen,indem für jeden Knoten des Graphen die Procedure Suche-Matchkante aufgerufen wird.Bei 1) und 3) wird für den Zielknoten jeder aus-gehenden und eingehenden Kante des Knotens überprüft,ob die Property Matchkante des Anfangs-und Endknoten dieser Kante nil ist.Wenn dies der Fall ist,wird diese Property dann bei 2) und 4) auf die Kante gesetzt und diese damit zur Matchkante bestimmt.

```
function TMatchgraph.AnzahlexponierteKnoten:Integer;
var Zaehler,Index:Integer;
begin
```

```
  Zaehler:=0;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      if TMatchknoten(Knotenliste.Knoten(Index)).Matchkante=nil
      then
        Zaehler:=Zaehler+1;
  AnzahlexponierteKnoten:=Zaehler;
end;
```

Die Function AnzahlexponierteKnoten bestimmt die Anzahl der exponierten Knoten des Graphen,indem sie die Anzahl der Knoten mit der Property Matchkante gleich nil zählt.

```
procedure TMatchgraph.VergroessereMatching(G:TInhaltsgraph);
var Index:Integer;

  procedure FaerbeKanteum(Ka:TInhaltskante);
  begin
    if Ka.Wert='L    '                                                          5)
    then
    begin
      Ka.Wert:=TInhaltskante(G.Kantenliste.Kante(Kantenliste.Position(Ka))).Wert;
      Ka.Farbe:=clblack;
      Ka.Stil:=pssolid;
    end;
    if Ka.Wert='A    '                                                          6)
    then
    begin
      Ka.Wert:='M';
      TMatchknoten(Ka.Anfangsknoten).Matchkante:=Ka;
      TMatchknoten(Ka.Endknoten).Matchkante:=Ka;
      Ka.Farbe:=clred;
      Ka.Stil:=psdot;
    end;
  end;

begin
  if not Kantenliste.leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
      FaerbeKanteum(TInhaltskante(Kantenliste.Kante(Index)));
end;
```

Die Methode VergroessereMatching „färbt“ die Kanten eines Graphen um,indem sie den Wert solcher Kanten mit dem Wert L auf den ursprünglichen im (Ausgangs-)Graphen G gespeicherten Wert zurücksetzt (5) und den Wert solcher Kanten mit dem Wert A durch den Wert M ersetzt(6).Außerdem werden die Kanten mit der entsprechenden Zeichenfarbe und dem entsprechenden Stil versehen.

```
procedure TMatchgraph.BestimmeMaximalesMatching(Flaeche:TCanvas;
  Ausgabe:TLabel;G:TInhaltsgraph);
var Index:Integer;
    ErweiternderWegfuerAlleKnotengefunden,ErweiternderWeggefunden:Boolean;
    Startknoten:TMatchknoten;


  procedure SucheerweiterndenWeg(X:TMatchknoten;Ka:TInhaltskante);
  label Fertig,NaechsteKante;
  var Y,Z:TMatchknoten;
      Index3,Index4:Integer;

  begin
    if not ErweiternderWeggefunden
    then
    begin
      Y:=TMatchknoten(Ka.Zielknoten(X));                                       18)
      if Ka.KanteistSchlinge  then goto NaechsteKante;
      if (Y.VorigeKante<>nil)or (Y=Startknoten)  then goto NaechsteKante;
      if  (Y.VorigeKante=nil) and (not Ka.KanteistSchlinge) and (Y<>Startknoten) then   19)
      begin
        if (Y.Matchkante<> nil) then                                           20)
        begin
          Ka .Wert:='A    ';                                                   21)
          Z:=TMatchknoten(Y.MatchKante.Zielknoten(Y));                         22)
          Y.VorigeKante:=Ka;                                                   23)
          Y.MatchKante.Wert:='L    ';                                          24)
          Z.VorigeKante:=Y.MatchKante;                                         25)
          if not Z.AusgehendeKantenListe.Leer then
            for index3:=0 to Z.AusgehendeKantenliste.Anzahl-1 do
            begin
              SucheerweiterndenWeg(Z,TInhaltskante(Z.AusgehendeKantenliste.Kante(Index3)));
```

```
                                                                              26)
            if ErweiternderWeggefunden then goto Fertig;
          end;
          if not Z.EingehendeKantenliste.Leer then
            for Index4:=0 to Z.EingehendeKantenliste.Anzahl-1 do
            begin
              SucheerweiterndenWeg(Z,TInhaltskante(Z.EingehendeKantenliste.Kante(Index4)));
                                                                              27)
              if ErweiternderWeggefunden then goto Fertig;
            end;
            Fertig:
      end
      else
      begin
        Ka.Wert:='A    ';                                                     28)
        ErweiternderWeggefunden:=true;                                        29)
        goto NaechsteKante;                                                   30)
      end;
    end;
    if not ErweiternderWeggefunden then
    begin
      Y.MatchKante.Wert:='M';                                                 31)
      Ka.Wert:=g.Kantenliste.Kante(Kantenliste.Position(Ka)).Wert;            32)
      Z.VorigeKante:=nil;
      Y.VorigeKante:=nil;
    end;
  end;
  NaechsteKante:
end;

procedure SucheerweiterndenWegvonKnoten(Knot:TMatchknoten);
label Ende;
var AnzKanten:Integer;
    Index1,Index2:Integer;
begin
  ErweiternderWeggefunden:=false;
  if Knot.Knotenistexponiert then                                             11)
  begin
    Startknoten:=Knot;
    AlleKnotensetzevorigeKanteaufNil;                                         12)
    if not Knot.AusgehendeKantenliste.Leer then
      for Index1:=0 to Knot.AusgehendeKantenliste.Anzahl-1 do
      begin
        SucheerweiterndenWeg(Knot,TInhaltskante(Knot.AusgehendeKantenliste.Kante(index1)));
                                                                              13)
        if ErweiternderWeggefunden then goto Ende;
      end;
    if not Knot.EingehendeKantenliste.Leer then
      for Index2:=0 to Knot.EingehendeKantenliste.Anzahl-1 do
      begin
        SucheerweiterndenWeg(Knot,TInhaltskante(Knot.EingehendeKantenliste.Kante(Index2)));
                                                                              14)
        if ErweiternderWeggefunden then goto Ende;
      end;
    Ende:
  end;
  if ErweiternderWeggefunden then                                             15)
    VergroessereMatching(G);                                                  16)
  if ErweiternderWeggefunden then ErweiternderWegfuerAlleKnotengefunden:=false;      17)
end;


begin
  repeat                                                                       7)
    ErweiternderWegfuerAlleKnotengefunden:=true;                               8)
    if AnzahlexponierteKnoten>1 then
      if not Leer then
        for Index:=0 to Knotenliste.Anzahl-1 do                                9)
          SucheerweiterndenWegvonKnoten(TMatchknoten(Knotenliste.Knoten(Index)));
  until ( ErweiternderWegfueralleKnotengefunden) or (AnzahlexponierteKnoten<=1);      10)
end;
```

Die Methode BestimmeMaximalesMatching erzeugt ein maximales Matching des Graphen,indem solange nach erweiternden Wegen im Graphen von exponierten Knoten aus gesucht und dann umgefärbt wird,bis keine exponierten Knoten mehr vorhanden sind oder kein erweiternder Weg mehr gefunden wurde.

Dazu wird eine Repeat-Schleife 7) solange durchlaufen,bis es entweder keine exponierten Knoten mehr gibt,oder aber die Boolesche Variable Erweiternder-WegfueralleKnotengefunden,die zuvor auf false gesetzt wurde (8),true

wird(10).Dazu wird bei 9) für jeden Knoten Knot des Graphen die Procedure SucheerweiterndenWegvonKnoten aufgerufen,in der die Variable Erweiternder-WegfueralleKnotengefunden verändert werden kann.Der Knoten Knot ist der Startknoten des erweiternden Weges.

Die Hauptanweisungen dieser Procedure werden nur ausgeführt,wenn der Knoten Knot ein exponierter Knoten ist (11).In diesem Fall wird die Property Vori-geKante für alle Knoten zunächst auf nil gesetzt (12).

Anschließend wird bei 13) und 14) mit den ausgehenden und eingehenden Kan-ten des Knotens Knot und dem Knoten Knot selber jeweils die Procedure Su-cheerweiterndenWeg aufgerufen,die -wenn möglich- einen erweiternder Weg vom Zielknoten dieser Kanten aus bestimmt.Wenn ein erweiternder Weg gefunden wird (15),wird die Methode VergroessereMatching aufgerufen (16) und die Va-riable ErweiternderWegfueralleKnotengefunden wird auf false gesetzt (17),weil bei diesem Durchlauf noch ein erweiternder Weg gefunden wurde.

Die rekursive Backtracking-Procedure SucheerweiterndenWeg hat als über-gebende Parameter den Knoten Knot,der unter dem Parameternamen X gespei-chert wird,sowie die von Knot=X ausgehende ausgewählte Kante Ka.Wenn bisher kein erweiternder Weg gefunden wurde,wird bei 18) der Zielknoten der über-gebenden Kante Ka mit Y bezeichnet.

Wenn die Property VorigeKante des Knotens Y nicht nil ist,und Y auch nicht der Startknoten des erweiternden Weges ist (19),als auch die Property Matchkante von Y ungleich nil ist (20),ist die Kante Ka eine Anwärterkante für eine Matchkante,und als ihr Wert wird der Buchstabe A gespeichert (21).

Anschließend wird der Zielknoten der Matchkante von Y mit Z bezeichnet (22).Die Property VorigeKante von Y wird auf Ka gesetzt (23),der Wert der Matchkante von Y auf den Buchstaben L,weil diese Kante ihren Status als Matchkante beim Umfärben verliert (24),und als VorigeKante von Z wird die Matchkante von Y gespeichert (25).

Anschließend wird die nächste Rekursionsebene bei (26) und (27) der Methode SucheerweiterndenWeg mit dem Knoten Z und entweder einer von Z ausgehenden oder einlaufenden Kante als Parameter aufgerufen.

Abbruchkriterium der Rekursion ist,dass der Knoten Y keine Matchkante be-sitzt (20).Dann wird als Wert der Kante Ka der Buchstabe A für Anwärter Matchkante beim Umfärben gespeichert,die Boolesche Variable Erweiternder-Weggefunden wird auf true gesetzt (29),und es wird an das Ende der Methode zwecks Aufruf der Procedure mit einer neuen Kante vom bisherigen Startkno-ten aus verzweigt (30).

Bei der Rückkehr aus der vorigen Rekursionsebene werden,wenn kein erwei-ternder Weg gefunden wurde,alle bei den Kanten und Knoten bei 21),23),24) und 25) gesetzten Werte wieder rückgängig gemacht:

Bei 31) wird der Wert der Matchingkante von Y auf M gesetzt,bei 32) wird der ursprüngliche Inhalt wieder als Wert der Kante Ka gespeichert und bei 33) und 34) werden die Propertys VorigeKante von Z und Y wieder auf nil ge-setzt.

```
procedure TMatchgraph.ErzeugeListe(var SListe:TStringList);
var Index:Integer;
    Ka:TInhaltskante;
begin
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
    begin
      Ka:=TInhaltskante(Kantenliste.Kante(Index));
      if Ka.Wert='M' then SListe.Add('Kante: '+Ka.Anfangsknoten.Wert+'-'+Ka.Endknoten.Wert);
    end;
  if AnzahlexponierteKnoten=0 then SListe.Add('Das Matching ist perfekt!');
end;
```

Die Methode ErzeugeListe erzeugt eine Stringliste SListe,deren Strings aus den Anfangs-und Endknoten der Matchingkanten des Graphen bestehen,und gibt sie als Referenzparameter zurück.

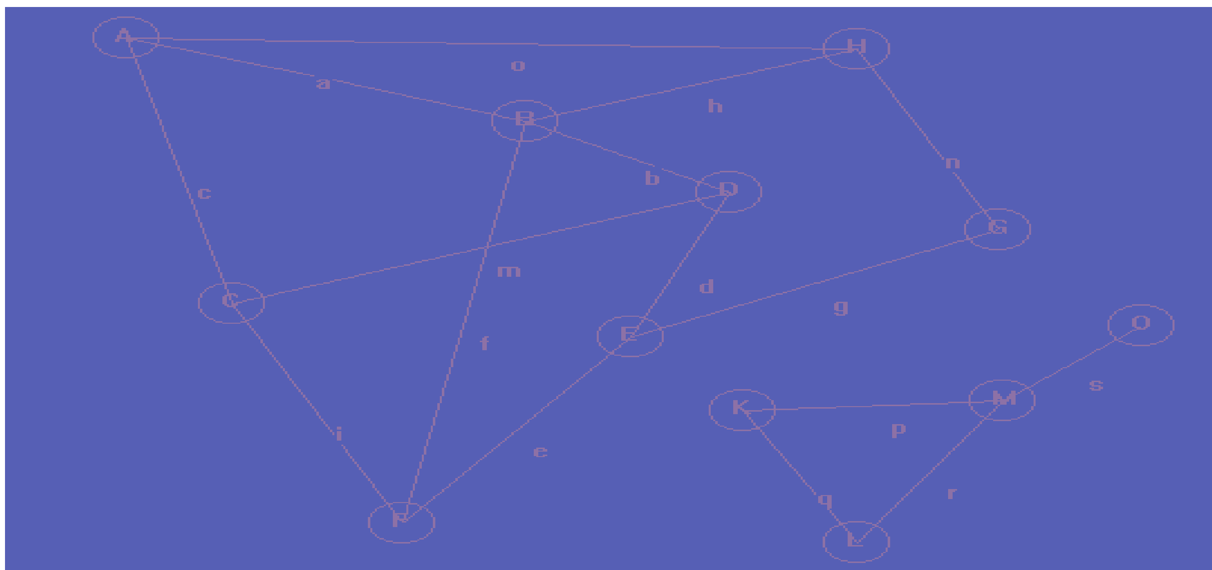
Aufgabe C X.7:

Bestimme im folgenden **nicht paaren** Graph ein maximales Matching nach dem Algorithmus Suche erweiternden Weg.

Ist das Matching perfekt?

Die Knoten wurden der alphabetischen Reihenfolge nach erzeugt,die Reihenfolge der Kanten bei einem Knoten entspricht der alphabetischen Ordnung.

(Lösung manuell,mit Hilfe des selbsterstellten Quellcodes in der Entwicklungsumgebung oder mit Hilfe des Programms Knotengraph/Menü Anwendungen/Maximales Matching,je nach Konzeption)



G056.gra

Hinweis für die manuelle Lösung:

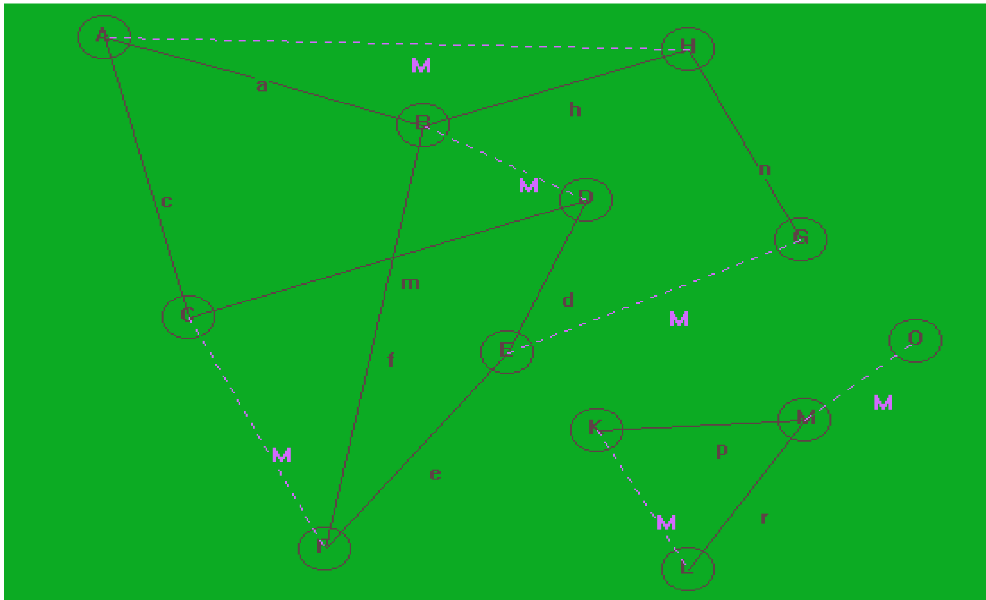
Untersuche die Erweiternden Wege jeweils der Reihe nach von den Knoten A,C,E,G,K und L aus.

Lösung:

Kanten des maximalen Matchings:

Kante: B-D
Kante: E-G
Kante: C-F
Kante: H-A
Kante: K-L
Kante: M-O

Das Matching ist perfekt!



G056.gra

C XI Gleichungssystem

Die Lösung von linearen Gleichungssystemen ist Unterrichtsinhalt des Mathematikunterrichts der Sekundarstufe I und II,wobei in der Sekundarstufe II die Lösung mit Hilfe des Gaußverfahrens sowie die Darstellung der Gleichungssysteme mit Hilfe von Matrizen als auch die Struktur der Lösungsmenge im Vordergrund steht.Die Umformungen beruhen auf Äquivalenzumformungen,bei denen die Lösungsmenge erhalten bleibt.Eine Methode,die Äquivalenzumformungen anschaulich mittels eines Graphen darzustellen,bietet der Algorithmus von Mason,der im Grunde ein Einsetzungsverfahren ist.Den Knoten eines Graphen werden dabei die Variablen des Gleichungssystem zugeordnet.Die Kanteninhalte stellen dabei die Koeffizienten des Gleichungssystem dar.Die Elimination einer Variablen geschieht anschaulich durch Löschen eines Knotens,wobei die Werte der Kanten so umgeformt werden,dass das reduzierte Gleichungssystem dieselbe Lösungsmenge bezüglich der um 1 verminderten Variablen hat.Auf diese Weise kann der Vorgang der Äquivalenzumformung graphisch verfolgt werden und wird dadurch verständlicher.

Diese graphische Lösung bildet die Grundlage und den Ausgangspunkt für das Reduzieren von Signalflussgraphen (mittels objektorientierter Vererbung der Datenstruktur) und führt zur Lösung der Übergangswahrscheinlichkeit von Markovketten auf der Grundlage der Mittelwertregeln der Wahrscheinlichkeitsrechnung (vgl. das entsprechende folgende Kapitel Graph reduzieren).

Gegeben sei eine nxn Matrix A ($n \in \mathbb{N}$) aus reellen Zahlen und das lineare Gleichungssystem $A\vec{x} = \vec{b}$ mit den Vektoren $\vec{x}$ und $\vec{b}$,die aus n reellen Komponenten bestehen.Der Vektor $\vec{x}$ ist der Lösungsvektor.

Das Gleichungssystem lautet anders geschrieben:

$$
\begin{array}{l}
a_{11}x_1+ \;\dots\; + a_{1i}x_i+ \;\dots\; + a_{1n}x_n = b_1 \\
a_{21}x_1+ \;\dots\; + a_{2i}x_i+ \;\dots\; + a_{2n}x_n = b_2 \\
\dots\dots\dots\dots\dots\dots\dots\dots \\
a_{i1}x_1+ \;\dots\; + a_{ii}x_i+ \;\dots\; + a_{in}x_n = b_i \\
\dots\dots\dots\dots\dots\dots\dots\dots \\
a_{k1}x_1+ \;\dots\; + a_{ki}x_i+ \;\dots\; + a_{kn}x_n = b_k \\
\dots\dots\dots\dots\dots\dots\dots\dots \\
a_{n1}x_1+ \;\dots\; + a_{ni}x_i+ \;\dots\; + a_{nn}x_n = b_n
\end{array}
$$

Wenn $a_{ii} \neq 0$ ist,kann man die i.te Zeile nach x_i auflösen und in die k.-te Zeile einsetzen:

$$x_i = \frac{1}{a_{ii}}(b_i - a_{i1}x_1 - \dots - a_{i,i-1}x_{i-1} - a_{i,i+1}x_{i+1} - \dots - a_{ik}x_k - \dots - a_{in}x_n)$$

$$a_{k1}x_1 + \dots + a_{k,i-1}x_{i-1} + a_{ki}\frac{1}{a_{ii}}(b_i - a_{i1}x_1 - \dots - a_{i,i-1}x_{i-1} - \dots . a_{ik}x_k \dots - a_{in}x_n)$$
$$+ a_{k,i+1}x_i + \dots + a_{kk}x_k + \dots + a_{kn}x_n = b_k$$

Geordnet ergibt sich:

$$(a_{k1} - \frac{a_{i1}a_{ki}}{a_{ii}})x_1 + \dots + (a_{k,i-1} - \frac{a_{i,i-1}a_{ki}}{a_{ii}})x_{i-1+}(a_{ki+1} - \frac{a_{i,i+1}a_{ki}}{a_{ii}})x_{i+1}$$
$$+ \dots + (a_{kk} - \frac{a_{ik}a_{ki}}{a_{ii}})x_k + \dots + (a_{kn} - \frac{a_{in}a_{ki}}{a_{ii}})x_n = b_k - \frac{a_{ki}b_i}{a_{ii}}$$

Dies ergibt ein reduziertes Gleichungssystem mit n-1 Gleichungen und n-1 Variablen x_k.

In dem reduzierten Gleichungssystem gilt:

I) $b_k' = b_k - \frac{a_{ki} b_i}{a_{ii}}$

II) $a_{kl}' = a_{kl} - \frac{a_{il} a_{ki}}{a_{ii}}$

Die Gleichungen I) und II) sind die Umrechnungsgleichungen der Koeffizienten des neuen Systems aus den Koeffizienten des alten Systems.

(Die Gleichungen können natürlich auch etwas konkreter schulgerecht an Hand eines 3x3-Gleichungssystems mit allgemeinen Koeffizienten abgeleitet werden.)

Ein Gleichungssystem der obigen Art läßt nun folgendermaßen als Graph darstellen:

Der Graph enthält n Knoten.Die Knoteninhalte sind die Zahlenwerte b_1, b_2 ... b_n.Dadurch ist automatisch eine Knotennumerierung von 1 bis n vorgegeben.Es werden jetzt gerichtete Kanten von jedem Knoten i zu jedem Knoten j eingefügt.Als Inhalt dieser Kante wird der Koeffizient a_{ij} gespeichert.Zu jedem Knoten wird außerdem eine Schlinge in den Graph eingefügt.Beim Knoten mit der Nummer i wird dann als Inhalt der Schlinge der Koeffizient a_{ii} gespeichert.

Auf diese Weise ist jeder Knoten mit jedem Knoten durch eine Kante verbunden,und jeder Knoten besitzt genau eine Schlinge.

Aufgabe C XI.1 (Einstiegsproblem):

Gegeben ist das folgende lineare Gleichungssystem.Erstelle zu dem Gleichungssystem mit dem Programm Knotengraph einen Graph nach der oben genannten Vorschrift.Ermittle die Lösungen des Gleichungssystems per Hand.

$2x_1 + 5x_2 + 9x_3 = 39$
$3x_1 + 6x_2 + 4x_3 = 27$
$4x_1 + 7x_2 + 8x_3 = 42$

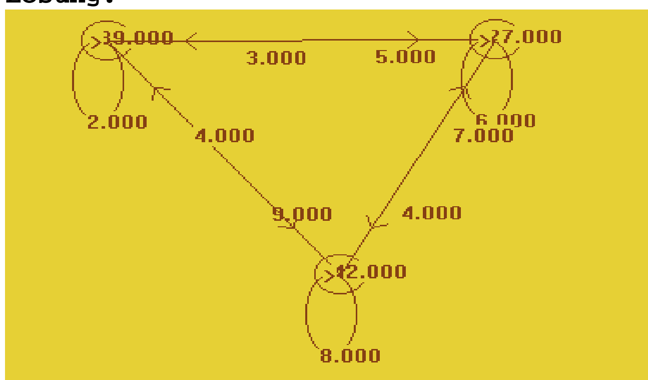
Rufe dann den Algorithmus des Menüs Anwendungen/Gleichungssystem des Programms Knotengraph auf,und lösche danach **nacheinander** alle Knoten des Graphen durch Anklicken in der Reihenfolge 3,2 und 1.Beobachte was passiert.

Wenn der letzte Knoten gelöscht ist,wird der ursprüngliche Graph wiederhergestellt.

Wiederhole das Löschen noch zwei weitere Male,so dass jedes Mal ein anderer Knoten als letzter Knoten des Graphen zu löschen ist.

Beobachtung und problemorientierte Erarbeitung:

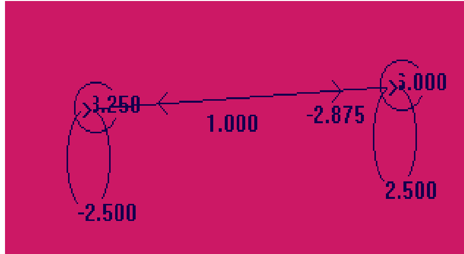
Lösung:



G057.gra

Die Werte der Kanten und Knoten verändern sich.Wenn der letzte Knoten gelöscht wird,wird jeweils die zugehörige Lösung x=1;2 oder 3 ermittelt.

Beispiel,wenn nacheinander Knoten 3,2 und 1 gelöscht werden:



G057.gra

Zugehöriges Gleichungssystem:

$-2.5x_1-2.875x_2=-3.25$
$1x_1 \quad +2.5x_2 \;= 5$

Lösung:$x_1=1,x_2=2$

G057.gra

Zugehöriges Gleichungssystem:

$-1.35\ x_1= -1.35$

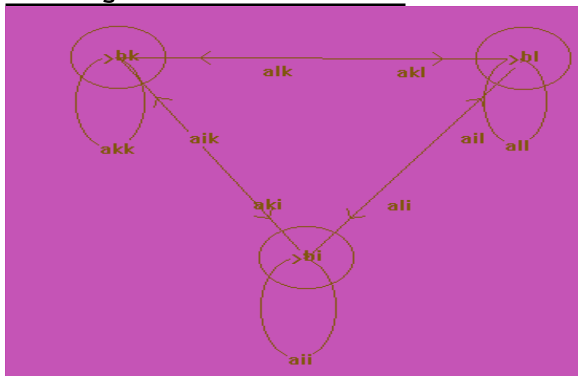
Lösung: $x_1=1$

Die Lösung $x_1 = 1$ wird ausgegeben.

Durch sukzessives Löschen der Knoten des Graphen kann also jeweils die Lösung des Graphen (falls sie existiert) für den Knoten bestimmt werden,der als letzter zu löschen ist.Dabei wird das ursprüngliche Gleichungssystem jeweils äquivalent umgeformt und als Graph dargestellt,wobei jeweils eine Variable eliminiert wird.

Das Verfahren arbeitet nach dem Algorithmus von Mason zur Lösung eines linearen Gleichungssystems.

Der Algorithmus von Mason



G058.gra

Verbale Beschreibung:

1)Bestimme einen zu löschenden Knoten des Graphen z.B. den Knoten mit der Nummer i (Inhalt b_i) Kno=Kno1.

2)Ersetze die Inhalte b_k aller k.ten Knoten Kno2,zu denen ausgehende Kanten von dem i.ten Knoten führen durch den Term:

$$b_k' = b_k - \frac{a_{ki} b_i}{a_{ii}}.$$

Dabei ist a_{ki} der Inhalt der vom Knoten k zum Knoten i führenden Kante.b_i ist der Inhalt des zu löschenden Knotens.a_{ii} ist der Inhalt der Schlinge des zu löschenden i.ten Knotens.

3)Ersetze den Inhalt a_{kl} der Kante zwischen jedem k.ten und jedem l.ten Knoten für k und l ungleich i durch den Term:

$$a_{kl}' = a_{kl} - \frac{a_{il} a_{ki}}{a_{ii}}$$

Dabei ist a_{il} der Inhalt der Kante vom i.ten zum l.ten Knoten,a_{ki} der Inhalt der Kante vom k.ten zum i.ten Knoten und a_{ii} der Inhalt der Schlinge des i.ten Knotens.

4)Lösche den i.ten Knoten aus dem Graph.

5)Wiederhole das Verfahren und beginne wieder mit Schritt 1),solange die Knotenzahl des Graphen größer als 1 ist.

6)Wenn die Knotenzahl gleich 1 ist,bestimme die i-te Lösung

zu $x_i = -\frac{b_i}{a_{ii}}$ .Lösche den Knoten und stelle danach den ursprünglichen Graph wieder her.

Wenn bei Schritt 2),3) oder 6) irgendwann der Koeffizient a_{ii} Null sein sollte,breche den Algorithmus mit der Meldung ab,dass es entweder keine oder keine eindeutige Lösung gibt.

Die Datenstruktur:

```
TGleichungssystemknoten = class(TInhaltsknoten)
 private
    Nummer_:Integer;
    Ergebnis_:string;
    procedure SetzeKnotennummer(Nu:Integer);
    function WelcheKnotennummer:Integer;
    function WelchesErgebnis:string;
    procedure SetzeErgebnis(S:string);
 public
    constructor Create;
    property Nummer:Integer read WelcheKnotennummer write SetzeKnotennummer;
    property Ergebnis:string read WelchesErgebnis write SetzeErgebnis;
    function Wertlisteschreiben:TStringlist;override;
    procedure Wertlistelesen;override;
  end;
```

TGleichungssystemknoten leitet sich von TInhaltsknoten durch Vererbung ab und enthält die zusätzlichen Datenfelder Nummer_ , zur Nummerierung der Knoten und Ergebnis_,um ein Ergebnis bestehend aus der Lösung des Gleichungssystems für die Variable,die durch diesen Knoten dargestellt wird,sowie dem Variablennamen aufzunehmen.Auf diese Felder wird mittels geeigneter Propertys zugegriffen.

```
TGleichungssystemgraph = class(TInhaltsgraph)
  public
    constructor Create;
    procedure NummeriereKnoten;
    procedure ErgaenzeSchlingen;
    procedure ErgaenzeKanten;
    procedure EliminiereKnoten(var Kno:TKnoten;Flaeche:TCanvas;Ausgabe:TLabel;
      var Loesung:Extended;var Ende:Boolean;var G:TInhaltsgraph;var Oberflaeche:TForm);
  end;

  TGleichungssystemgraph = class(TInhaltsgraph)
    constructor Create;
    procedure NumeriereKnoten;
    procedure ErgaenzeSchlingen;
    procedure ErgaenzeKanten;
    procedure EliminiereKnoten(var kno:TKnoten;
    Flaeche:TCanvas;var sliste:Tstringlist;ausgabe:TLabel;
    var Loesung:Extended;var ende:Boolean;var g:TInhaltsgraph;var Oberflaeche:TForm);
  end;
```

TGleichungssystemgraph leitet sich von TInhaltsgraph ab und enthält keine neuen Datenfelder.

Der Quellcode:

Die Methoden von TGleichungssystemgraph werden im folgenden erläutert:

```
constructor TGleichungssystemgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TGLeichungssystemknoten;
  InhaltsKanteclass:=TInhaltskante;
end;
```

Der Constructor legt die Datentypen für die Knoten und die Kanten des Graphen als TGleichungssystemknoten und TInhaltskante fest.

```
procedure TGleichungssystemgraph.NumeriereKnoten;
var Index:Integer;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      TGleichungssystemknoten(Knotenliste.Knoten(Index)).Nummer:=Index+1;
end;
```

Die Methode NummeriereKnoten nummeriert die Knoten der Knotenliste des Graphen mittels der um den Index der Knotenliste vergrößert um 1 bestimmten Zahl.Diese Zahl wird jeweils der Property Nummer zugewiesen.

```
procedure TGleichungssystemgraph.ErgaenzeSchlingen;
var Index:Integer;
    Kno:TGleichungssystemknoten;
    Ka:TInhaltskante;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TGleichungssystemknoten(Knotenliste.Knoten(Index));
      if Kno.AnzahlSchlingen=0 then
      begin
        Ka:=TInhaltskante.Create;
        Ka.Weite:=30;
        Ka.Wert:='0';
        FuegeKanteein(Kno,Kno,true,Ka);
      end;
    end;
end;
```

Die Methode ErgänzeSchlinge ergänzt bei allen Knoten des Graphen,die noch keine Schlinge besitzen,Schlingen mit dem Wert Null.

```
procedure TGleichungssystemgraph.ErgaenzeKanten;
var Index1,Index2:Integer;
    Kno1,Kno2:TGleichungssystemknoten;
    Ka:Tinhaltskante;

begin
  if not Leer then
    for Index1:=0 to Knotenliste.Anzahl-1 do
```

```
    begin
      for Index2:=0 to Knotenliste.Anzahl-1 do
      begin
        Kno1:=TGleichungssystemknoten(Knotenliste.knoten(Index1));
        Kno2:=TGleichungssystemknoten(Knotenliste.knoten(Index2));
        if (Kno1<>Kno2) and (not KanteverbindetKnotenvonnach(Kno1,Kno2)) then
        begin
          Ka:=TInhaltskante.Create;
          Ka.Wert:='0';
          FuegeKanteein(Kno1,Kno2,true,Ka);
        end;
      end;
    end;
end;
```

Die Methode ErgaenzeKanten ergänzt zwischen zwei Knoten Kno1 und Kno2 der Knotenliste,zwischen denen bisher noch keine gerichtete Kante von Kno1 nach Kno2 existiert,eine gerichtete Kante von Kno1 nach Kno2 mit dem Wert Null.

```
procedure TGleichungssystemgraph.EliminiereKnoten(var Kno:TKnoten;Flaeche:TCanvas;
  Ausgabe:TLabel;var Loesung:Extended;var Ende:Boolean;
  var G:TInhaltsgraph;var Oberflaeche:TForm);
var Kno1,Kno2,Kno3:TGleichungssystemknoten;
    Ka:TInhaltskante;
    Akkk,Bkk,Aii,Akk,Aki,Bk,Aik,Ail,Bi,Akl:Extended;
    Index1,Index2:Integer;

  procedure Veraendereakl(Kant:TInhaltskante);
  begin
    if not Kant.KanteistSchlinge then                                          18)
      if Kant.Endknoten<>Kno1  then                                            19)
      begin
        Akl:=StringtoReal(Kant.Wert);                                          20)
        Kno3:=TGleichungssystemknoten(Kant.Endknoten);                         21)
        if ErsteKantevonKnotenzuKnoten(Kno1,Kno3)<>nil                         22)
        then
          Ail:=StringtoReal(TInhaltskante(ErsteKantevonKnotenzuKnoten(Kno1,Kno3)).Wert) 23)
        else
          Ail:=0;                                                              24)
        if Aii<>0 then Akl:=Akl-Ail*Aki/Aii;                                   25)
        Kant.Wert:=RealtoString(Akl);                                          26)
     end;
  end;

begin
  Ende:=false;
  Kno1:=TGleichungssystemknoten(Kno);                                          1)
  if AnzahlKnoten>1 then                                                       2)
  begin
    if not Kno1.EingehendeKantenliste.Leer then
      for Index1:=0 to Kno1.EingehendeKantenliste.Anzahl-1 do                  3)
      begin
        Loesung:=0;
        Aii:=0;
        Akk:=0;
        Bk:=0;
        Ail:=0;
        Aki:=0;
        Aik:=0;
        Bi:=0;
        Akl:=0;
        Ka:=TInhaltskante(Kno1.EingehendeKantenliste.Kante(Index1));
        Kno2:=TGleichungssystemknoten(Ka.Anfangsknoten);                       4)
        if  Kno2<>Kno1 then
        begin
          Aki:=StringtoReal(Ka.Wert);                                          5)
          Bi:=StringtoReal(Kno1.Wert);                                         6)
          Bk:=StringtoReal(Kno2.Wert);                                         7)
          if ErsteKantevonKnotenzuKnoten(Kno1,Kno2)<>nil then                  8)
            Aik:=StringtoReal(TInhaltskante(ErsteKantevonKnotenzuKnoten(Kno1,Kno2)).Wert);
          if ErsteSchlingezuKnoten(Kno1)<>nil then
            Aii:=StringtoReal(TInhaltskante(ErsteSchlingezuKnoten(Kno1)).Wert);         9)
          if Aii<>0 then                                                       10)
          begin
            if ErsteSchlingezuKnoten(Kno2)<>nil then
            begin
              Akk:=StringtoReal(TInhaltskante(ErsteSchlingezuKnoten(Kno2)).Wert);      12)
              Akk:=Akk-Aik*Aki/Aii;                                            13)
              TInhaltskante(ErsteSchlingezuKnoten(Kno2)).Wert:=RealtoString(Akk);      14)
             end;
            Bk:=Bk-Aki*Bi/Aii;                                                 15)
            Kno2.Wert:=RealtoString(Bk);                                       16)
            if not Kno2.AusgehendeKantenliste.Leer then
```

```
              for Index2:=0 to Kno2.AusgehendeKantenliste.Anzahl-1 do
                Veraendereakl(TInhaltskante(Kno2.AusgehendeKantenliste.Kante(Index2))); 17)
          end
          else
          begin
            Ende:=true;                                                         11)
            exit;
          end;
        end
      end;
      Knotenloeschen(Kno1);                                                     36)
      ZeichneGraph(Flaeche);
    end
    else
    begin
      Akkk:=0;                                                                  27)
      Bkk:=0;
      Kno1:=TGleichungssystemknoten(Anfangsknoten);                             28)
      Akkk:=StringtoReal(TInhaltskante(ErsteSchlingezuKnoten(Kno1)).Wert);      29)
      Bkk:=StringtoReal(Kno1.Wert);                                             30)
      if Akkk <>0 then                                                          31)
      begin
        Loesung:=Bkk/Akkk;                                                      32)
        Knotenloeschen(Kno1);                                                   35)
        Ausgabe.Caption:='';
        Ausgabe.Refresh;
        ZeichneGraph(Flaeche);
      end
     else
     begin
       Ende:=true;                                                              33)
       exit;                                                                    34)
     end;
  end;
end;
```

Der Methode EliminiereKnoten wird der zu löschende Knoten als Referenzparameter Kno übergeben.Der Knoten Kno wird bei 1) unter dem (Zusatz-)Namen Kno1 abgespeichert.

Wenn die Knotenanzahl des Graphen größer als 1 ist (2),werden dann folgende Schritte ausgeführt:

Bei 3) werden alle nach Kno1 einlaufenden Kanten Ka nacheinander ausgewählt,und es werden die Variablen Aii, Akk, Bk usw. mit Null initialisiert.Bei 4) wird der Anfangsknoten der Kante Ka als Kno2 gespeichert.

Vergleiche zu den folgenden Erläuterungen die obige Zeichnung bei der verbalen Beschreibung des Algorithmus.

Wenn die Kante Ka keine Schlinge ist,wird bei 5) der Wert der Kante (als Real-Zahl) in der Variablen Aki gespeichert,der Wert des Knotens Kno1 (als Real-Zahl) in der Variablen Bi und der Wert von Kno2 (als Real-Zahl) in der Variablen Bk.Der Wert der Rückkante zu Ka von Kno1 nach Kno2 wird,falls diese Kante existiert,(als Real-Zahl) in der Variablen Aik abgespeichert.Der Wert der Schlinge des Knotens Kno1 wird (als Real-Zahl) in der Variablen Aii abgespeichert.(je 6,7,8,9)

Wenn der Wert von Aii ungleich Null ist (10),werden die nachfolgenden Anweisungen (ab 12) weiter ausgeführt.

Sonst wird die Variable Ende (Referenzparameter der Methode) auf true gesetzt (11),und der Algorithmus (die Methode) wird beendet,weil es keine Lösung oder keine eindeutige Lösung des Gleichungssystems gibt.

Bei 12) wird der Wert der Schlinge des Knotens Kno2 (als Real-Zahl) in der Variablen Akk abgespeichert.Falls Aii ungleich Null ist,kann der Wert des Terms Akk-Aik*Aki/Aii berechnet werden (13) und ergibt nach der oben genannten II. Formel den neuen Wert für Akk,der bei 14) als Wert der Schlinge des Knoten Kno2 gespeichert wird.

Ebenfalls kann bei 15) der Term Bk-Aki*Bi/Aii berechnet werden,der nach der oben genannten I. Formel den neuen Wert für Bk ergibt und als neuer Knotenwert des Knotens Kno2 bei 16) gespeichert wird.

Damit sind die Operationen des obigen Schrittes 2) komplett durchgeführt und von den Operationen zu Schritt 3) ist schon der neue Wert für die Schlinge des k.ten Knotens a_{kk} berechnet und abgespeichert.

Es verbleibt die Berechnung des Wertes für A_{kl} für alle gerichteten Kanten vom k.ten zu allen l.ten Knoten (mit l ungleich i).

Dazu wird bei 17) für alle vom k.ten Knoten Kno2 ausgehenden Kanten (zu den l.ten Knoten) die Procedure Veraendereakl mit der Kante als Parameter aufgerufen.Wenn diese Kante Kant keine Schlinge ist (18) und auch nicht im Knoten Kno1 endet (19),wird der Inhalt der Kante (als Real-Zahl) in der Variablen Akl gespeichert (20),und der Endknoten der Kante wird mit Kno3 bezeichnet (21).

Wenn eine Kante von Kno1 nach Kno3 existiert (22),wird der Wert der (ersten dieser) Kante(n) (als Real-Zahl) in der Variablen Ail gespeichert (23) (Kante vom i.ten zum l.ten Knoten),andernfalls ist Ail gleich Null (24). Damit kann der Term Akl-Ail*Aki/Aii berechnet werden (25) (für Aii ungleich Null),und als neuer Wert für Akl gemäß der obigen II. Formel als Wert der Kante Kant gespeichert werden (26).

Wenn die Knotenanzahl des Graphen gleich 1 ist (vgl. 2),werden folgende Schritte ausgeführt:

Die Variablen Akkk und Bkk werden auf Null gesetzt (27).Der einzige Knoten des Graphen wird als Kno1 gespeichert (28),der Wert der Schlinge dieses Knotens wird (als Real-Zahl) in der Variablen Akkk und der Wert des Knotens Kno1 selber (als Real-Zahl) in der Variablen Bkk (29 und 30) gespeichert.Die Lösung für die Variable des Gleichungssystem,das zu diesem Knoten gehört,läßt sich dann als Quotient aus Bkk und Akkk (für Akkk ungleich Null (31))berechnen (32) und wird auf dem Referenzparameter Loesung gespeichert.

Wen Akkk gleich Null ist,wird der Referenzparameter Ende auf true gesetzt (33) und der Algorithmus abgebrochen (34) (die Methode verlassen),weil es keine Lösung oder keine eindeutige Lösung gibt.

In beiden Fällen (mehrere Knoten oder nur noch ein Knoten im Graph) wird schließlich der Knoten Kno1 aus dem Graph gelöscht (35 und 36).

Ähnlich wie bei dem Algorithmus endlicher Automat beschrieben wurde,wird ein Knoten jeweils durch Mausklick ausgewählt.Danach wird mit diesem Knoten die Methode EliminiereKnoten aufgerufen,bis sich keine Knoten mehr im Graph befinden.Der ursprüngliche Graph wird zwischengespeichert und zum Schluss wiederhergestellt (vgl. Menü TKnotenformular.GleichungssystemClick sowie TKnotenformular.GleichungssystemMousedown in der Unit UFormular).

Zuvor wurden die Methoden ErgaenzeKanten,ErgaenzeSchlingen und NummeriereKnoten aufgerufen.Nachdem alle Knoten eliminiert worden sind,wird die Lösung zusammen mit der Knotennumerierung als String in der Stringliste SListe für die Ausgabe im Ausgabefenster gespeichert.

Aufgabe C XI.2:

Bestimme die Lösungen der folgenden Gleichungssysteme nach dem obigen Algorithmus von Mason mit Hilfe eines geeigneten Graphen zunächst per Hand und Taschenrechner und dann mittels des Programms Knotengraph und dem Algorithmus des Menüs Anwendungen/Gleichungssystem oder mittels des selbst erstellten Quellcodes in der Entwicklungsumgebung (je nach Konzeption). Benutze den Demomodus,um den Ablauf des Algorithmus und um Zwischenergebnise zu verfolgen.

A)Gleichungssystem:

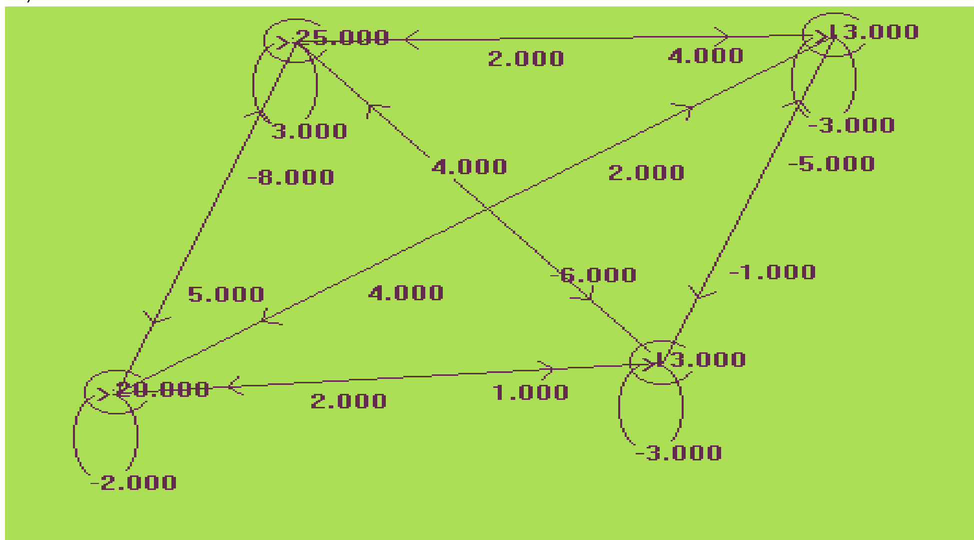
1) $3x_1+4x_2-6x_3+5x_4= -25$
2) $2x_1-3x_2- x_3+4x_4= 13$
3) $4x_1-5x_2-3x_3+2x_4= 13$
4) $-8x_1+2x_2+ x_3-2x_4= -20$

B)Gleichungssystem:

1) $3x_1+4x_2-6x_3+5x_4-4x_5 = -5$
2) $2x_1-3x_2- x_3+4x_4+2x_5 = 3$
3) $4x_1-5x_2-3x_3+2x_4-3x_5 = 28$
4) $-8x_1+2x_2+ x_3-2x_4-3x_5 = -5$
5) $6x_2-2x_2- x_3+3x_4-4x_5= 37$

Lösungen:

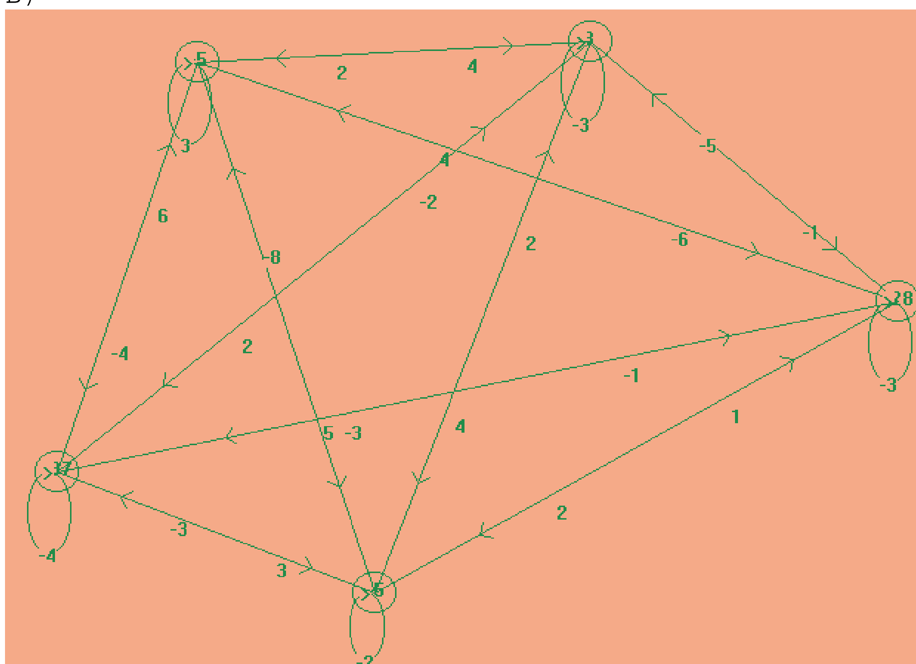
A)



G059.gra

$x_1=2$ $x_2=-3$ $x_3=4$ $x_4=1$

B)



G060.gra

$x_1=2$ $x_2= -3$ $x_3=4$ $x_4=1$ $x_5= -5$

Die folgende Aufgabe,insbesondere der Zusatz dient zur Überleitung auf das Thema Signalflußgraphen/Markovketten,das auf dem Algorithmus Geichungssystem von Mason aufbaut,und im nächsten Kapitel besprochen wird:

Aufgabe C XI.3:

Ermittle die Lösung mit dem Menü Anwendungen/Gleichungssystem von Knotengraph (oder mit dem selbsterstellten Quelltext in der Entwicklungsumgebung,je nach Konzeption),und erzeuge zuvor einen geeigneten Graphen:

Zur Herstellung eines Produkts werden 4 Maschinen A,B,C und D benötigt.

Maschine A benötigt 8 Stunden vermindert um den 5.Teil der Zeit,die die Maschine B benötigt.

Maschine B benötigt das Doppelte der Zeit von Maschine A minus der eigenen Zeit plus der Zeit von Maschine D.

Maschine C benötigt das Doppelte der Maschine B vermindert um die Zeit von Maschine D.

Maschine D benötigt das Doppelte von Maschine A vermindert um die Hälfte der Zeit von Maschine D.

Wie lang muß jede der Maschinen laufen?

Lösung:

Es ergeben sich folgende Bedingungsgleichungen für die Zeiten der Maschinen:

1) $t_A = 8 - \frac{1}{5}t_B$

2) $t_B = 2t_A - t_B + t_D$

3) $t_C = 2t_B - t_D$

4) $t_D = 2t_A - \frac{1}{2}t_D$

In Normalform lautet das Gleichungssystem:

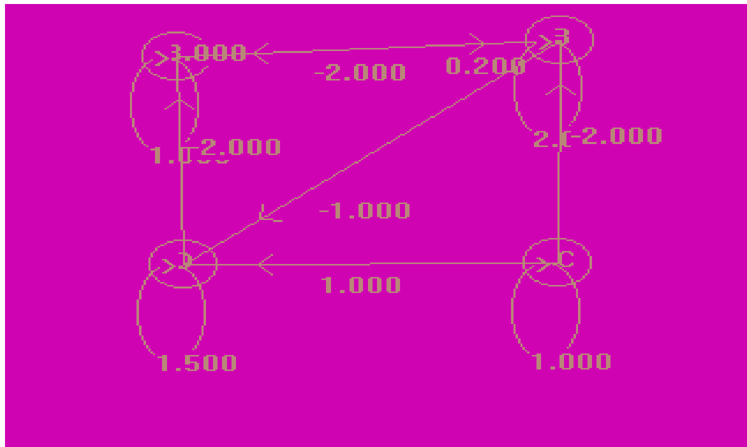
I) $t_A + \frac{1}{5}t_B = 8$

II) $-2t_A+2t_B-t_D = 0$

III) $-2t_B+t_C+t_D = 0$

IV) $-2t_A + \frac{3}{2} t_D = 0$

Der zugehörige Graph sieht folgendermaßen aus:



G061.gra

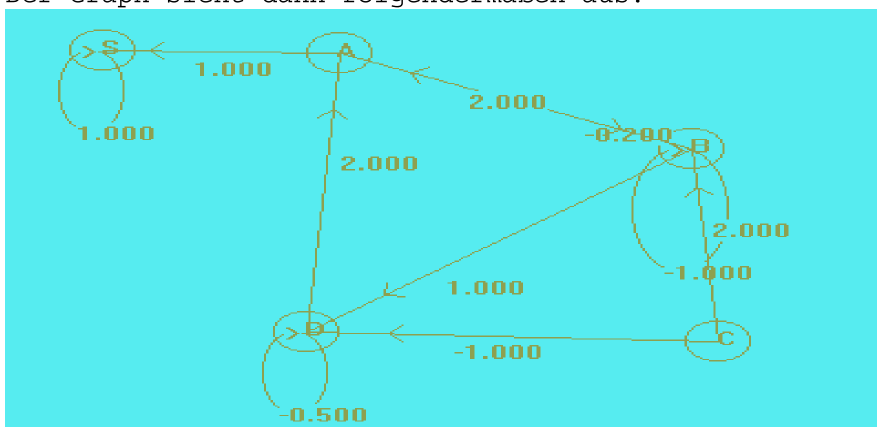
Als Lösung ergibt sich mit dem Algorithmus des Menüs Anwendungen/Gleichungsystem des Programms Knotengraph (oder dem selbst erstellten Quellcode in der Entwicklungsumgebung):

t_A= 6h, t_B=10h, t_C=12h, t_D=8h.

Zusatzaufgabe:

Wünschenswert wäre es jetzt,wenn man die Gleichungen 1) bis 4) nicht erst in die Normalform des Gleichungssystem I) bis IV) transformieren müßte,sondern direkt einen Graph mit den Gleichungen 1) bis 4) erstellen könnte.

Der Graph sieht dann folgendermaßen aus:



G062.gra

Der Knoten S ist ein zusätzlich zugefügter Startknoten.Es gilt

$t_A=t_S-0.2\ t_B$,wobei in S die Zahl 8 als Wert durch Eingabe von der Tastatur gespeichert wird.

Erzeuge ebenfalls diesen Graph mit Knotengraph.

Benutze den Algorithmus Anwendungen/Graph reduzieren im Programm Knotengraph (Beschreibung im nächsten Kapitel),und lösche mehrmals nacheinander alle Knoten des Graphen,so dass jeweils immer ein anderer Knoten als letzter zu löschen ist.Gib bei der entsprechenden Abfrage an,dass der Graph nicht als Markovkette (sondern als Signalflussgraph) aufzufassen sein soll,und gib für den (Signal-)Fluss 8 ein.

Die schon oben ermittelten Lösungen werden nach dem Löschen des jeweils letzten Knoten wiederum ausgegeben.

Diese Art der Darstellung des Graphen legt es nahe,die Zeit als Signalfluss oder Übergangsfluss aufzufassen,der durch den Graphen geschickt wird.Die Verteilung des (Signal/Übergangs-) Flusses von Knoten zu Knoten wird durch die obigen Gleichungen beschrieben.

(Die Pfeilrichtungen der Kanten müßten eigentlich für die Interpretation als (Signal/Übergangs-)Fluss umgekehrt gewählt werden,werden aber beibehalten,um den Algorithmus auf den Algorithmus Gleichungssystem zurückführen zu können.Der (Signal/Übergangs-)Fluss verläuft hier also entgegen der Pfeilrichtung.)

Werden die Übergangswahrscheinlichkeiten bei einer absorbierenden Markovkette als Signalfluss oder Übergangsfluss aufgefaßt,führt diese Methode zur Bestimmung der Ergebnisse (Übergangswahrscheinlichkeiten) bei Markovketten.Weiteres dazu im zweiten Teil des nächsten Kapitel C XII.

Allgemein ist ein Signalflussgraph oder Übergangsgraph ein Graph,der die algebraischen Beziehungen zwischen Variablen,die mit den Knoten des Graphen identifiziert werden,darstellt.Ein Signalflussgraph oder Übergangsgraph ist nicht identisch mit dem Typ Flussgraph,wie er in den Kapiteln C XIII, C XIV und C XV benutzt wird (z.B. gilt die Flussbedingung für innere Knoten nicht).

C XII Markovketten/Reduktion von Graphen

Ein Unterrichtsgebiet der Schulmathematik,in dem Graphen standardmäßig zur Veranschaulichung eingesetzt werden,ist das Gebiet der Wahrscheinlichkeitsrechnung bzw. Stochastik. Schon bei der Berechnung von Wahrscheinlichkeiten der Ereignisse mehrstufiger Zufallsversuche in der Sekundarstufe I ist die Benutzung von Wahrscheinlichkeitsbäumen und Baumpfaden als Darstellungsmittel unbedingt erforderlich.Das Programm Knotengraph stellt in Form der Pfadalgorithmen im Menü Pfade geeignete Algorithmen bereit,um die Wahrscheinlichkeiten von Pfaden in Bäumen bestimmen zu können (vgl. dazu die Kapitel C II und C III und die Aufgaben C XII.9 ff. am Ende dieses Kapitels).

Entfernt man die Beschränkung auf Wahrscheinlichkeitspfade in Bäumen und läßt beliebige Graphen zu,in denen auch Kreise und Schlingen möglich sind,gelangt man zum Begriff der Markovkette.Viele Probleme der Wahrscheinlichkeitsrechnung lassen sich erst durch den Gebrauch von Markovketten darstellen und lösen.Wegen der größeren Komplexität sollte eine entsprechende Unterrichtsreihe erst in der Sekundarstufe II erfolgen.Da in Markovketten nicht mehr von vornherein Blätter als Endzustände und eine Wurzel als Startzustand wie in Bäumen ausgezeichnet sind,ist es nötig bestimmte Knoten als Randzustände in (absorbierenden) Markovketten zu kennzeichnen,zu denen dann von den anderen inneren Knoten (Zuständen) her,die Übergangswahrscheinlichkeiten bestimmt werden sollen.Bei stationären Markovketten entfällt diese Unterscheidung,da quasi jeder Knoten als Endzustand nach einer Iteration der Zufallsversuche aufgefaßt wird.

Die Berechnung der Wahrscheinlichkeiten in Markovketten mit Hilfe der Pfadregeln ist eine auf der Schule kaum praktikable Angelegenheit,da hier fast immer unendliche Summen zu bestimmen sind.Aber auch die Lösung mit Hilfe der Mittelwertregeln ist oft sehr mühselig.An einigen leichten Einführungsbeispielen,wie sie in diesem Kapitel dargestellt sind,sollten diese Verfahren vor Benutzung des Programm Knotengraph als Werkzeug jedoch im Unterricht besprochen werden,damit die Art,die Problematik und die Schwierigkeit des prinzipiellen Lösungsverfahren den Schülern bewußt wird.

Nach einer Idee von A.Engel (Lit 15,S. 212) läßt sich die Bestimmung von Wahrscheinlichkeiten und mittleren Schrittzahlen in Markovketten als Spielalgorithmus durchführen,indem nach bestimmten Regeln Spielsteine auf dem Markovgraphen als Spielbrett so lange gezogen werden,bis der Ausgangszustand (quasi) wieder erreicht wird.Auf diese Weise lassen sich die (exakten) Übergangswahrscheinlichkeiten sehr motivierend auch schon von Schülern jüngeren Alters ermitteln.

Die Regeln diese Spiels sind recht einfach zu verstehen, und deshalb bietet es sich an,einen Algorithmus zum Thema absorbierende und stationäre Markovketten gemäß diesen Spielregeln zu konstruieren.Dieses ist ein erstes Thema dieses Kapitels.

Ein anderer Ansatz,die Übergangswahrscheinlichkeiten von (absorbierenden) Markovketten zu bestimmen,ist,wie oben erwähnt,die Mittelwertregeln zu benutzen.Sie stellen algebraische Beziehungen zwischen den verschiedenen Wahrscheinlichkeitszuständen,d.h. Knoten des Graphen (die als Flussvariable aufgefaßt werden) dar.Die Auswertung solcher Beziehungen war Gegenstand des Mason-Algorithmus im vorigen Kapitel C XI,der ein lineares Gleichungssystem graphisch löste.

Es bietet sich also an,die Lösung mit Hilfe der Mittelwertregeln auf den Mason-Algorithmus durch Vererbung und Erweiterung der dort benutzten Datenstruktur zurückzuführen.Dieses führt insgesamt auf den Begriff des Signalflussgraphen,dessen Spezialisierung auf einen Fluss der Größe Eins wiederum ein Markovgraph ist.Dabei können noch Regeln zur Vereinfachung von Flussgraphen wie Beseitigung von Schlingen und Parallelkanten ausgenutzt werden.Der entsprechende Algorithmus,der ebenfalls die Übergangswahrscheinlichkeiten einer absorbierenden Markovkette bestimmt,ist ein zweites Thema dieses Kapitels.Er läßt sich durch das Menü Anwendungen/Graph reduzieren des Programms Knotengraph starten.

So erhält man einen Vergleich von zwei verschieden Algorithmen zur Berechnung der Übergangswahrscheinlichkeiten von Markovketten,und das Problem läßt sich methodisch von zwei verschieden Aspekten aus betrachten.

Die Programmierung der Algorithmen dieses Kapitels sind zwar nicht schwierig zu verstehen (insbesondere die Spielalgorithmen nicht),aber leider relativ umfangreich.Deshalb ist die Durchführung entsprechender Programmierprojekte nur wenn viel Zeit zur Verfügung steht - z.B. in einem Informatik Leistungskurs - zu empfehlen.Der entsprechende Quellcode wird in diesem Kapitel (deshalb nur) als Überblick über die wichtigsten Methoden besprochen.

Es bietet sich nämlich hier (vor allem im Mathematikunterricht) mehr an das Programm Knotengraph als Werkzeug zu benutzen (bei den Aufgaben dieses Kapitels wird von dieser Option ausgegangen),um mittels der (fertig codierten) Anwendungen absorbierende Markovkette,stationäre Markovkette und Graph reduzieren des Menüs Anwendungen die Funktionsweise und Wirkungsweise dieser Verfahren mittels des Demomodus zu besprechen und zu demonstrieren,sowie im Unterricht geeignete Problemaufgaben zu lösen und dieses Gebiet der Wahrscheinlichkeitsrechnung auf diese Weise zu veranschaulichen und interessanter zu gestalten(z.B. die Veranschaulichung des Hauptsatzes über Markovketten).

Dabei kann dann das Hauptgewicht des Unterrichts auf das Erstellen von Wahrscheinlichkeitsmodellen mittels Markovketten gelegt werden und die Auswertung dem Programm Knotengraph überlassen werden.Das heißt der Mathematisierungs-und Abstraktionsprozeß real vorgegebener Systeme in die Form eines Graphen wird Hauptgegenstand der Betrachtung (Modellbildung).

Im folgenden wird davon ausgegegangen,dass bei den Schülern schon elementare (Mittelstufen-)Kenntnisse zum Thema Wahrscheinlichkeit vorhanden sind und das Thema Markovketten neu begonnen wird.

Aufgabe C XII.1:(Einstiegsproblem):

Mit einem Spielwürfel wird mehrfach gewürfelt.Wie groß ist die Wahrscheinlichkeit,dass die Zahlen 4 oder 6 in unmittelbarer Folge hintereinander (also 4,6 oder 6,4) vor der Zahl 2 oder einer ungeraden Zahl auftritt?

Beim Würfeln der Zahl 2 oder einer ungeraden Zahl wird das Würfeln abgebrochen oder aber beim Erreichen einer Zahlenfolge 4,6 bzw. 6,4.

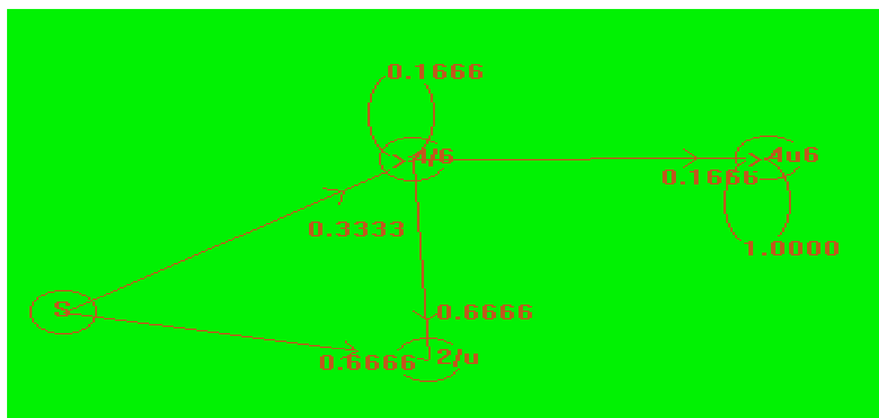
Teil I:

Stelle das Spiel als geeigneten Graphen (Markovkette) dar.

Lösung:

Problemorientierte Erarbeitung mit den Schülern im Unterrichtsgespräch:

I)Die Aufgabe läßt sich folgendermaßen als Graph (Markovkette) darstellen:



G063.gra

Der Knoten S bedeutet Startzustand.In ihm beginnt das Würfeln.Beim ersten Wurf gibt es zwei Möglichkeiten:Entweder wird eine 4 oder 6 oder aber eine 2 oder eine ungerade Zahl gewürfelt.Die erste Möglichkeit wird als Knoten 4/6 dargestellt,und besitzt die Wahrscheinlichkeit $\frac{2}{6}=\frac{1}{3}$,die zweite Möglichkeit wird als 2/u (u=ungerade) dargestellt und besitzt die Wahrscheinlichkeit $\frac{4}{6}=\frac{2}{3}$.Zu den Knoten werden von Knoten S aus Kanten mit der entsprechenden reellen Kantenbewertung erzeugt.

Im Zustand 4/6 gibt es wiederum beim zweiten Wurf die Möglichkeit des Werfens der Zahl 4 oder 6,die im ersten Wurf noch nicht erreicht wurde oder der anderen Zahl,die im ersten Wurf erreicht wurde.Im ersten Fall wird zu einem Knoten 4u6 (4 und 6) mittels einer Kante mit der Wahrscheinlichkeit $\frac{1}{6}$ verzeigt,ansonsten wird eine Schlingenkante zum selben Zustand (Knoten) zurück mit der Wahrscheinlichkeit $\frac{1}{6}$ erzeugt,da immer noch auf das Eintreffen der anderen Zahl 4 oder 6 beim weiteren Würfeln gewartet wird.Eine dritte Möglichkeit tritt ein,wenn die Zahl 2 oder eine ungerade Zahl geworfen wird.Dazu wird eine Kante mit der Wahrscheinlichkeit $\frac{4}{6}=\frac{2}{3}$ zum Knoten 2/u eingefügt.

Der Zustand (Knoten) 4u6 ist der gewünschte Gewinnzustand.Wenn jetzt noch weiter gewürfelt würde,müßte man in diesem Zustand verbleiben.Deshalb wird hier noch zusätzlich eine Schlingenkante mit der Wahrscheinlichkeit 1 eingefügt.

Der Zustand 2/u ist ein Verlustzustand (Endzustand).Dieser Knoten besitzt deshalb keine ausgehenden Kanten.

Die Summe der Wahrscheinlichkeiten der von den übrigen Knoten ausgehenden Kanten ist jeweils gleich 1,da eine der Möglichkeiten auf jeden Fall eintritt.

Weiterführung der Aufgabe: siehe weiter unten (Teil II)

<u>Definition C X.II.1:</u>

Eine absorbierende Markovkette ist ein gerichteter,zusammenhängender Graph mit einer reellen Kantenbewertung,bei der der Wert jeder Kante größer oder gleich Null ist und kleiner gleich 1 ist.Die Summe der Werte aller ausgehenden Kanten eines Knotens ist gleich Eins.Die Knoten nennt man Zustände der Markovkette.Die Kanten der Markovkette mit ihrer Bewertung stellen die Übergangswahrscheinlichkeiten und Übergänge zwischen den Zuständen dar.

Ein Knoten,der nur eine ausgehende Schlingenkante mit dem Wert 1 oder keine ausgehende Kante besitzt,heißt absorbierender Zustand.Die Menge der absorbierenden Zustände einer Markovkette wird der Rand der Markovkette genannt.Die übrigen Knoten heißen innere Zustände oder innere Knoten.

Im Programm Knotengraph in den Menüs Anwendungen/(abs) Markovkette,(stat) Markovkette und Graph reduzieren ist es wie eben gesagt,zulässig für Randknoten statt einer Schlingenkante mit dem Wert 1 auch einen Knoten mit keiner ausgehenden Kante zu erzeugen.Der Unterschied zwischen den Knotenarten ist,dass die Knoten mit den Schlingenkanten mit Wert 1 die (gewünschten) Gewinnzustände bedeuten und die Knoten ohne ausgehende Kanten die Verlustzustände.

Ziel der Berechnung einer Markovkette ist es,die Wahrscheinlichkeit eines inneren Zustands (Startzustands) oder aller inneren Zustände beim Übergang in die Gewinnzustände (oder Verlustzustände gleich der Komplementwahrscheinlichkeit) zu errechnen.

Darunter versteht man die Summe aller Pfadwahrscheinlichkeiten,die vom inneren Knoten(zustand) zum Gewinnzustand bzw. zu den Gewinnzuständen des Randes (Teilmenge des Randes) führen.

Unter der mittleren Schrittzahl (Dauer) von einem inneren Knoten zu den Zuständen des Randes versteht man die Summe der mit den Pfadwahrscheinlichkeiten multiplizierten (gewichteten) Pfadlängen vom inneren Knoten zu allen Randknoten (Randzuständen).

(Die Wahrscheinlichkeiten,die für die inneren Knoten mit Knotengraph berechnet werden,sind immer die Wahrscheinlichkeiten eines Übergangs in die Gewinnzustände.Wenn es nur Randknoten mit der Schlingenkante 1 gibt,ist die Übergangswahrscheinlichkeit der inneren Knoten immer gleich 1.)

Berechnung der Übergangswahrscheinlichkeiten mit Hilfe der Pfadregeln:

Die Wahrscheinlichkeit des Zustandes S in den Zustand 4u6 läßt sich bei der obigen Markovkette nach der Pfadregel der Stochastik längs des Pfades S->4/6->4u8 durch Multiplikation der Wahrscheinlichkeiten längs der Kanten bestimmen.Dabei muß beachtet werden,dass in dem Pfad eine Schlinge beim Knoten 4/6 liegt,die unendlich oft durchlaufen werden kann,so dass sich für die Übergangswahrscheinlichkeit von Knoten S zu Knoten 4/6 folgende unendliche Reihe ergibt:

$$\sum_{i=0}^{\infty} \frac{1}{3} \cdot \frac{1}{6} \cdot \left(\frac{1}{6}\right)^i = \frac{1}{18} \cdot \frac{1}{1-\frac{1}{6}} = \frac{1}{15} = 0{,}0\overline{6}$$

,da es sich um eine geometrische Reihe handelt.

Auf gleiche Weise ergibt sich die Übergangswahrscheinlichkeit für Knoten 4/6 zu:

$$\sum_{i=0}^{\infty} \frac{1}{6} \left(\frac{1}{6}\right)^i = \frac{1}{5} = 0{,}2$$

Die Übergangswahrscheinlichkeiten für Knoten 4u6 ist 1 und für Knoten 2/u ist 0.

Die mittleren Schrittzahlen der Knoten lassen sich folgendermaßen bestimmen:

Zu den Randknoten führen von S aus die Pfade:

1)S->4/6->4u6 mit der Länge 2

2)S->4/6->...->4/6->4u6 mit der Länge 2+i (Schlinge von 4/6 i- mal durchlaufen mit i>0)

3)S->2/u mit der Länge 1
4)S->4/6->4u6 mit der Länge 2
5)S->4/6->...->4/6->2/u mit der Länge 2+i (Schlinge von 4/6 i mal durchlaufen mit i>0)

Gewichtet mit den Pfadwahrscheinlichkeiten unter Berücksichtigung der folgenden Formeln ergibt sich für die mittlere Schrittzahl m_S in S:

$$\sum_{i=0}^{\infty} q^i i - q\sum_{i=1}^{\infty} q^{i-1}(i-1) = \sum_{i=1}^{\infty} q^i = \frac{q}{1-q} \; und \; s = \sum_{i=0}^{\infty} q^i i = \sum_{i=1}^{\infty} q^{i-1}(i-1) fo\lg t \; s = \frac{q}{(1-q)^2} mit |q| < 1$$

m_S=

$$2 \cdot \frac{1}{3} \cdot \frac{1}{6} + \sum_{i=1}^{\infty} \frac{1}{3} \cdot \frac{1}{6}\left(\frac{1}{6}\right)^i (2+i) + 1 \cdot \frac{2}{3} + 2 \cdot \frac{1}{3} \cdot \frac{2}{3} + \sum_{i=1}^{\infty} \frac{1}{3} \cdot \frac{2}{3}\left(\frac{1}{6}\right)^i (2+i) = \frac{63}{45} = 1{,}4$$

Die unendliche Summe ohne die auszuklammernden Faktoren ist eine unendliche Reihe mit dem Wert $\frac{16}{25}$ und läßt sich mittels obiger Summenformeln und der Summenformel geometrischer Reihen ableiten.

In ähnlicher Weise ergibt sich für die mittlere Schrittzahl im Knoten 4/6:

$$m_{4/6}=1 \cdot \frac{1}{6} + \sum_{i=1}^{\infty} \frac{1}{6}\left(\frac{1}{6}\right)^{i}(1 + i) + 1 \cdot \frac{2}{3} + \sum_{i=1}^{\infty} \frac{2}{3}\left(\frac{1}{6}\right)^{i}(1 + i) = \frac{6}{5} = 1{,}2$$

Die unendliche Summe ohne auszuklammernde Faktoren hat hier den Wert $\frac{11}{25}$.

Die Pfade sind hier :

1)4/6->4u6 mit der Pfadlänge 1

2)4/6->...->4/6->4u6 mit der Pfadlänge i+1 (Schlinge bei 4/6 wird i-mal durchlaufen mit i>0.)

3)4/6->2/u mit der Pfadlänge 1

4)4/6->...->4/6->2/u mit der Pfadlänge i+1 (Schlinge bei 4/6 wird i-mal durchlaufen mit i>0)

Die Knoten 2/u und 4u6 haben die mittlere Schrittzahl 0.

Wie man sieht,ist die Verwendung der Pfadregeln sehr umständlich für die Berechnung der Übergangswahrscheinlichkeiten und mittleren Schrittzahlen.Sie lassen sich allerdings durch die bequemeren Mittelwertregeln für Markovketten ersetzen:

Mittelwertregel 1:

Die Übergangswahrscheinlichkeit eines inneren Zustands (Knotens) ist gleich der Summe der Produkte der Wahrscheinlichkeiten der vom Knoten ausgehenden Kanten multipliziert mit der Übergangswahrscheinlichkeiten der Zielknoten dieser Kanten (gleich dem gewichteten Mittel der Wahrscheinlichkeiten der Nachbarknoten/Nachbarzustände).

Mittelwertregel 2:

Die mittlere Schrittzahl eines inneren Knotens ist gleich 1 + der Summe der Produkte der Wahrscheinlichkeiten der vom Knoten ausgehenden Kanten multipliziert mit der mittleren Schrittzahl der Zielknoten dieser Kanten (gleich 1+gewichtetes Mittel der mittleren Schrittzahlen der Nachbarknoten/Nachbarzustände)

Z.B. ist im obigen Beispiel die Übergangswahrscheinlichkeit von Knoten S gleich $\frac{1}{3} \cdot \frac{1}{5} + \frac{2}{3} \cdot 0 = \frac{1}{15}$ und die mittlere Schrittzahl $1 + \frac{1}{3} \cdot \frac{6}{5} + \frac{2}{3} \cdot 0 = \frac{7}{5} = 1{,}4$ .Die Mittelwertregeln gestatten es von den Randzuständen,deren Übergangswahrscheinlichkeiten und mittlere Schrittzahlen zu 1 bzw. 0 bekannt sind, auf die inneren Zustände zurückzurechnen.

Ist die Übergangswahrscheinlichkeit des Knotens 4/6 a,so gilt beispielsweise: $a=\frac{1}{6} \cdot a + \frac{1}{6} \cdot 1 + \frac{2}{3} \cdot 0$,also $a= \frac{1}{5} =$ 0,2.Die Begründung der Mittelwertregeln ergibt sich direkt aus der Pfadregeln,wobei zu beachten ist,dass sich die mittlere Schrittzahl um 1 erhöht,wenn man zu den Nachbarknoten übergeht.(Beweis z.B. Lit 15,,S.22)

Weiterführung der Aufgabe C XII.1:

Teil II:

a)Ermittele mittels des Algorithmus des Menüs Anwendungen/Markovkette (abs) des Programms Knotengraph zu dem Graphen (Markovkette von Teil I) die Übergangswahrscheinlichkeiten für die Knoten (Zustände) S,4/6,4u6 und 2/u.

b)Benutze für den Graphen (Markovkette von Teil I) den Algorithmus des Menüs Anwendungen/Graph reduzieren des Programms Knotengraph zur Berechnung der Übergangswahrscheinlichkeiten.Wähle dazu die Option Graph als Markovkette,und lösche durch Mausklick nacheinander alle Knoten des Graphen,bis der Knoten übrigbleibt,dessen Übergangswahrscheinlichkeit bestimmt werden soll.Lösche dann auch diesen Knoten.

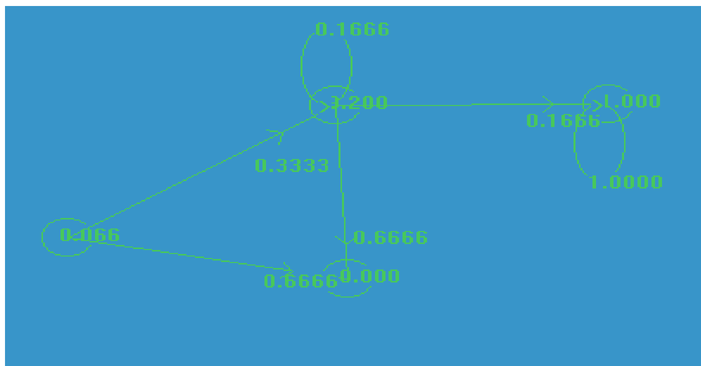
Die Übergangswahrscheinlichkeit des Knotens wird angezeigt.

Führe das Verfahren mehrfach durch,so dass jeder Knoten des Graphen einmal als letzter Knoten zu löschen ist.

(Randknoten können bei diesem Verfahren erst dann gelöscht werden,wenn es maximal noch einen inneren Knoten gibt.)

Lösung:

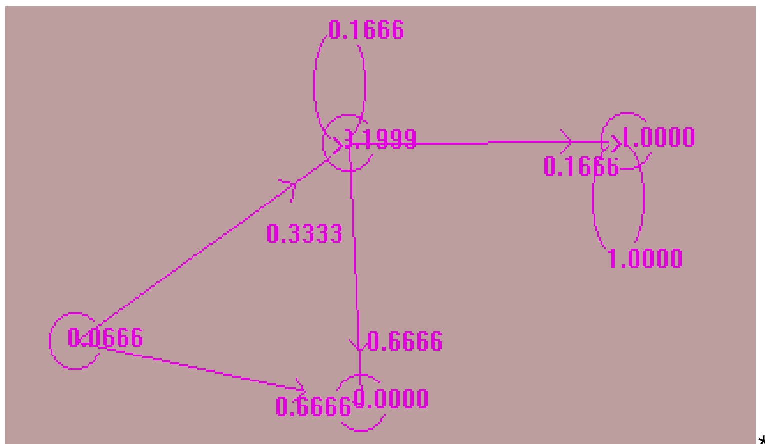
a)



G063.gra

Knoten: S Wahrscheinlichkeit: 0.066 Mittlere Schrittzahl: 1.400
Knoten: 4/6 Wahrscheinlichkeit: 0.200 Mittlere Schrittzahl: 1.202
Knoten: 4u6 Wahrscheinlichkeit: 1.000 Mittlere Schrittzahl: 0.000
Knoten: 2/u Wahrscheinlichkeit: 0.000 Mittlere Schrittzahl: 0.000

b)



*

G063.gra

w 4/6 = 0.1999
wS = 0.0666

w 2/u = 0.0000
w 4u6 = 1.0000

Bemerkungen:

Die Brüche werden durch die eingegebenen Dezimalzahlen nur ungefähr numerisch genau erreicht.

Die Lösung b) beruht auf Berechnungen nach den Mittelwertregeln über Markovketten und ist im allgemeinen genauer als Lösung a).

Näheres dazu bei der Besprechung der Algorithmen

Resultat zu a) und b):

Also beträgt die Übergangswahrscheinlichkeit,die Folge 4,6 oder 6,4 vor 2 oder einer ungeraden Zahl zu werfen, $0,0\bar{6}$ oder $\frac{1}{15}$,und die mittlere Schrittzahl beträgt 1,4.

<u>Aufgabe C XII.2:</u>

Beim Roulettespiel gewinnt man beim Setzen auf die einfachen Chancen Rot oder Schwarz jeweils den Einsatz auf diese Chancen hinzu.Die Wahrscheinlichkeit,dass Rot oder Schwarz erscheint beträgt jeweils 0,5.

(In Wirklichkeit etwas weniger,da auch noch die Zahl 0 mit der Farbe Grün fallen kann.Hier wir der Einfachheit halber aber mit 0,5 gerechnet,um numerisch einfache Ergebnisse zu erreichen.)

Ein Spieler setzt stets auf Rot.Kann er seine Gewinnchanchen dadurch verbessern,dass er als Spielsystem eine Progression benutzt?

Dies bedeutet,dass er im Falle des Verlustes,seinen vorigen Spieleinsatz jeweils verdoppelt.Wenn er z.B. 1 Chip verloren hat,setzt er beim nächsten Spiel 2 Chips.Gewinnt er dann,hat er 4 Chips gewonnen und 3 Chips eingesetzt,also insgesamt 1 Chip gewonnen.

Leider besteht aber beim Setzen auf die einfachen Chancen eine Höchstgrenze des Einsatzes,die hier zu 4 Chips angenommen werden soll,d.h. 8 Chips können nicht mehr gesetzt werden.Dann ist das Spiel verloren und beendet.Der Verlust beträgt dann 7=1+2+4 Chips.

Im Falle des Gewinns besteht das Spielsystem des Spielers darin,den Einsatz plus dem Gewinn als neuen Einsatz auf der Chance Rot fürs nächste Spiel stehen zu lassen.Wenn auf diese Weise 8 Chips insgesamt erreicht sind,beendet er das Spiel und nimmt den Gewinn von 7 Chips (plus 1 Chip Einsatz) entgegen.

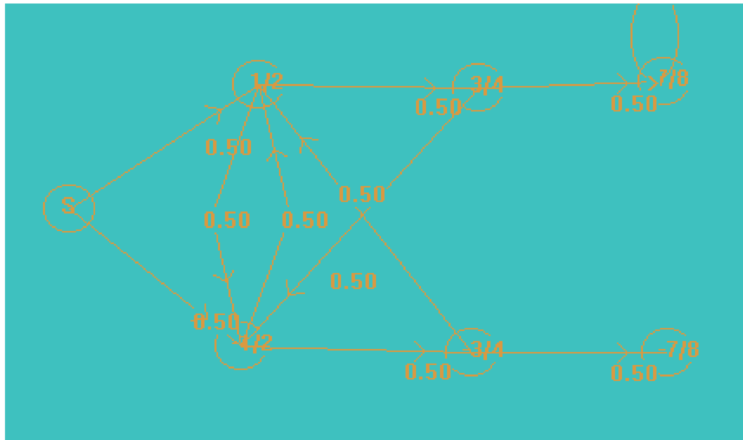
a)Erzeuge einen Graphen als Markovkette.Ermittle jeweils mit Hilfe der Mittelwertregeln die Übergangswahrscheilichkeit und die mittleren Schrittzahlen in den einzelnen Zuständen.

b)Überprüfe die Ergebnisse mit Hilfe des Programms Knotengraph mit den Algorithmen der Menüs Anwendungen/Markovkette (abs) und Graph reduzieren.

Lösung:

Die Bezeichnung x/y zu jedem Knoten bedeutet,x Chips insgesamt gewonnen bzw. verloren (x negativ) und y Chip fürs nächste Spiel neu gesetzt.S ist der Startzustand.Die Zustände 7/8 und -7/8 sind die Gewinn- bzw. Verlustzustände.

a) **Graph:**



G064.gra

Berechnung der Übergangswahrscheinlichkeiten und mittleren Schrittzahlen für die einzelnen Zustände:

Die Übergangswahrscheinlichkeit bzw. mittlere Schrittzahl im Knoten 1/2 sei a bzw. m_a und im Knoten -1/2 sei b bzw. m_b.Dann ist die Übergangswahrscheinlichkeit und die mittlere Schrittzahl im Knoten:

Knoten	Übergangswahrscheinlichkeit:	Mittlere Schrittzahl:
3/4:	$\frac{b}{2} + \frac{1}{2}$	$1 + \frac{m_b}{2}$
-3/4:	$\frac{a}{2}$	$1 + \frac{m_a}{2}$
1/2:	$a = \frac{b}{2} + \frac{b}{4} + \frac{1}{4}$	$m_a = 1 + \frac{1}{2} + \frac{m_b}{4} + \frac{m_b}{2}$
-1/2:	$b = \frac{a}{4} + \frac{a}{2}$	$m_b = 1 + \frac{1}{2} + \frac{m_a}{4} + \frac{m_a}{2}$

Aus den jeweils letzten beiden Gleichungen folgt:

$a = \frac{4}{7}$ und $b = \frac{3}{7}$ $\qquad$ $m_a = 6$ und $m_b = 6$

Also gilt für:

3/4: $\frac{10}{14}$

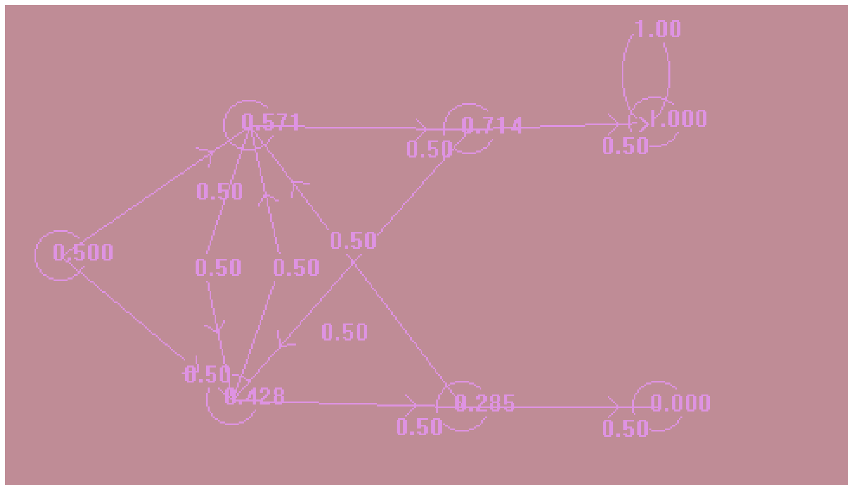
-3/4: $\frac{2}{7}$

S: $w_S = \frac{1}{2} \cdot \frac{4}{7} + \frac{1}{2} \cdot \frac{3}{7} = \frac{1}{2}$ $\qquad$ $m_S = 1 + 3 + 3 = 7$

Die Übergangswahrscheinlichkeiten in 7/8 und -7/8 sind 1 und 0.

Die mittleren Schrittzahlen in beiden Knoten sind 0.

b) **Lösung mit Algorithmus (abs) Markovkette:**

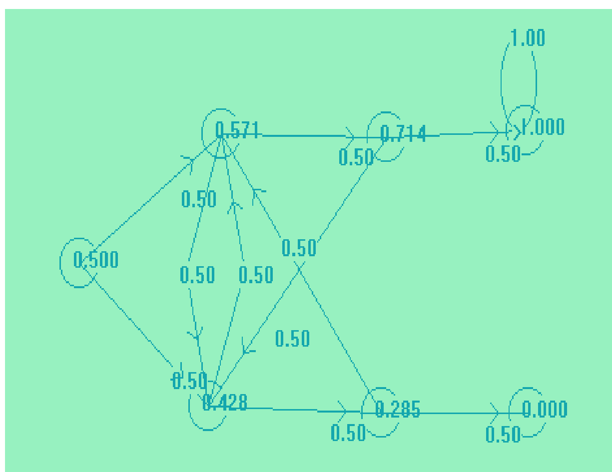


G064.gra

Knoten: S Wahrscheinlichkeit: 0.500 Mittlere Schrittzahl: 7.000
Knoten: ½ Wahrscheinlichkeit: 0.571 Mittlere Schrittzahl: 6.000
Knoten: ¾ Wahrscheinlichkeit: 0.714 Mittlere Schrittzahl: 4.000
Knoten: -1/2 Wahrscheinlichkeit: 0.428 Mittlere Schrittzahl: 6.000
Knoten: -3/4 Wahrscheinlichkeit: 0.285 Mittlere Schrittzahl: 4.000
Knoten: -7/8 Wahrscheinlichkeit: 0.000 Mittlere Schrittzahl: 0.000
Knoten: 7/8 Wahrscheinlichkeit: 1.000 Mittlere Schrittzahl: 0.000

Bemerkung:Die Dezimalzahlen entsprechen den obigen Brüchen.

Lösung mit Algorithmus Graph reduzieren:



G064.gra

wS = 0.500
w1/2 = 0.571
w-1/2 = 0.428
w3/4 = 0.714
w-3/4 = 0.285
w7/8 = 1.000
w-7/8 = 0.000

Ergebnis:

Das Spielsystem der Progression bringt keinen Vorteil,weil die Gewinnwahrscheinnlichkeit in S nach wie vor 0.5 für Einsatz auf die Chance Rot beträgt.Im Mittel ist das Spiel nach 7 Einsätzen verloren oder gewonnen.

Wie die obigen Rechnungen per Hand zeigen,ist es trotz Einsatz der Mittelwertregeln noch immer besonders bei großen Graphen mit Schlingen und Kreisen mühsam,das entstehende Gleichungssystem aufzustellen und zu lösen.Besser geeignet sind Algorithmen,wie sie das Programm Knotengraph in Form der Algorithmen der Menüs Anwendungen/(abs) Markovkette und Anwendungen/Graph reduzieren bereitstellt.

Wie funktionieren diese Algorithmen?

Der letztgenannte basiert auf den Beziehungen der Knoten bzw. der zugehörigen Variablen auf Grund der Mittelwertregeln.

Diese Beziehungen sind nichts anderes als ein lineares Gleichungssystem und lassen sich mit dem Algorithmus des Menüs Anwendungen/Gleichungssystem lösen.Das letztgenannte Verfahren wird an zweiter Stelle besprochen.

Zunächst aber zu dem Algorithmus,der durch das Menü Absorbierende Markovkette gestartet wird.

Algorithmus Absorbierende Markovkette

Verbale Beschreibung:

Die Kantenwerte des Markovgraphen bedeuten Übergangswahrscheinlichkeiten von einem Knoten zum anderen.Faßt man die Wahrscheinlichkeit als (Übergangs-)Fluss auf,der in Pfeilrichtung der Kanten von einem Knoten zum anderen fließt,so bedeuten die Knoten Verteilung des Flusses gemäß der Wahrscheinlichkeiten der von ihnen ausgehenden Kanten.Die Verteilungssumme ist dabei jeweils 1.Die Randknoten sind dann die Knoten,in denen sich der Fluss sammelt und absorbiert wird.Der Startknoten ist dann der Ausgangspunkt des Flusses.

Diese Überlegung legt folgenden Algorithmus nahe:

Setze in den Startknoten eine in beliebig feine Teile zu teilende Masse der Größe Eins und verteile sie in aufeinanderfolgenden Schritten jeweils gemäß den Übergangswahrscheinlichkeiten von dort zu allen Nachbarknoten,und von diesen wieder zu den Nachbarknoten der Nachbarknoten usw..In den Randknoten sammeln sich die Massenanteile,die dort eintreffen,an.Nach sehr vielen Zugschritten ist dann der in den jeweiligen Randknoten angesammelte Massenanteil gleich der Übergangswahrscheinlichkeit vom Startknoten zum Randknoten.

Der Algorithmus hat in dieser einfachen Form den Nachteil,dass das Verfahren stets ein Näherungsverfahren ist,und die Zahl der benötigten Schritte,um eine vorgegebene Genauigkeit zu erreichen,von der Größe des Graphen abhängt,und deshalb die Schrittzahl unbestimmt ist.

A.Engel (Lit 15, S. 212) hat das Verfahren in der Weise verbessert,dass es prinzipiell eine exakte Lösung liefert,ein endliches Verfahren ist,bei dem das Ende des Algorithmus klar definiert ist,und außerdem dazu noch als (didaktisches) Brettspiel (Wahrscheinlichkeitsabakus) aufgefaßt werden kann,bei dem Spielsteine gezogen werden sollen.

Dieses sehr anschauliche Verfahren scheint deshalb für einen Algorithmus,der von Schülern nachvollzogen werden soll,besonders geeignet zu sein.

Der Algorithmus im einzelnen:

Die Wahrscheinlichkeitsbewertung der Kanten sei in Form von (gekürzten) Brüchen gegeben.Es sei N_j der Hauptnenner der Nenner $N_{j1},N_{j2},...N_{jk}$ der Brüche der Kantenwahrscheinlichkeiten der ausgehenden Kanten eines inneren Knotens

k_j des Markovgraphen.Jedem Knoten wird dann eine Anzahl (Spiel-)Steine zugeordnet.

1)Setze die Zahl der Spielsteine jedes Knotens k_j zu Beginn auf den Wert N_j-1.Die Randknoten erhalten dabei Null Steine.Wenn ein innerer Knoten die Zahl von N_j-1 Steinen hat,heißt der Knoten kritisch geladen.Wenn alle inneren Knoten des Graphen kritisch geladen sind,heißt der Graph kritisch geladen.

Spielsteine sollen von einem Knoten zum anderen längs der Kantenrichtungen gezogen werden,wobei man den Graphen als Spielbrett aufzufassen hat.(Die Steine befinden sich stets in den Knoten und werden längs der Kanten gezogen.)

Wichtige Bedingung ist,dass nur ganze Steine gezogen werden können.Dies ist nur dann möglich,wenn sich in einem Knoten k_j mindestens N_j Steine d.h. N_j+r Steine (r ganzzahlig und nichtnegativ,sowie $r<N_j$) befinden.Die Steine werden dann als ganzahliger Bruchteil $N_{jl} \cdot N_j$ längs der l.ten ausgehenden Kante auf die Nachbarzielknoten des Knotens k_l verteilt.Dabei bleiben r Steine beim j.ten Knoten liegen.

2)Man beginnt dabei zunächst die Steinezahl des Startknotens S um 1 zu erhöhen,so dass der Knoten mit N_S Steinen überkritisch geladen ist.

3)Diese Steine können dann alle auf die Nachbarknoten verteilt werden,die dadurch ebenfalls überkritisch geladen werden.Es wird solange in beliebiger Reihenfolge der Knoten gezogen,solange noch überkritische Knoten vorhanden sind.In den Randknoten bleiben die Steine liegen und sammeln sich an.

4)Wenn kein innerer Knoten mehr überkritisch geladen ist,wird geprüft,ob der Graph erneut kritisch geladen ist.Dann ist der (Spiel-)Algorithmus zu Ende,und die Ergebnisse können bestimmt werden.Setze dazu das Verfahren bei Schritt 6) fort.

5)Wenn der Graph nicht kritisch geladen ist,wird die Zahl der Steine im Startknoten solang um Eins erhöht,bis dieser Knoten wieder die Zahl N_S Steine besitzt.Anschließend kann wieder gezogen werden.Setze dann den Algorithmus bei Schritt 3) fort.

6)Bei Ende des Algorithmus läßt sich die Übergangswahrscheinlichkeit vom Startknoten S zu einer Auswahlmenge (Teilmenge) des Randes T als die Summe der Anzahl der Steine in den Knoten,die zu T gehören dividiert durch die Gesamtzahl der Steine,die sich in allen Knoten des Randes befinden,bestimmen.Die letzte Zahl ist gleich der Gesamtzahl der Steine,die jeweils benötigt wurden,um den Startknoten jeweils evtl. wiederholt auf N_S Steinen zu nachzuladen.

Es sei die Gesamtzahl aller Züge von Steinen zwischen den Knoten gleich G.Dann ist die mittlere Schrittzahl vom Startknoten einen der Knoten des Randes zu erreichen,gleich dem Quotienten aus der Zahl G und der Summe der Steine in den Randknoten.

8)Wird der Algorithmus mit jedem inneren Knoten des Graphen als Startknoten durchgeführt,ergibt sich so die Übergangswahrscheinlichkeit und die mittlere Schrittzahl für jeden Knoten des Graphen.

(In den Randknoten ist die Übergangswahrscheinlichkeit gleich 1 (Auswahlknoten der Teilmenge T) oder 0 und die mittlere Schrittzahl 0)

Wird das Verfahren mit Brüchen als Kantenbewertung durchgeführt,liefert das Verfahren exakte Ergebnisse.Bei Knotengraph sind dagegen die Kantenbewertungen als (Pseudo-) reelle Zahlen (mit endlicher Stellenzahl vom Datentyp Extended) vorgegeben,und deshalb können die Wahrscheinlichkeiten als Brüche nur angenähert mit endlicher Stellenzahl dargestellt werden (z.B. mit n Nachkommastellen).Dann ist ein Standardwert für den gemeinsamen Nenner aller Brüche die Zehnerpotenz 10^n.

Wenn mit großen Werten von n d.h. mit großer Genauigkeit gerechnet wird,müssen unter Umständen sehr viele Steine durch den Graphen bewegt werden,bis der Algorithmus gemäß der Abbruchbedingung terminiert.Auf diese Weise kann die zur Errechnung der Ergebnisse benötigte Zeit bis zum Abbruch sehr lang werden.Also muß n so gewählt werden,dass sowohl die Ausführungs-

zeit als auch die Genauigkeit annehmbar sind.Wird 3 (Stellen Genauigkeit) als Vorgabewert gewählt (angemessener Zeitaufwand),kann dann unter Berücksichtigung von Rundungsfehler mit einer Genauigkeit der Ergebnisse von ca. 1 Promille gerechnet werden.

Eine reine Integer-Lösung (und auch Long-Integer) statt Extended als Datentyp des Algorithmus hat den Nachteil,dass bei umfangreichen Graphen und bei großen Nenner der Brüche der Kantenwahrscheinlichkeiten leicht die Höchstgrenze des Integerbereichs überschritten wird.

Aufgabe C XII.I (Fortsetzung)

III.Teil:

Beobachtung und problemorientierte Erarbeitung:

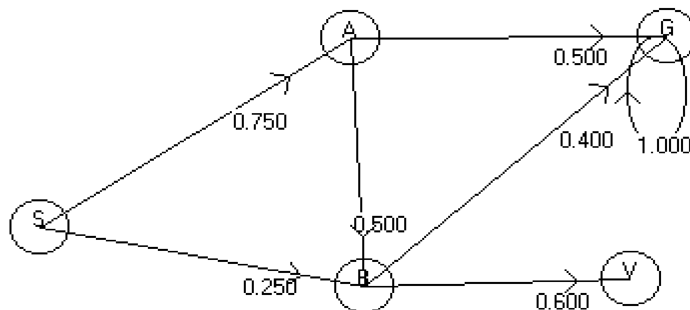
Beobachte den Ablauf des Algorithmus Anwendungen/Markovkette (abs) des Programms Knotengraph an Hand des Graphen G063a.gra (nur für Knoten S) im Demo- oder Einzelschrittmodus und notiere die Spielsteine-Verteilung nach jedem Schritt in einer Tabelle.Erkläre anschließend die jeweilige Verteilung an Hand der Spielregeln.Spiele das Spiel dann per Hand nach.(Verfahre analog auch bei anderen Graphen z.B. G063.gra,G064.gra)

Lösung:(G063a.gra)

Tabelle und Graph:

(Die erste Zeile zeigt die kritische Zahl der Steine in den Knoten.)

3	1	4	-	-
S	A	B	V	G
3	1	4	0	0
4	1	4	0	0
0	4	5	0	0
0	0	7	0	2
0	0	2	3	4
3	0	2	3	4
4	0	2	3	4
0	3	3	3	4
0	3	3	3	4
0	1	4	3	5
3	1	4	3	5



G063a.gra

Knoten: S Wahrscheinlichkeit: 0.625 $=\frac{5}{8}$

Der Quellcode im Auszug:

Die Datenstruktur:

```
TMarkovknoten = class(TInhaltsknoten)
  private
    Wahrscheinlichkeit_:Extended;
    MittlereSchrittzahl_:Extended;
    Anzahl_:Extended;
    Minimum_:Extended;
    VorigeAnzahl_:Extended;                                          *)
    Delta_:Extended;                                                 *)
    Ergebnis_:string;
    Anzeige_:string;
    procedure SetzeWahrscheinlichkeit(Wa:Extended);
    function WelcheWahrscheinlichkeit:Extended;
    procedure SetzeMittlereSchrittzahl(Schr:Extended);
    function WelcheMittlereSchrittzahl:Extended;
    procedure SetzeAnzahl(Anz:Extended);
```

```
    function WelcheAnzahl:Extended;
    procedure SetzeMinimum(Mi:Extended);
    function WelchesMinimum:Extended;
    procedure SetzevorigeAnzahl(Vaz:Extended);
    function WelchevorigeAnzahl:Extended;
    procedure SetzeDelta(De:Extended);
    function WelchesDelta:Extended;
    function WelchesErgebnis:string;
    procedure SetzeErgebnis(S:string);
    function WelcheAnzeige:string;
    procedure SetzeAnzeige(S:string);
  public**
    constructor Create;
    property Wahrscheinlichkeit:Extended read WelcheWahrscheinlichkeit write SetzeWahrschein-
lichkeit;
    property MittlereSchrittzahl:Extended read WelcheMittlereSchrittzahl write SetzeMittlere-
Schrittzahl;
    property Anzahl:Extended read WelcheAnzahl write SetzeAnzahl;
    property Minimum:Extended read WelchesMinimum write SetzeMinimum;
    property VorigeAnzahl:Extended read WelchevorigeAnzahl write SetzevorigeAnzahl;      *)
    property Delta:Extended read WelchesDelta write SetzeDelta;                          *)
    property Ergebnis:string read WelchesErgebnis write SetzeErgebnis;
    property Anzeige:string read WelcheAnzeige write SetzeAnzeige;
    function Wertlisteschreiben:Tstringlist;override;
    procedure Wertlistelesen;override;
    procedure ErhoeheAnzahlumEins(var Steine:Extended);
    function Randknoten:Boolean;
    function KnotenungleichRandistkritisch:Boolean;
    function Fehler:Boolean;
    function Auswahlknoten:Boolean;
    function KnotenungleichRandistueberkritisch:Boolean;
    procedure LadeKnotenkritisch;
  end;
```

Die mit *) gekennzeichneten Felder werden erst von der Anwendung (stat) Markovkette verwendet.

TMarkovknoten leitet sich durch Vererbung von TInhaltsknoten ab und enthält folgende neue Datenfelder,auf die mittels Propertys zugegriffen werden kann:

Wahrscheinlichkeit_ :Speichert die Übergangsswahrscheinlichkeit.
MittlereSchrittzahl_ :Speichert die mittlere Schrittzahl.
Anzahl_ : Speichert die Anzahl der Spielesteine.
Minimum_: Speichert die Zahl Spielsteine,bei der Knoten kritisch ist vergrößert um 1.
VorigeAnzahl_: Speichert die Anzahl Spielsteine des vorigen Zuges.
Delta_ :Speichert die Differenz,um die die Anzahl Spielsteine vergrößert werden.
Ergebnis_ :Nimmt ein Ergebnis auf.
Anzeige_ :Nimmt ein Ergebnis für die Anzeige auf der Zeichenoberfläche auf.

```
TMarkovkante = class(TInhaltskante)
public
    function KanteistSchlingemitWerteins:Boolean;
    function Fehler:Boolean;
  end;
```

TMarkovkante leitet sich von TInhaltskante durch Vererbung ab und enthält keine neuen Datenfelder.

```
TMarkovgraph = class(TInhaltsgraph)
  public
    constructor Create;
    function Fehler:Boolean;
    procedure AnzahlgleichNull;
    procedure LadeKnotenkritischohneStartundRandknoten(Startknoten:TMarkovknoten);
    function AlleKnotenohneRandsindkritisch:Boolean;
    procedure AddiereAnzahlaufAuswahlknoten(Var Gesamtzahl:Extended);
    function AlleKnotenausserRandsindkritischoderunterkritisch:Boolean;
    procedure LadeStartknotenkritischnach(Startknoten:TMarkovknoten;Var Steine:Extended);
    function EnthaeltAuswahlknoten:Boolean;
    procedure BestimmeMinimum(Genauigkeit:Integer);
    procedure SetzeKnotenWahrscheinlichkeitgleichNull;
    procedure ZieheSteineabs(Ausgabe:TLabel;var Schrittzahl:Extended;Flaeche:TCanvas);
    procedure Markovabs(Kno:TMarkovknoten;Ausgabe1,Ausgabe2:Tlabel;
      var Gesamtzahl:Extended;var SListe:TStringlist;Flaeche:TCanvas;Genauigkeit:Integer);
    procedure Markovkette(Flaeche:TCanvas;Ausgabe1,Ausgabe2:TLabel;var Sliste:TStringlist);
  end;
```

TMarkovgraph leitet sich von TInhaltsgraph durch Vererbung ab und enthält keine neuen Datenfelder.

Die Methode Markovabs bestimmt für den übergebenen Startknoten Kno die Übergangswahrscheinlichkeit und die mittlere Schrittdauer.

```
procedure TMarkovgraph.Markovabs(Kno:TMarkovknoten;Ausgabe1,Ausgabe2:TLabel;
  var Gesamtzahl:Extended;Var Sliste:TStringlist;
  Flaeche:TCanvas;Genauigkeit:Integer);
var Index:Integer;
    Startknoten:TMarkovknoten;
    Schrittzahl,Steine:Extended;
begin
  AnzahlgleichNull;
  Startknoten:=Kno;                                                          1)
  Schrittzahl:=0;
  Steine:=0;
  Gesamtzahl:=0;
  BestimmeMinimum(Genauigkeit);                                              2)
  if not Startknoten.Randknoten then                                         3)
  begin
    LadeKnotenkritischohneStartundRandknoten(Startknoten);                   4)
    Startknoten.LadeKnotenkritisch;                                          5)
    repeat                                                                   6)
      Startknoten.ErhoeheAnzahlumeins(Steine);                               7)
      repeat                                                                 8)
        ZieheSteineabs(Ausgabe2,Schrittzahl,Flaeche)                         9)
      until AlleKnotenausserRandsindkritischoderunterkritisch;              10)
      LadeStartKnotenkritischnach(Startknoten,Steine);
    until  AlleKnotenohneRandsindKritisch;                                  11)
    AddiereAnzahlaufAuswahlknoten(Gesamtzahl);                              12)
    Startknoten.Wahrscheinlichkeit:=Gesamtzahl/Steine;                      13)
    Startknoten.MittlereSchrittzahl:=Schrittzahl/Steine;                    14)
  end
  else
    if Startknoten.Auswahlknoten then                                       15)
    begin
      Startknoten.Wahrscheinlichkeit:=1;
      Startknoten.MittlereSchrittzahl:=0;
    end
    else
      if Startknoten.Randknoten then
      begin
        Startknoten.Wahrscheinlichkeit:=0;
        Startknoten.MittlereSchrittzahl:=0;
      end;
  with Startknoten do                                                       16)
  begin
    Ergebnis:=Wert+chr(13)+'Wahrscheinlichkeit: '+
      RundeZahltostring(Wahrscheinlichkeit,Knotengenauigkeit)
      +chr(13)+'Mittlere Schrittzahl: '+
      RundeZahltostring(MittlereSchrittzahl,Knotengenauigkeit);
    Sliste.Add('Knoten: '+Wert+' Wahrscheinlichkeit: '+
    RundeZahltostring(Wahrscheinlichkeit,Knotengenauigkeit)
      +' Mittlere Schrittzahl: '+
      RundeZahltostring(MittlereSchrittzahl,Knotengenauigkeit));
    Anzeige:=
      RundeZahltostring(Wahrscheinlichkeit,Knotengenauigkeit);
  end;
end;
```

Bei 1) wird zunächst die Variable Startknoten mit dem Knoten Kno belegt, und die Werte von Schrittzahl,Steine und Gesamtzahl mit 0 initiiert.Bei 2) wird das Minimum der Steine bestimmt,bei dem in den einzelnen inneren Knoten gezogen werden kann und als Steineminimum in den Knoten gespeichert.Die Zahl richtet sich nach der vorgegebenen Genauigkeit.Wenn bei 3) der Startknoten nicht zum Rand gehört,werden die folgenden Schritte ausgeführt:

Bei 4) werden alle inneren Knoten außer dem Startknoten kritisch geladen, und bei 5) wird der Startknoten kritisch geladen (Anzahl=Minimum-1).

Bei 6) beginnt eine Schleife,so dass die folgenden Anweisungen solange wiederholt werden,bis alle inneren Knoten bei 11) wieder kritisch geladen sind.

Dazu wird bei 7) die Anzahl (der Steine) des Startknoten um 1 erhöht,so dass gezogen werden kann.Bei 8),9) und 10) (vgl. die Methode ZieheSteineabs im Quelltextlisting) werden jeweils solange Steine von den inneren Knoten zu den Nachbarknoten der ausgehenden Kanten gezogen,bis es bei keinen inneren Knoten mehr möglich ist,weil die Steineanzahl in allen inneren Knoten weniger als das Minimum beträgt.Bei 12) wird die Summe der Anzahl der Steine in den Auswahlknoten auf dem Rand,zu denen vom Startknoten aus die Übergangswahrscheinlichkeit berechnet werden sollen (gleich obiger Teilmenge T),berechnet und bei 13) und 14) können damit die Übergangswahrscheinlichkeit im Startknoten und die mittlere Schrittzahl nach den oben genannten Formeln berechnet werden.

Der Fall,dass der Startknoten zum Rand gehören,wird bei 15) behandelt.Entweder ist dann die Übergangswahrscheinlichkeit 1 oder 0,je nachdem,ob es sich dabei um einen Auswahlknoten oder um einen anderen Randknoten handelt.Die Mittlere Schrittzahl ist dort stets 0.

Bei 16) werden alle Ergebnisse im Feld Ergebnis des Startknoten sowie in der Stringliste SListe zwecks Ausgabe im Ausgabefenster zusammengestellt und gespeichert.

Die Methode Markovkette ruft die Methode Markovabs mit den gewünschten Startknoten auf:

```
procedure TMarkovgraph.Markovkette(Flaeche:TCanvas;Ausgabe1,Ausgabe2:TLabel;
  var Sliste:TStringlist);
var Kno:TMarkovknoten;
    Strz,Fehlerstri:string;
    NurEinknoten,eingabe:Boolean;
    Index:Integer;
    Gesamtzahl:Extended;
    Str:string;
    Genauigkeit:Integer;
begin
  if Fehler then                                                          17)
  begin
    ShowMessage('Zuerst Fehler beheben!');
    exit;
 end;
 if not EnthaeltAuswahlknoten then                                        18)
 begin
   ShowMessage('Keine ausgewählten Zustände (Knoten) des '+chr(13)+'Randes (mit 1) mar
   kiert!');
   exit;
 end;
 if MessageDlg('Nur ein Knoten?',mtConfirmation,[mbYes,mbNo],0)=mryes     19)
 then
   NureinKnoten:=true
 else
   NurEinknoten:=false;
   Str:='3';
 Repeat                                                                   20)
   Eingabe:=Inputquery('Eingabe Genauigkeit:','Genauigkeit (1 bis 5 Stellen):',str);
   if StringtoInteger(Str)<=0 then Genauigkeit:=3 else Genauigkeit:=abs(StringtoInteger(Str));
 until Eingabe and (Genauigkeit<6);
 BestimmeMinimum(Genauigkeit);
 SetzeKnotenWahrscheinlichkeitgleichNull;                                 21)
 Kno:=TMarkovknoten(LetzterMausklickknoten);
 if Kno=nil then Kno:=TMarkovknoten(Self.Anfangsknoten);
 Gesamtzahl:=0;
 if  NurEinKnoten
 then
   Markovabs(Kno,Ausgabe1,Ausgabe2,Gesamtzahl,SListe,Flaeche,Genauigkeit)   22)
 else
   if not Leer then
     for Index:=0 to Knotenliste.Anzahl-1 do                              23)
       Markvabs(TMarkovknoten(Knotenliste.knoten(Index)),Ausgabe1,
       Ausgabe2,Gesamtzahl,SListe,Flaeche,Genauigkeit);
  Strz:='Startknoten '+Kno.Wert+':'+chr(13)+                              24)
 `'Wahrscheinlichkeit: '+RundeZahltoString(Kno.Wahrscheinlichkeit,Kantengenauigkeit)+
  chr(13)+'Mittlere Schrittzahl: '+
  RundeZahltoString(Kno.MittlereSchrittzahl,Kantengenauigkeit);
  ShowMessage(Strz);
end;
```

Bei 17) wird zunächst geprüft,ob es sich bei dem Graphen um eine Markovkette (Wahrscheinlichkeitssumme der ausgehenden Kantenbewertung eines Knotens gleich 1/Methode Fehler) handelt und bei 18),ob überhaupt mindestens ein Knoten als Auswahlknoten markiert ist.Bei 19) wird vorgegeben,ob die Über-

gangswahrscheinlichkeit in einem inneren Knoten (gleich letzter mit der Maus durch Mausklick ausgewählter Knoten,falls existent,oder der erste in den Graph eingefügte Knoten) oder für alle inneren Knoten bestimmt werden soll.Bei 20) wird die Rechengenauigkeit (Stellenzahl) vorgegeben,und bei 21) wird das Wahrscheinlichkeitsfeld aller Knoten mit 0 initiiert.Bei 22) wird dann die Methode Markovabs für einen Knoten als Startknoten (falls diese Option gewählt wurde) und bei 23) für alle Knoten des Graphen als jeweiliger Startknoten aufgerufen.Bei 24) wird jeweils die Übergangswahrscheinlichkeit und die mittlere Schrittzahl für den letzten mit der Maus durch Mausklick ausgewählten Knoten (oder falls nicht existent für den ersten in den Graph eingefügten Knoten) ausgegeben.

Die Methode BestimmeMinimum bestimmt für jeden Knoten das Minimum der Steine,bei denen in den einzelnen inneren Knoten gezogen werden kann:

```
procedure TMarkovgraph.BestimmeMinimum(Genauigkeit:Integer);
var Index:Integer;
    Anfangswert:Extended;

  procedure BestimmeMinimum(Kno:TMarkovknoten);
  var Divisor:Extended;
      Index,Zaehl:Integer;

    procedure FindeDivisor(Ka:TKante);
    begin
      if Round(StringtoReal(TInhaltskante(Ka).Wert)*Anfangswert)<>0 then
        Divisor:=GGT(Round(Divisor),Round(StringtoReal(TInhaltskante(Ka).Wert)*Anfangswert));
    end;

  begin
    Anfangswert:=1;
    for Zaehl:=1 to Genauigkeit do Anfangswert:=Anfangswert*10;                    25)
    Divisor:=Anfangswert;
    if not Kno.AusgehendeKantenliste.Leer then
      for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
        FindeDivisor(Kno.AusgehendeKantenliste.Kante(Index));                      26)
    Kno.Minimum:=Round(Anfangswert) div Round(Divisor);                            27)
  end;

begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      BestimmeMinimum(TMarkovKnoten(Knotenliste.Knoten(Index)));
end;
```

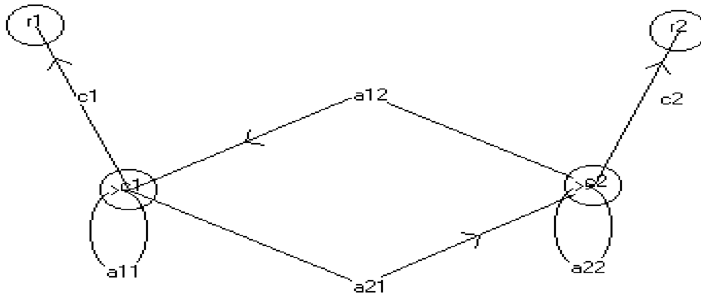
Bei 25) wird zunächst eine Zehnerpotenz 10^n erzeugt,wobei n die bei den Kantenwahrscheinlichkeiten zu berücksichtigende Stellengenauigkeit ist.Bei 26) und 27) wird dann für jeden Knoten der größte gemeinsame Teiler der Zähler aller als Brüche mit der Zehnerpotenz 10^n geschriebenen Kantenwahrscheinlicheiten (bzw. deren Näherungswert als Bruch: z.B. wird aus 0,33333333 mit n=3 $0{,}333=\frac{333}{1000}$) ermittelt. (z.B. für die Wahrscheinlichkeiten 0,5;0,2 und 0,3 ergibt sich für n=3 GGT(500,200,300)=100,d.h. Divisor =100) Das Minimum der Steine ergibt sich dann als Quotient der Zehnerpotenz dividiert durch den GGT (vgl. Zeile 27).

(Im letztem Beispiel also 1000:100=10 Steine,so dass jeweils 5,3 und 2 Steine auf einmal längs der Kanten gezogen werden können.

In diesem Beispiel genügt auch eine Genauigkeit von n=1,um dasselbe Ergebnis zu erhalten.)

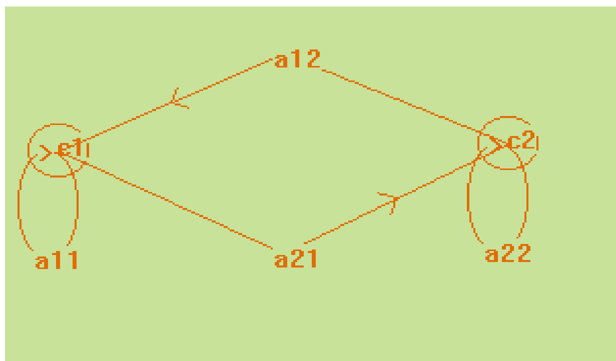
Algorithmus Graph reduzieren

Der Algorithmus Graph reduzieren beruht einerseits auf den Mittelwertregeln für Markovketten und andererseits auf der Zusammenfassung von Schlingen und Parallelkanten.



G065.gra

Betrachtet wird dazu zunächst der obige (Signal-oder Übergangs-)Flussgraph als Teilgraph einer Markovkette.Die Knoten mit der Bezeichnung r_1 und r_2 seien Knoten des Randes mit den Übergangswahrscheinlichkeiten 1 oder 0.



G066.gra

Diese Übergangswahrscheinlichkeiten machen sich gemäß den Mittelwertsregeln in den Knoten,von denen eingehende Kanten nach r_1 und r_2 verlaufen,mit den Wahrscheinlichkeitsanteilen c_1 und c_2 jeweils multipliziert mit 1 oder Null bemerkbar.Um die Bezeichnungen zu vereinfachen,seien c_1 und c_2 schon gleich dem Wert der eben genannten Produkte gesetzt.(Also bleibt der Wert der Variablen erhalten,oder er ist gleich 0.)

Dann kann der Graph, wie der obige zweite Graph zeigt, so vereinfacht werden,dass der Übergangswahrscheinlichkeitseinfluss der Randknoten als Knoteninhalt der entsprechenden inneren Knoten geschrieben wird.Wenn ein innerer Knoten keine ausgehende Kante hat,die in einem Randknoten endet,ist der Knoteninhalt natürlich gleich 0.

Dem Knoten mit dem Wert c_1 sei jetzt die Übergangswahrscheinlichkeitsvariable x_1 und dem Knoten mit dem Wert c_2 die Variable x_2 zugeordnet.

Dann gilt nach der 1. Mittelwertregel:

$x_1 = a_{11}x_1 + a_{21}x_2 + c_1$

$x_2 = a_{12}x_1 + a_{22}x_2 + c_2$

Die Gleichungen können folgendermaßen umgeformt werden:

$(1-a_{11})x_1 + (-a_{21})x_2 = c_1$

$(-a_{12})\ x_1 + (1-a_{22})x_2 = c_2$

Dieses ist ein Gleichungssystem,das mit dem im letzten Kapitel XI beschriebenen Algorithmus von Mason auf einem geeigneten Graphen (Algorithmus des Menüs Anwendungen/Gleichungssystem des Programm Knotengraph) gelöst werden kann.

Um den dortigen Algorithmus zu benutzen (vgl. die Bezeichnungen in Kapitel XI),müßten nur die Werte der Kanten folgendermaßen geändert werden:

α)Die Schlingenkante beim i.ten Knoten K_i mit dem Wert s_{ii} erhält den neuen Wert $1-s_{ii}$.

β)Alle anderen Kanten zwischen verschiedenen Knoten z.B. vom Knoten K_j zum Knoten K_l mit dem Wert w_{jl} erhalten den Wert $-w_{jl}$.

Auf diese Weise entsteht ein äquivalentes Gleichungssystem in der Schreibweise von Mason.

(Dort wurden die Konstanten c_1 und c_2 (allgemein c_i) ebenfalls gerade als Knoteninhalte gespeichert.)

(Analoge Ergebnisse erhält man natürlich,wenn das obige Gleichungssystem aus n Gleichungen mit n Variablen besteht.)

Dann kann nach dem Algorithmus von Kapitel C XI ein Knoten gelöscht werden,wodurch ein neues äquivalentes Gleichungssystem mit einer um 1 verminderten Knotenzahl entsteht.

Durch Rückverwandlung wieder mit den Transformationsgleichungen α) und β) (welche symmetrisch in der Umwandlungsrichtung sind),kann jetzt wieder aus dem Gleichungssystem nach Mason ein Markovgraph gewonnen werden.

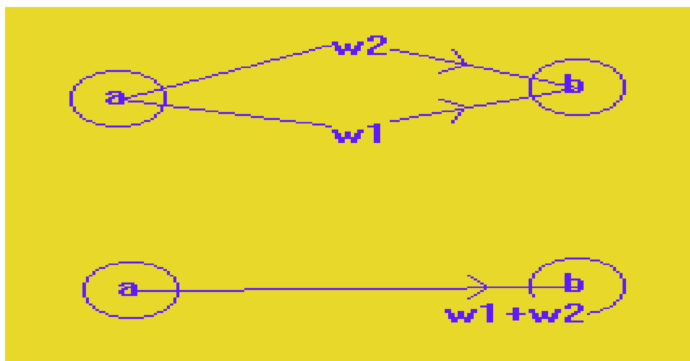
(Dazu werden eventuelle Kanten mit dem Wert 0 eliminiert.)

Ein Problem bei der Verwandlung eines Markovgraphen in ein Mason-Graph-Gleichungssystem ist allerdings,dass in dem Markovgraphen keine Parallelkanten auftreten dürfen,weil diese beim Mason-Algorithmus nicht berücksichtigt werden (und auch nicht erlaubt sind).

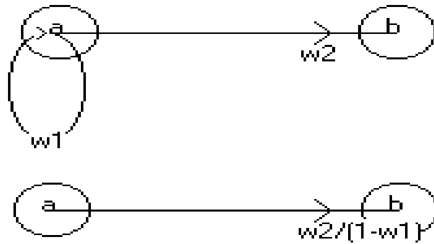
Deshalb müssen alle Parallelkanten vorher beseitigt werden.

Es gelten nämlich folgende Reduktionsregeln zur Vereinfachung von Markovketten bzw. (Signal-)Flussgraphen:

A)Zwei Parallelkanten können durch eine Kante ersetzt werden,deren Wert die Summe der Wahrscheinlichkeiten (der Flüsse) der Kanten ist.



B)Ein Schlingenkante mit dem Wert w_1,die zu einem Knoten gehört,der eine ausgehende Kante mit der Wahrscheinlichkeit w_2 besitzt,läßt sich dadurch beseitigen,dass der ausgehenden Kante eine neue Wahrscheinlichkeit (Fluss) des Wertes $\frac{w_2}{1 - w_1}$ zugewiesen wird.



Die Regel A) ergibt sich direkt daraus,dass gemäß den Mittelwertregeln die Summe der Wahrscheinlichkeiten der parallelen Pfade zu berechnen ist (bzw. aus der Eigenschaften der Erhaltung von Flüssen) und die Regel B) aus der Mittelwertbeziehung:

$x_a=w_1x_a + w_2x_b$,also $x_a = \frac{w_2}{1 - w_1} x_b$

Daher ergibt sich folgender Algorithmus:

Algorithmus Graph reduzieren

Verbale Beschreibung:

1)Reduziere Parallelkanten und Schlingen im vorgegebenen Markov- (oder Fluss-)Graphen.

2)Ändere die Werte der Kanten nach den Transformationsgleichungen α) und β) und erzeuge dadurch einen Mason-Graph.

(Für alle Knoten K_i ohne Schlingen werden dazu noch Schlingenkanten mit dem Wert 1(=a_{ii}) erzeugt.)

3)Prüfe,ob der Graph nur noch aus einem Knoten besteht.Gehe dann nach Schritt 6).

4)Wende den Algorithmus von Kapitel XI nach Mason an,und lösche im Graph einen Knoten.Erhalte dadurch ein äquivalentes Gleichungssystem,das durch den neu entstandene Graphen dargestellt wird.

5)Ändere die Werte der Kanten erneut nach den Transformationsgleichungen α) und β),und erzeuge dadurch einen Markov-(oder Fluss-)Graphen.

(Lösche dazu noch Kanten mit dem Wert 0.)

Gehe dann nach Schritt 2).

6)Berechne die Lösung für die Knotenvariable,und gib sie aus.Die Lösung ist dann die Übergangswahrscheinlichkeit.

Bemerkung:

Bei der Entfernung von inneren Knoten ohne Schlingen des Graphen läuft das Verfahren,wie sich aus der Regel

$a_{kl}'=a_{kl}-\frac{a_{il}a_{ki}}{a_{ii}}$ des Schrittes 3 des Algorithmus von Mason in Kapitel C XI ergibt,auf die Anwendung der Pfadregeln unter zusätzlicher Anwendung der

Regel A hinaus.Die Regel lautet nämlich mit den entsprechenden Übergangswahrscheinlichkeiten geschrieben:

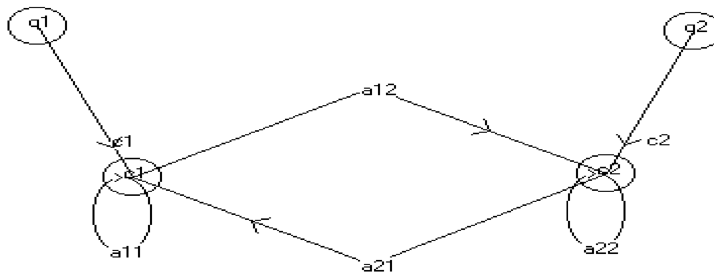
$w_{kl}' = w_{kl} + w_{il}w_{ki}$,da sich der Wert von a_{ii}=0 nach 1

transformiert (keine Schlinge).(Die Werte von a_{kl},a_{il},a_{ki},a_{kl}' transformieren sich jeweils in die Werte mit entgegengesetztem Vorzeichen.Bei der Rücktransformation wird das Vorzeichen dann wieder umgekehrt.)

Daher kann das Verfahren auch leicht per Hand ausgeführt werden,wenn man zu Beginn alle Schlingen nach Regel B reduziert,und Randknoten erst als letzte Knoten gelöscht werden (siehe Aufgabe C XII.3).

Allgemeine Signalflussgraphen

Man kann den oben als Beispiel benutzten Graphen auch mit Kantenpfeilen in umgekehrter Richtung zeichnen:



Statt Randknoten bedeuten die Knoten q_1 und q_2 jetzt zwei Quellenknoten,von denen ein Signalfluss der Stärke c_1 und c_2 in die Knoten mit den Signalflüssen x_1 und x_2 hineinfließt.Die Werte der einlaufenden Kanten des Knotens x_i vom Knoten x_j (i,j=1,2) bedeuten jetzt,mit welchem Anteil ein Signalfluss der Stärke x_j beim Knoten x_i ankommt.

Im Gegensatz zu den Flussgraphen der nächsten Kapitel C XIII und C XIV (u.a. Transportflüsse) sowie von Kapitel C IX (Maximalflussproblem) bedeuten also hier die Kantenwerte nicht Flüsse sondern zu übertragende Anteile,und der Fluss wird durch die Knotenwerte (Knotenvariablen) angezeigt.

Der oben genannte Algorithmus läßt sich dann in gleicher Weise auf die Bestimmung von (Signal-)Flüssen in Signal-oder Übergangsgraphen übertragen.Lediglich die Pfeilrichtung der Kanten zeigen dabei in umgekehrte Richtung,weil sie so wie bei Markovketten üblich gewählt worden sind.

Außerdem müssen statt Auswahlknoten Quellknoten mit Angabe des Flusses festgelegt werden.Die Quellknoten werden wieder durch eine Schlingenkante mit dem Wert 1 oder alternativ mit dem Wert q gekennzeichnet.Die letzte Kennzeichnung funktioniert auch dann noch,wenn der Knoten noch zusätzlich ausgehende Kanten besitzt,also dann sowohl Quellknoten als auch innerer Knoten ist.

Man erhält den Algorithmus für Signalflussgraphen,indem man beim Ablauf des Algorithmus des Menüs Anwendungen/Graph reduzieren des Programms Knotengraphs bei der entsprechenden Abfrage die Option Markovgraph? nein wählt.

Der Quellcode im Auszug:

Der Algorithmus ist also die direkte Erweiterung des Algorithmus Gleichungssystem.Deshalb wird TMarkovreduziereKnoten von TGleichungssystemknoten durch Vererbung abgeleitet.Der Datentyp enthält keine neuen Datenfelder.

```
TMarkovreduziereKnoten = class(TGleichungssystemknoten)
  public
    function Randknoten:Boolean;
    function Fehler:Boolean;
    function Auswahlknoten:Boolean;
  end;
```

Ebenfalls wird TMarkovreduzieregraph von TGleichungssytemgraph durch Vererbung abgeleitet.Der Datentyp enthält keine neuen Datenfelder.

```
TMarkovreduzieregraph = class(TGleichungssystemgraph)
  public
    function  Fehler:Boolean;
    procedure SetzeKnotenNullundAuswahlknoteneins;
    procedure SetzeKnotenNullundAuswahlknotenWert;
    procedure FindeundreduziereSchlinge;
    procedure SetzeSchlingenaufNull;
    procedure ErgaenzeSchlingenWerteins;
    procedure SpeichereSchlingenundKantenum;
    procedure SucheundreduziereParallelkanten;
    procedure LoescheKantenmitWertNull;
    function  AnzahlRandknoten:Integer;
    procedure LoescheKnotenGraphreduzieren(var Kno:TKnoten;
      Flaeche:TCanvas;var Sliste:Tstringlist;Ausgabe1,Ausgabe2:TLabel;
      var Loesung:Extended;var Ende:Boolean;
      var G:TInhaltsgraph;var Oberflaeche:TForm);
    procedure GraphInit(Markov:Boolean;Flaeche:TCanvas;var Oberflaeche:TForm;
      var G:TInhaltsgraph;Ausgabe2:Tlabel);
  end;
```

Die Methode Graphinit bereitet den Signalfluss-bzw. Markovgraphen durch Reduktion von Parallelkanten und Schlingen gemäß 1) auf die Transformation zum Mason-Graph vor:

```
procedure TMarkovreduziereGraph.Graphinit(Markov:Boolean;Flaeche:TCanvas;
  var Oberflaeche:TForm;
var G:TInhaltsgraph;Ausgabe2:Tlabel);
begin
  if Markov
  then
    SetzeKnotenNullundAuswahlknoteneins                                    1)
  else
    SetzeKnotenNullundAuswahlKnotenWert;                                   2)
  ZeichneGraph(Flaeche);
  SucheundreduziereParallelkanten;                                         3)
  ZeichneGraph(Flaeche);
  FindeundReduziereSchlinge;                                               4)
  ZeichneGraph(Flaeche);
  SetzeSchlingenaufNull;                                                   5)
  LoescheKantenmitwertNull;                                                6)
  ZeichneGraph(Flaeche);
  LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
end;
```

Bei 1) und 2) wird er Wert der Randknoten des Markovgraphen bzw. im Falle des Signalflussgraphen der Wert des (der Fluss durch den) Quellenknoten gesetzt.Bei 3) und 4) werden nacheinander Parallelkanten und Schlingen gesucht,und gemäß den obigen Formeln reduziert.Bei 5) und 6) werden die Schlingenkanten der Auswahlknoten gelöscht.

Damit ist ein reduzierter Markovgraph vorbereitet.

Die Methode LoescheKnotenGraphreduzieren führt dann die Schritte 2) bis 6) des obigen Verbalalgorithmus aus:

```
procedure TMarkovreduzieregraph.LoescheKnotenGraphreduzieren(var Kno:TKnoten;Flaeche:TCanvas;
  var Sliste:TStringlist;Ausgabe1,Ausgabe2:TLabel;var Loesung:Extended;var Ende:Boolean;
  var G:Tinhaltsgraph;var Oberflaeche:TForm);
label Endproc;
begin
  If (Self.AnzahlKnoten>self.AnzahlRandknoten+1)and (TMarkovreduziereknoten(Kno).Randknoten)
  then
```

```
  begin                                                                          7)
    ShowMessage('Randknoten können erst gelöscht werden,wenn maximal'
      +chr(13)+'noch ein innerer Knoten zusätzlich vorhanden ist!');
    goto Endproc;
  end;
  ErgaenzeKanten;                                                                8)
  ZeichneGraph(Flaeche);
  SpeichereSchlingenundKantenum;                                                 9)
  ZeichneGraph(Flaeche);
  ErgaenzeSchlingenWerteins;                                                    10)
  ZeichneGraph(Flaeche);
  EliminiereKnoten(Kno,Flaeche,Ausgabe1,Loesung,Ende,G,Oberflaeche);            11)
  ZeichneGraph(Flaeche);
  SpeichereSchlingenundKantenum;                                                12)
  ZeichneGraph(Flaeche);
  LoescheKantenmitWertNull;                                                     13)
  ZeichneGraph(Flaeche);
  FindeundreduziereSchlinge;                                                    14)
  LoescheKantenmitWertNull;
  ZeichneGraph(Flaeche);
  Endproc:
end;
```

Um die Eigenschaft eines Markov- oder Signalflussgraphen mit Randknoten bis zuletzt zu erhalten,können die Randknoten erst dann gelöscht werden,wenn nur noch ein innerer Knoten existiert (7).Dann ist die Übergangswahrscheinlichkeit dieses inneren Knoten zu den Randknoten direkt an den Kantenwerten ablesbar.

Bei 8) werden die eventuell fehlenden Kanten mit dem Wert 0 zwischen den Knoten zur Lösung des Gleichungssystems erzeugt,und bei 9) werden dann die Kanten gemäß den Transformationsgleichungen α) und β) für den Gleichungssystemgraph nach Mason verändert.Fehlende Schlinge mit dem Wert 1 werden in den Knoten des Graphen bei 10) ergänzt,so dass jetzt die Umwandlung in den entsprechenden Mason-Gleichungssystemgraph vollständig ist.

Bei 11) kann dann nach dem Algorithmus von Kapitel C XI ein Knoten gelöscht werden (vererbte Methode),und bei 12) werden dann wieder bei dem reduzierten Graphen die Schlingen und Parallelkanten zurücktransformiert,um wieder einen Signalfluss- bzw. Markov-Graphen zu erhalten.Dazu werden bei 13) alle Kanten mit dem Wert 0 gelöscht und bei 14) für Markovgraphen alle Schlingen bei inneren Knoten reduziert,so dass ein Schlingenfreier Markov-oder Signalflussgraph entsteht.

(Die Reduktion der Schlingen kann nur bei inneren Knoten mit dem Wert 0 erfolgen,da sonst die Äquivalenz zwischen Graph und Gleichungssystem nicht mehr gegeben ist.)

Die Methode LoescheKnotenGraphreduzieren wird,nachdem jeweils ein neuer zu löschender Knoten mit der Maus durch Mausklick ausgewählt wurde,von der Menümethode Graphreduzierenclick von TKnotengraph (unter Einsatz von GleichungssystemMouseDown vgl. voriges Kapitel C XI) mit diesem Knoten als Parameter aufgerufen.Der auch ohne Beschreibung verständliche Quelltext der im obigen Text erwähnten übrigen Methoden kann im Listing des Gesamtprogramms Knotengraph (im Anhang) nachgelesen werden.

Bemerkung:

Der Algorithmus Graph reduzieren kann auch leicht manuell ausgeführt werden,wenn man folgendes Schema benutzt:

1)Reduziere alle Schlingen und Parallelkanten nach den oben genannten Regeln.

2)Die Randknoten werden prinzipiell nicht gelöscht.

3)Jeweils in Serie liegende Kanten können mit den Pfadregeln der Wahrscheinlichkeitsrechnung durch Multiplikation der Wahrscheinlichkeiten zusammengefasst werden.Dadurch lassen sich Zwischenknoten eliminieren.

4)Dabei neu auftretende Parallelkanten lassen sich wieder nach den oben genannten Regeln zusammenfassen.

5)Kreise im Graphen lassen sich als Schlinge berücksichtigen und nach oben genannter Regel reduzieren.

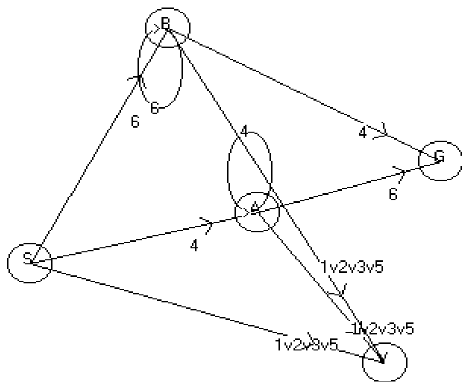
6)Setze das Verfahren unter Löschen innerer Knoten fort,bis alle inneren Knoten gelöscht sind.Danach können die Wahrscheinlichkeiten an Hand der verbleibenden Pfade abgelesen werden.

Aufgabe C XII.3:(Einstiegsaufgabe als weitere Fortsetzung von CXII.1)

Beobachtung und problemorientierte Erarbeitung:

Löse Aufgabe C XII.1 (II. Teil) nach diesem Verfahren per Hand.Erstelle zunächst als Graph einen Automaten,der das Spiel simuliert,und wandele den Automaten dann in eine Markovkette um.Beobachte die Schritte anschließend mit Hilfe des Programm Knotengraph Anwendungen/Menü Graph reduzieren an Hand des Graphen G063.b.gra durch Löschen der Knoten.

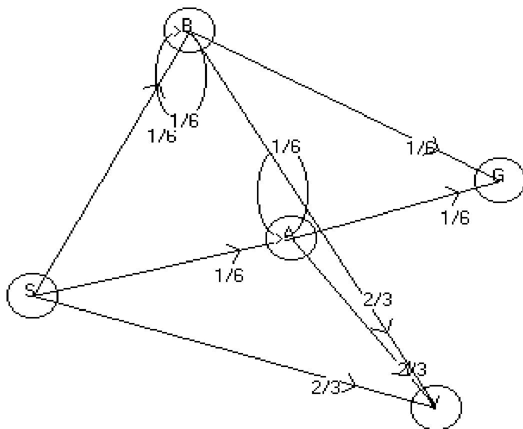
Lösung:

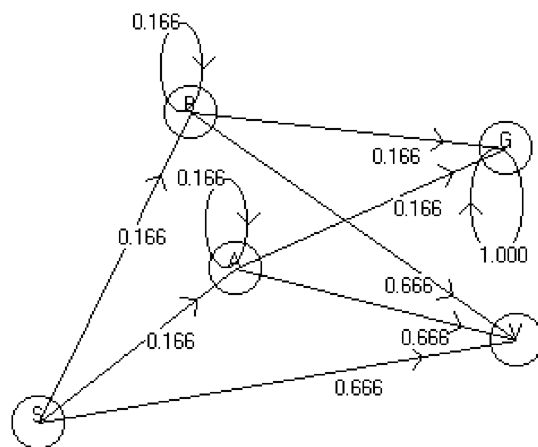


Automat als Graph:

Dieser Graph entspricht G063.gra, der die Zustände A und B kumuliert. Die Aufgabe kann auch mit G063.gra mit Bruchbewertung der Kanten gelöst werden,falls das Spiel nicht mit dem Automaten simuliert werden soll.

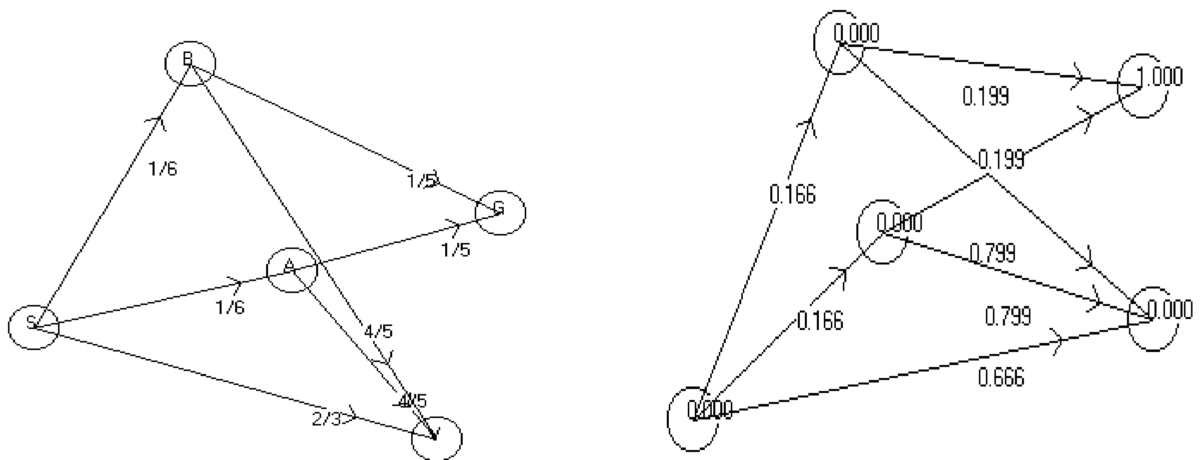
Wahrscheinlichkeiten als Brüche (Markovkette):





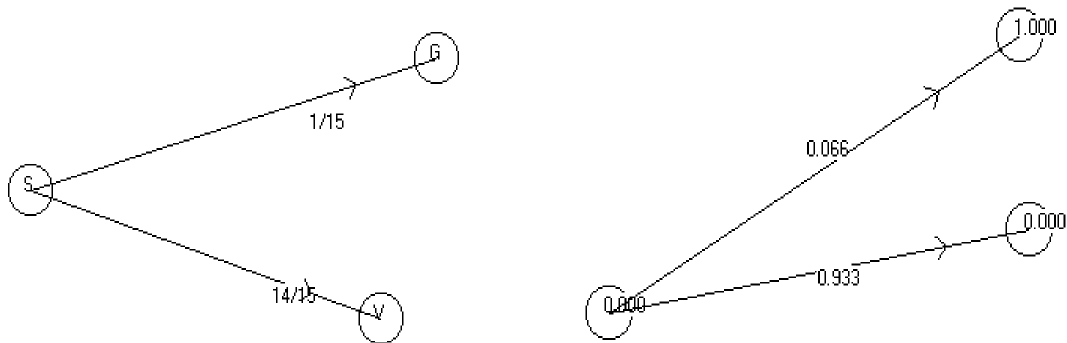
G063b.gra

Schlingen nach obiger Regel reduzieren:



G063b.gra

Pfadregel für Serien-Kanten anwenden und Parallelkanten zusammenfassen:



G063b.gra

Lösung: $\frac{1}{15}$ = 0,066... wie oben

Auf diese Weise lassen sich die Unterrichtsthemen Endlicher Automat und Markovketten miteinander didaktisch verknüpfen.

Zusatz: Simuliere den obigen Automaten mit Hilfe des Menüs Anwendungen/Endlicher Automat des Programms Knotengraph. (vgl. Kapitel VII)

Aufgabe C XII.4 (Übungsaufgabe als Fortsetzung von C XII.1 und C XII.3)

Aufgabe C XII.1 Teil a) wird so abgeändert, dass das Spiel nicht verloren ist, wenn die Zahl 2 gewürfelt wird. Ansonsten führt nach wie vor das Würfeln einer ungeraden Zahl zum Verlust und die Kombinationen 4,6 oder 6,4 (unmittelbar hintereinander) zum Gewinn.

a) Erstelle mit Knotengraph einen Markovgraphen.

b) Ermittle die Übergangswahrscheinlichkeiten und die mittlere Schrittdauer für alle Knoten mit dem Algorithmus des Menüs Anwendungen/Markovkette (abs)

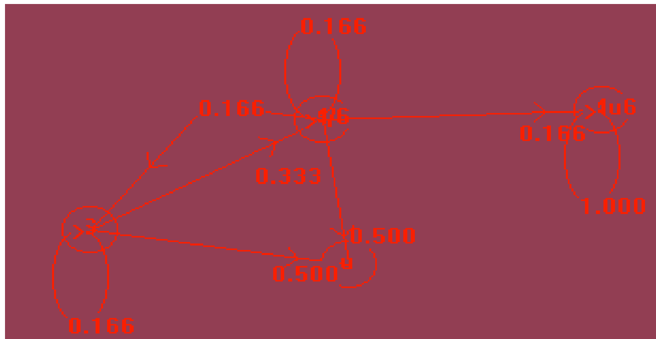
(für die Genauigkeit 3).Verfolge den Ablauf des Verfahrens im Demomodus (für die Genauigkeiten 1 oder 2).

c)Ermittle die Übergangswahrscheinlichkeit für alle Knoten mit dem Algorithmus des Menüs Anwendungen/Graph reduzieren.Verfolge den Ablauf des Algorithmus im Demomodus,insbesondere auch die Erstellung und Lösung des Gleichungssystems.

d)Bestimme die Lösungen der Übergangswahrscheinlichkeiten für den Knoten S auch ohne Programmeinsatz per Hand.

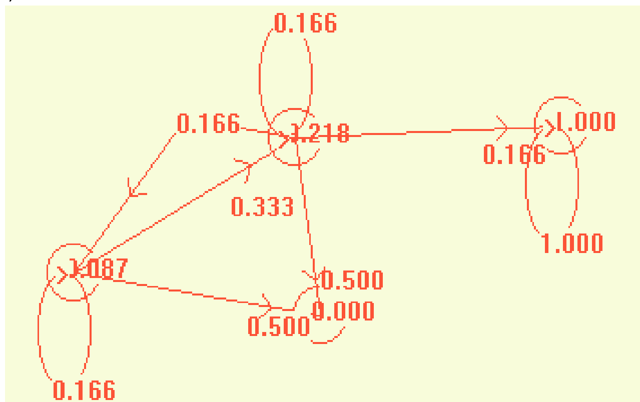
Lösung:

a)Markovgraph:



G068.gra

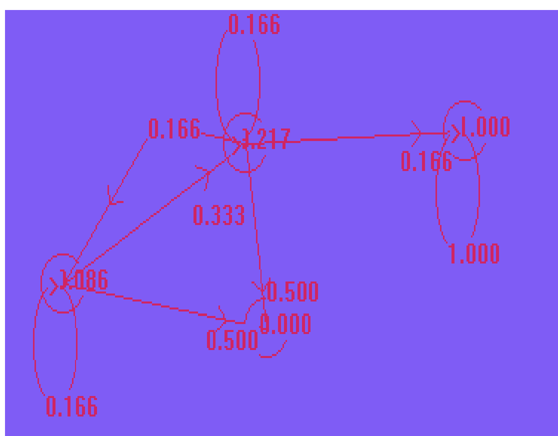
b)



G068.gra

Knoten: S Wahrscheinlichkeit: 0.087 Mittlere Schrittzahl: 1.828
Knoten: 4/6 Wahrscheinlichkeit: 0.218 Mittlere Schrittzahl: 1.570
Knoten: 4u6 Wahrscheinlichkeit: 1.000 Mittlere Schrittzahl: 0.000
Knoten: u Wahrscheinlichkeit: 0.000 Mittlere Schrittzahl: 0.000

c)



G068.gra

wS = 0.086
w4/6 = 0.217
w4u6 = 1.000
wu = 0.000

d)Lösung wie bei Aufgabe C XII.3

Aufgabe C XII.5:(Einstiegsaufgabe)

Eine Münze wird wiederholt geworfen,bis man gewonnen oder verloren hat.Gewonnen hat man,wenn unmittelbar hintereinander die Kombination Wappen,Zahl,Wappen (WZW) auftritt.Das Spiel ist verloren,wenn unmittelbar hintereinander dreimal Zahl geworfen wird (ZZZ).

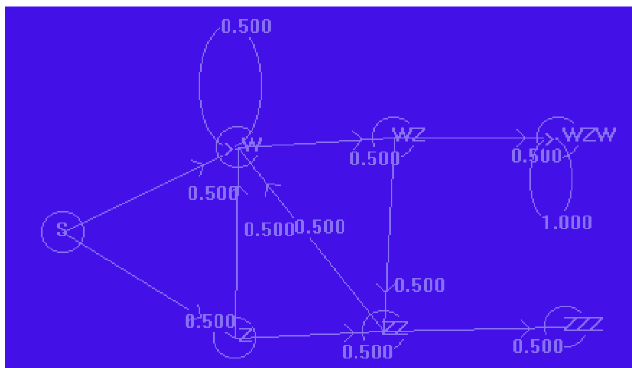
a)Erstelle mit Knotengraph eine Markovgraphen.

b)Ermittle die Übergangswahrscheinlichkeiten und die mittleren Schrittzahlen in den einzelnen Knoten (Zuständen) mit Hilfe des Algorithmus des Menüs Anwendungen/Markovkette (abs).

c)Ermittle die Übergangswahrscheinlichkeiten in den einzelnen Knoten mit Hilfe des Algorithmus des Menüs Anwendungen/Graph reduzieren.

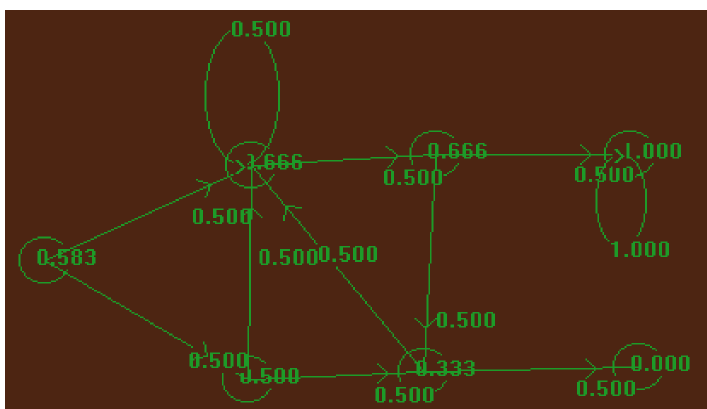
Lösung:

a)Markovgraph:



G069.gra

b)

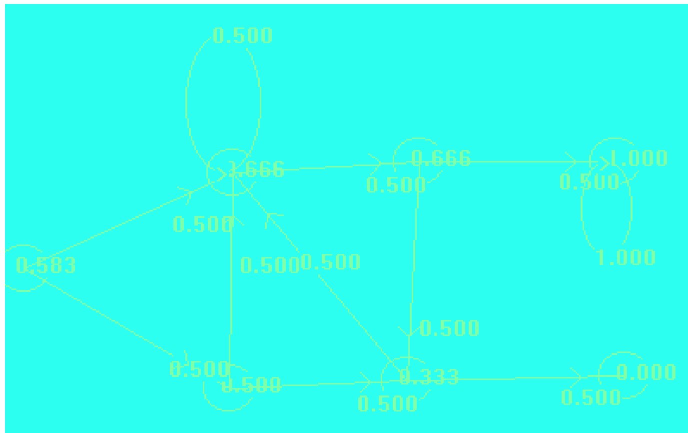


G069.gra

Knoten: S Wahrscheinlichkeit: 0.583 Mittlere Schrittzahl: 5.833
Knoten: W Wahrscheinlichkeit: 0.666 Mittlere Schrittzahl: 4.666
Knoten: WZ Wahrscheinlichkeit: 0.666 Mittlere Schrittzahl: 2.666

Knoten: WZW Wahrscheinlichkeit: 1.000 Mittlere Schrittzahl: 0.000
Knoten: Z Wahrscheinlichkeit: 0.500 Mittlere Schrittzahl: 5.000
Knoten: ZZ Wahrscheinlichkeit: 0.333 Mittlere Schrittzahl: 3.333
Knoten: ZZZ Wahrscheinlichkeit: 0.000 Mittlere Schrittzahl: 0.000

c)



G069.gra

w S = 0.583
w W = 0.666
w Z = 0.500
w WZ = 0.666
w ZZ = 0.333
w WZW = 1.000
w ZZZ = 0.000

An Hand des letzten Beispiels soll jetzt der Hauptsatz für absorbierende Markovketten und sein Zusammenhang mit dem Algorithmus Graph reduzieren erläutert werden.

<u>Satz C XII.1 Hauptsatz für absorbierende Markovketten:</u>

Vorgegeben sei ein absorbierender Markovgraph.

Es sei b_{ij} die mittlere Anzahl der Besuche im Knoten k_j bei Start in Knoten k_i, m_i die mittlere Schrittzahl in die ausgewählten Knoten des Randes beim Start im Knoten k_i sowie u_{ij} die Übergangswahrscheinlichkeit beim Start in Knoten k_i im Knoten k_j absorbiert zu werden.

Die inneren Knoten des Graphen seien mit $k_1,k_2....k_m$ nummeriert,und die Knoten des Randes mit $k_{m+1},...k_{m+r}$.(mit m+r=n Gesamtzahl der Knoten)

Es seien w_{ij} die Wahrscheinlichkeitswerte der Kanten vom Knoten k_i zum Knoten k_j .

Mit W sei die mxm-Matrix der Wahrscheinlichkeiten w_{ij} der Kanten zwischen den inneren Knoten bezeichnet.

Mit R sei die nxr-Matrix der Wahrscheinlichkeiten w_{ij} der Kanten zwischen den inneren Knoten als Anfangsknoten und den Randknoten als Endknoten bezeichnet.

Mit B sei die mxm-Matrix der mittleren Anzahlen der Besuche b_{ij} im inneren Knoten k_j des Graphen beim Start vom inneren Knoten k_i bezeichnet.

Mit U sei die mxr-Matrix der Übergangswahrscheinlichkeiten u_{ij},vom inneren Knoten k_i aus im Randknoten k_j absorbiert zu werden,bezeichnet.

E sei die mxm-Einheitsmatrix.

Behauptung:

Es gilt:

1) $B=(E-W)^{-1}$
2) $m_i=b_{i1} + b_{i2}+ ... + b_{im}$ für i=1 bis m
3) $U=(E-W)^{-1} R$

Die Matrix E-W sei mit N und die Matrix $(E-W)^{-1}$ mit M bezeichnet.

Beweis:

1) Für die mittlere Anzahl der Besuche im Knoten k_j bei Start in k_i gilt nach den Mittelwertsregeln:

a) $b_{ij}=\sum_{s=1}^{m} w_{is}b_{sj}$,falls die Knoten k_i und k_j verschieden sind.

b) $b_{ij}=1+\sum_{s=1}^{m} w_{is}b_{sj}$,falls der Knoten k_i gleich k_j ist,weil dann die Anzahl der zukünftigen Besuche im Knoten k_i um 1 höher ist.

In Matrix-Schreibweise ergibt sich daraus:

$B = E+ WB$
$B - WB= E$
$(E-W)B = E$
$B = (E - W)^{-1}$
$B = M$

2) Die Werte b_{ij} geben die mittlere Schrittzahl beim Start in k_i in den Knoten k_j an. Daher ergibt sich die mittlere Gesamtschrittzahl, vom Knoten k_i aus im Rand absorbiert zu werden, als die Summe der b_{ij} für j=1 bis m.

3) Nach der 1. Mittelwertregel gilt für die Übergangswahrscheinlichkeit, vom Knoten k_i aus im Knoten k_j absorbiert zu werden:

$$u_{ij} = w_{ij} + \sum_{s=1}^{m} w_{is}u_{sj} \quad \text{für } 0\leq i\leq m \text{ und } m+1\leq j\leq m+r$$

Das bedeutet in Matrix-Schreibweise:

$U= R + WU$
$U - WU = R$
$(E-W)U = R$
$U = (E-W)^{-1} R$
$U = MR$

Dies bedeutet,dass wenn die Matrix $M = N^{-1}$ bekannt ist,sowohl die Übergangswahrscheinlickeiten von inneren Knoten zu Knoten des Randes (Auswahlknoten) mittels U=MR,als auch die mittlere Schrittzahl (Verweildauer) im inneren Knoten k_j beim Start im inneren Knoten k_i als Matrix B,und daraus als jeweilige Zeilensumme dieser Matrix die mittleren Schrittzahlen bis zur Absorption in den Randknoten m_i bestimmt werden können.

Die Matrix M heißt **Fundamentalmatrix** der absorbierenden Markovkette.

Wie läßt sich die Matrix $M = N^{-1}$ bestimmen?

Es gilt N = E - W.

Übertragen auf den Graphen bedeutet dies,dass die obigen Regeln α) und β)

α)Die Schlingenkante beim i.ten Knoten k_i mit dem Wert s_{ii} erhält den neuen Wert $1-s_{ii}$.

β)Alle anderen Kanten zwischen verschiedenen Knoten z.B. vom Knoten k_i zum Knoten k_j mit dem Wert w_{ij} erhalten den Wert $-w_{ij}$.

auf die Kanten zwischen den inneren Knoten des Graphen anzuwenden sind,d.h. dass gerade aus dem Markov-Graph ein Mason-Gleichungssystemgraph herzustellen ist.

Die Matrix N kann also im Demo-Modus als Werte der Kanten w_{ij} vom Knoten k_i zum Knoten k_j,nachdem das Gleichungssystem erzeugt worden ist,abgelesen werden.

Daraus erkennt man,dass der Algorithmus Graph reduzieren auf der Grundlage des Haupsatzes über absorbierende Markovketten arbeitet.

Die Matrix M kann natürlich durch Matrizeninversion gewonnen werden.Man kann sie jedoch auch folgendermaßen als Resultat des Algorithmus Graph reduzieren gewinnen:

Die Matrizen M und N beinhalten nur Elemente,die Relationen zwischen den inneren Knoten des Graphen beschreiben.

Lösche deshalb alle Randknoten des Markovgraphen,d.h. im obigen Beispiel die Knoten WZW und ZZZ.

Die verbleibenden inneren Knoten S,W,Z,WZ und ZZ werden jetzt der Reihe nach zusätzlich mit einem absorbierender Randknoten mit der Schlingenkante q über eine Kante mit der Bewertung Eins verbunden.Dann wird der Algorithmus Graph reduzieren mit der Option **Markovkette? nein** auf den vorhandenen Signalflussgraph angewendet (als Fluss wird dabei 1 vorgegeben),und jeweils durch fortgesetztes Löschen der Knoten der Signalfluss in den Knoten S,W,Z,WZ, und ZZ bestimmt.Dieses Verfahren muß insgesamt 5 Mal angewendet werden,so dass jeder der Knoten einmal zusätzlich mit einem Randknoten mit der Schlinge q über eine Kante mit dem Wert 1 verbunden ist.
Die dabei gewonnen Lösungen für den Signalfluss in den 5 Knoten bilden jeweils eine Spalte der Matrix M.

Begründung:

Die Gleichung

$(E-W)Y = E$

hat als Lösung die Matrix $Y= (E-W)^{-1}$.

Wenn also jeweils die Vektorgleichungen $(E-W)\vec{x}_i=\begin{pmatrix}0\\1\\.\\.\\0\end{pmatrix}$ (mit 1 in i.ter Zeile)

gelöst werden,bilden die Vektoren $\vec{x}_i$ die Spalten der Matrix Y (für i=1..m).

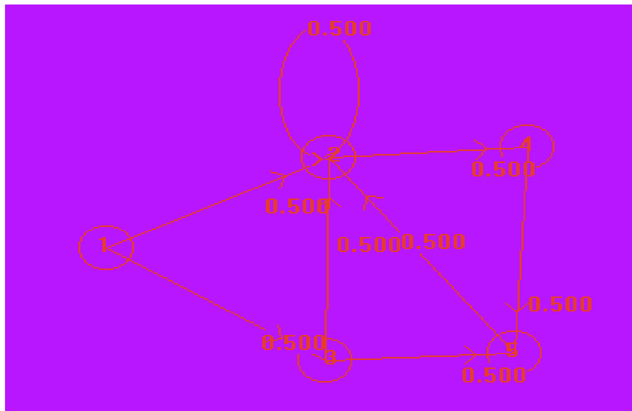
Dies bedeutet,dass der Fluss in einem (Signal-)Flussgraph gesucht wird bzw. ein Gleichungssystem gelöst werden muß,bei dem der der Flusswert c_i des i.ten Quellenknoten den Wert 1 besitzt (vgl. S. 344).

Zusatzaufgabe zur Anwendungsaufgabe C XII.5:

Ermittle die Fundamentalmatrix M der absorbierenden Markovkette von Anwendungsaufgabe XII.5 mit dem eben genannten Verfahren ohne Matrixinversion.

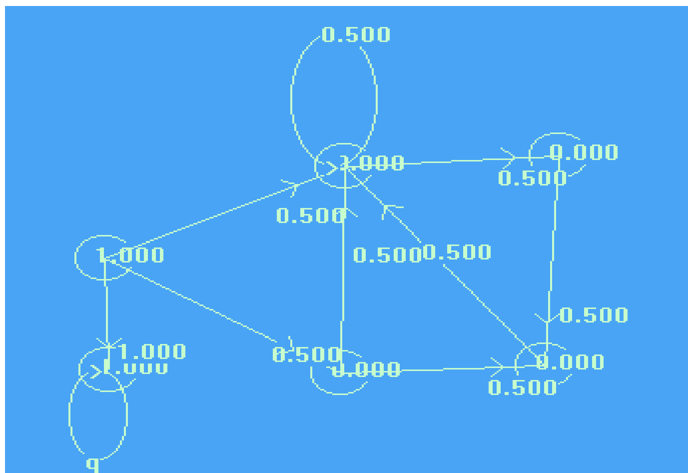
Lösung:

Die Randknoten des Graphen werden gelöscht und die inneren Knoten zur Bezeichnung der Matrixelemente folgendermaßen durchnummeriert:



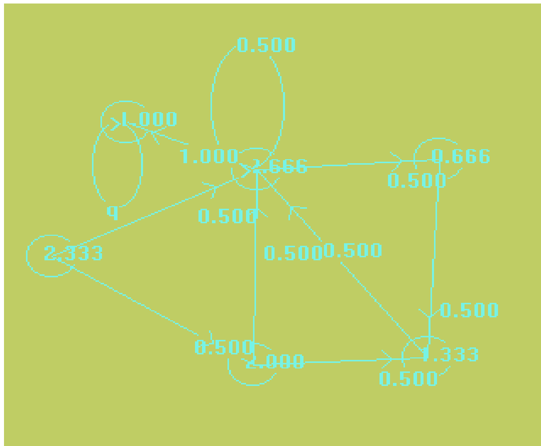
G070.gra

Anschließend wird jeder der Knoten mit einem neu erzeugten Quellknoten Q mit der Schlingenkante q verbunden,und die Signalflüsse durch sukzessives Löschen von Knoten (bis ein Knoten übrigbleibt) für alle Knoten ermittelt.



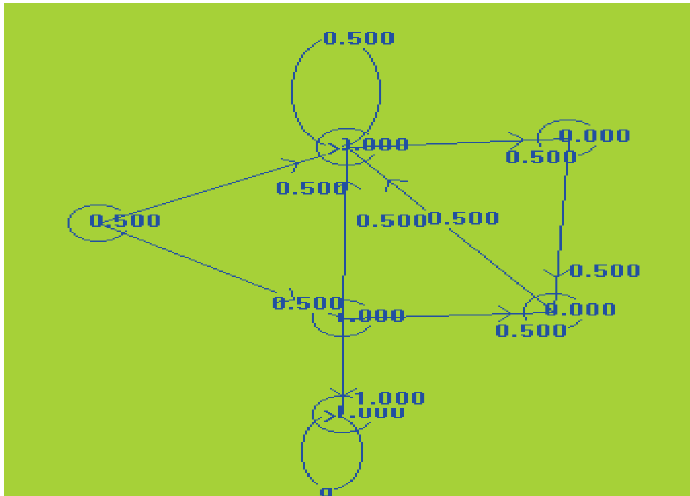
G071.gra

(fQ = 1.000)
fS = 1.000
f W = 0.000
f Z = 0.000
f WZ = 0.000
f Z = 0.000
f ZZ = 0.000



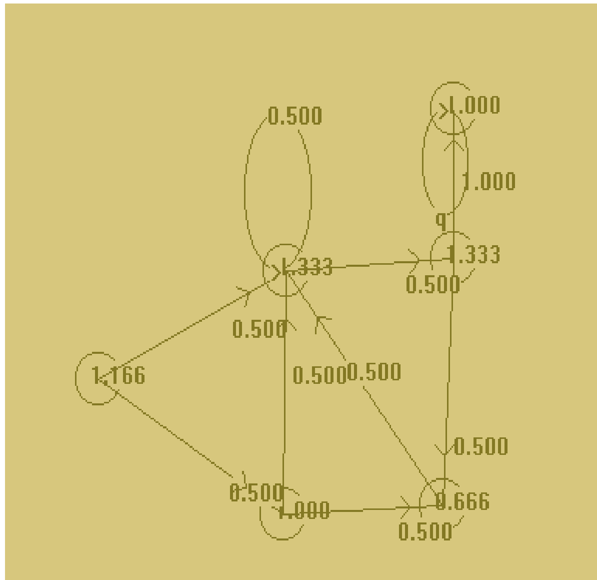
G072.gra

(fQ = 1.000)
fS = 2.3333
f W = 2.6666
f Z = 2.0000
f WZ = 0.6666
f ZZ = 1.3333



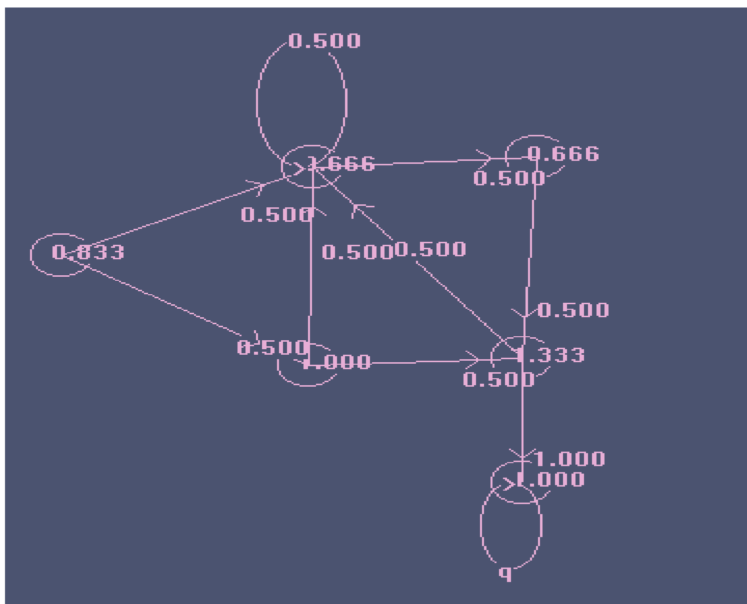
G073.gra

(fQ = 1.0000)
fS = 0.5000
f W = 0.0000
f Z = 1.0000
f WZ = 0.0000
f ZZ = 0.0000



G074.gra

(fQ= 1.0000)
fS = 1.1666
f W = 1.3333
f Z = 1.0000
f WZ = 1.3333
f ZZ = 0.6666



G075.gra

(fQ = 1.0000)
fS = 0.8333
f W = 0.6666
f Z = 1.0000
f WZ = 0.6666
f ZZ = 1.3333

Die Ergebnisse bedeuten jeweils die mittlere Anzahl der Besuche bzw. die mittlere Verweildauer in dem mit dem Schlingenknoten q verbundenen Knoten beim beim Start in dem Knoten,dem das Ergebnis zugewiesen ist.

Zum Beispiel ist die mittlere Verweildauer im Knoten ZZ beim Start in Z gleich 1.

Also lassen sich auf diese Weise alle mittleren Verweildauern aller innerer Knoten beim Start in einem anderen inneren Knoten mit dem Algorithmus Anwendungen/Graph reduzieren bestimmen.

Die Spalten der Fundamentalmatrix M bestehen gerade aus den obigen Ergebnissen:

$$M = \begin{pmatrix} 1 & 2,\overline{3} & 0,5 & 1,1\overline{6} & 0,8\overline{3} \\ 0 & 2,\overline{6} & 0 & 1,\overline{3} & 0,\overline{6} \\ 0 & 2 & 1 & 1 & 1 \\ 0 & 0,\overline{6} & 0 & 1,\overline{3} & 0,\overline{6} \\ 0 & 1,\overline{3} & 0 & 0,\overline{6} & 1,\overline{3} \end{pmatrix}$$

Die Summe der Zeilen dieser Matrix ergibt gerade die mittlere Schrittzahl bis zur Absorption in den Randknoten.Die Summe der Zeilen ergibt:

Knoten: S Mittlere Schrittzahl: $5.8\overline{3}$

Knoten: W Mittlere Schrittzahl: $4.\overline{6}$
Knoten: Z Mittlere Schrittzahl: 5.0

Knoten: WZ Mittlere Schrittzahl: $2.\overline{6}$

Knoten: ZZ Mittlere Schrittzahl: $3.\overline{3}$

Die mittlere Schrittzahl in den Randknoten ist immer gleich Null.

Dies entspricht den oben mit Hilfe des Algorithmus des Menüs Anwendungen/Markovkette (abs) ermittelten Ergebnissen.

Diese Ergebnisse lassen sich also auch auf diese Weise mittels des Algorithmus Graph reduzieren ermitteln.

Die Matrix R für das obige Beispiel lautet:

$$R = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0,5 & 0 \\ 0 & 0,5 \end{pmatrix}$$

Es gilt U=MR:

$$U = \begin{pmatrix} 0,58\overline{3} & 0,41\overline{6} \\ 0,\overline{6} & 0,\overline{3} \\ 0,5 & 0,5 \\ 0,\overline{6} & 0,\overline{3} \\ 0,\overline{3} & 0,\overline{6} \end{pmatrix}$$

(Die Spaltenvektoren ergeben sich auf Grund der Struktur der Matrix R als die Hälfte der letzten beiden Spaltenvektoren der Matrix M.)

Die erste Spalte der Matrix U enthält gerade die schon oben mittels der beiden Graphenalgorithmen berechneten Übergangswahrscheinlichkeiten in den Auswahlknoten WZW und die zweite Spalte jeweils die Komplementwahrscheinlichkeit als Übergangswahrscheinlichkeit in den Randknoten ZZZ.

Bemerkung:

Die Spalten von U als Lösungsvektor für die Übergangswahrscheinlichkeiten

lassen sich auch als Lösung der Gleichungssysteme $(E-W)\vec{x}_i=\begin{pmatrix}0\\1\\.\\.\\0\end{pmatrix}$ (i=1..m) mit

N=E-W durch Anwendung der Cramerschen Regel bestimmen.Dabei steht dann im Nenner des Terms die Determinante von N und im Zähler eine Determinante,in

der eine Spalte durch den Vektor $\begin{pmatrix}0\\1\\.\\.\\0\end{pmatrix}$ ersetzt ist.Diese Determinanten lassen

sich auch wieder auf dem Graphen als Summe bzw. als Differenz von disjunkten Kreisen des Graphen deuten (Schleifenprodukt).

Man nennt diese Regel die Formel von Mason.

(siehe z.B.: Lit 15, S.61ff.)

Zur Vervollständigung seien jetzt noch die Matrizen W und N=E-W angegeben:

$$W=\begin{pmatrix}0 & 0{,}5 & 0{,}5 & 0 & 0\\0 & 0{,}5 & 0 & 0{,}5 & 0\\0 & 0{,}5 & 0 & 0 & 0{,}5\\0 & 0 & 0 & 0 & 0{,}5\\0 & 0{,}5 & 0 & 0 & 0\end{pmatrix}$$

$$N=\begin{pmatrix}1 & -0{,}5 & -0{,}5 & 0 & 0\\0 & 0{,}5 & 0 & -0{,}5 & 0\\0 & -0{,}5 & 1 & 0 & -0{,}5\\0 & 0 & 0 & 1 & -0{,}5\\0 & -0{,}5 & 0 & 0 & 1\end{pmatrix}$$

Die inverse Matrix von N ist M,die hier (siehe oben) nicht durch algebraische Matrixinversion sondern durch einen Graphenalgorithmus gewonnen werden konnte.
(Das Verfahren eignet sich also auch dazu,spezielle Matrizen zu invertieren.)

Mit Hilfe des Algorithmus Graph reduzieren lassen sich also durch Ermitteln der Fundamentalmatrix für absorbierende Markovketten alle relevanten Ergebnisse für Markovketten gewinnen.

Als Anwendungsaufgabe für allgemeine Signalflussgraphen wird jetzt noch einmal die Aufgabe C XI.3 des letzten Kapitels aufgegriffen,deren dortiger Lösungsalgorithmus jetzt verständlich wird:

Aufgabe C XII.6 (wie C XI.3):

Löse folgende Aufgabe mit dem Algorithmus des Menüs Anwendungen/Graph reduzieren von Knotengraph als Signalflussgraph (mit der Option Markovgraph? nein),und erzeuge zuvor dafür einen geeigneten Graphen:

Zur Herstellung eines Produkts werden 4 Maschinen A,B,C und D benötigt.

Maschine A benötigt 8 Stunden vermindert um den 5. Teil der Zeit,die die Maschine B benötigt.

Maschine B benötigt das Doppelte der Zeit von Maschine A minus der eigenen Zeit plus der Zeit von Maschine D.

Maschine C benötigt das Doppelte der Maschine B vermindert um die Zeit von Maschine D.

Maschine D benötigt das Doppelte von Maschine A vermindert um die Hälfte der Zeit von Maschine D.

Wie lang muß jede der Maschinen laufen?

Lösung:

Es ergeben sich folgende Bedingungsgleichungen für die Zeiten der Maschinen:

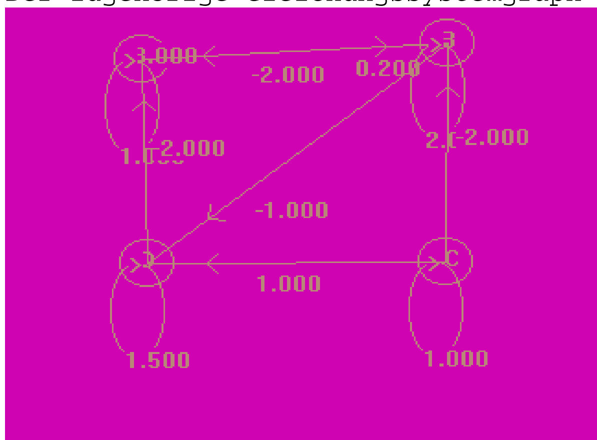
1) $t_A = 8 - \frac{1}{5} t_B$

2) $t_B = 2t_A - t_B + t_D$

3) $t_C = 2t_B - t_D$

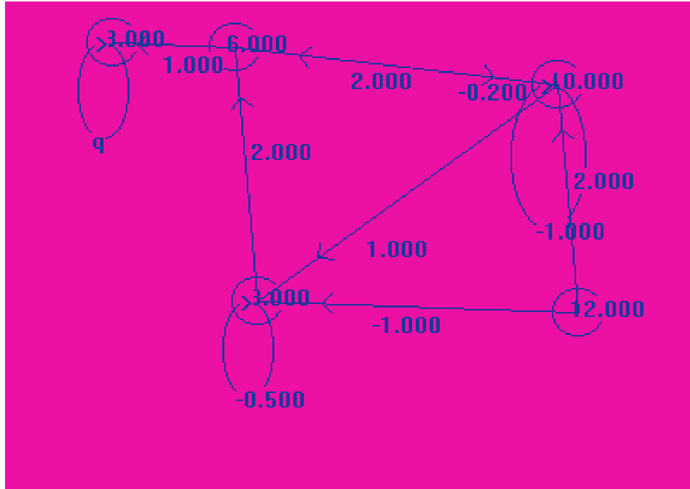
4) $t_D = 2t_A - \frac{1}{2} t_D$

Der zugehörige Gleichungssystemgraph ist (vgl. Kap. CXI):



G061.gra

Der Signalflussgraph (mit Lösungen,vgl. Aufg. CXI.3 Zusatz) ist:



G062.gra

Als Kennzeichnung für die (zusätzliche) Quelle wird jetzt ein Knoten mit einer Kantenschlinge mit dem Wert q verwendet.

Die Laufzeiten der Maschinen sind (Signalfluss):

f A = 6.000
f B = 10.00
f C = 12.000
f D = 8.000

Es sei nochmal darauf hingewiesen,dass die Pfeilrichtungen beim Signalflussgraphen eigentlich umgekehrt gerichtet sein sollten,und die umgekehrte Richtung der Kanten nur deshalb gewählt wurde,um den gleichen Algorithmus wie den zu den Markovketten (Graph reduzieren) benutzen zu können.

Mit dieser (leichten) Einschränkung lassen sich so jedoch beliebige Signalflussgraphen-Probleme lösen.

Im dritten Teil dieses Kapitel soll jetzt die Lösung von stationären Markovketten behandelt werden.

Stationäre Markovketten

Aufgabe C XII.7 (Einstiegsproblem):

In einer Bevölkerungsgruppe werde ein Raucher mit einer (konstanten) Wahrscheinlichkeit von 20% zu einem Nichtraucher,und ein Nichtraucher mit einer (konstanten) Wahrscheinlichkeit von 10% zu einem Raucher.Ist dann zu erwarten,dass die Raucher insgesamt aussterben,oder wie verteilt sich das Verhältnis zwischen beiden Gruppen nach längerer Zeit?
Angenommen wird,dass sich in der Bevölkerungsgruppe zu Beginn jeweils 50% Raucher und 50% Nichtraucher befinden.

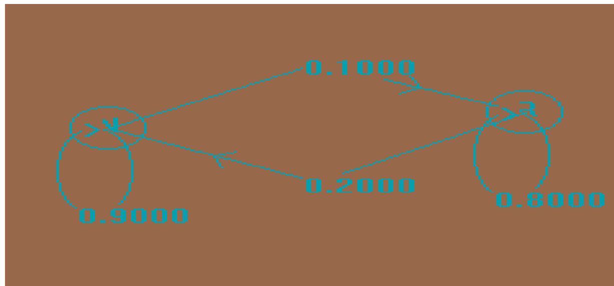
(Weiter unten wird gezeigt,dass die Lösung der Aufgabe von dieser Anfangsverteilung unabhängig ist.Deshalb muß sie beim Algorithmus Markovkette (stat) des Programms Knotengraph gar nicht erst eingegeben werden.)

Erzeuge einen geeigneten Graphen,und ermittle die Lösung mit Hilfe des Algorithmus des Menüs Anwendungen/Markovkette (stat) des Programm Knotengraph.

Lösung:

Es kann ein (stationärer) Markovgraph zur Lösung des Problems erstellt werden,wobei die beiden Knoten jeweils Nichtraucher (N) und Raucher (R) und die Bewertung der gerichteten Kanten die Übergangswahrscheinlichkeiten zwischen beiden Gruppen darstellen.Durch die Schlingenkanten wird der Anteil der Personen berücksichtigt,die jeweils Nichtraucher bzw. Raucher bleiben.

Graph:



G076.gra

Wie man sieht,gibt es in diesem Markovgraphen keine Randknoten bzw. absorbierenden Zustände.Er wird deshalb stationärer Markovgraph (stationäre Markovkette) genannt.

Fortsetzung der Aufgabe:weiter unten

Definition C XII.3:

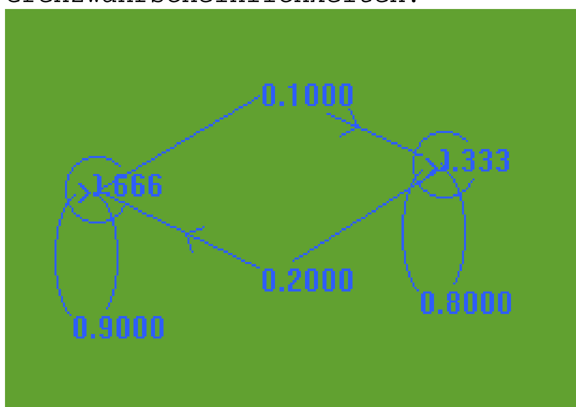
Eine stationäre Markovkette ist ein gerichteter,zusammenhängender Graph mit einer reellen Kantenbewertung,bei der der Wert jeder Kante größer oder gleich 0 und kleiner gleich 1 ist.Die Summe der Werte aller ausgehenden Kanten eines Knotens ist gleich 1.Die Knoten nennt man Zustände der Markovkette.Die Kanten der Markovkette mit ihrer Bewertung stellen die Übergangswahrscheinlichkeiten und Übergänge zwischen den Zuständen dar.

(In einer stationären Markovkette gibt es keine Randknoten und auch keinen Startknoten.)

Ziel ist es, aus einer Anfangsverteilung,die den Knoten des stationären Markovgraphen zugeordnet wird,eine Grenzverteilung zu berechnen,die sich einstellt,nachdem die Zufallsereignisse,die den Knoten durch die Wahrscheinlichkeitsbewertung der Kanten zugewiesen sind,im Grenzfall unendlich oft eingetreten sind.

Fortsetzung der Lösung von Anwendungsaufgabe XII.7:

Grenzwahrscheinlichkeiten:



Knoten: N Wahrscheinlichkeit: 0.666 Mittlere Schrittzahl: 1.499
Knoten: R Wahrscheinlichkeit: 0.333 Mittlere Schrittzahl: 3.000

Also sind im Grenzfall (nach längerer Zeit) $\frac{2}{3}$ der Bevölkerung Nichtraucher und $\frac{1}{3}$ Raucher.
Wie läßt sich ein solches Ergebnis gewinnen?

Mathematische Überlegungen:

Die Wahrscheinlichkeiten der Kanten des Graphen lassen sich wieder als Matrix W schreiben:

$$W=\begin{pmatrix} 0{,}9 & 0{,}1 \\ 0{,}2 & 0{,}8 \end{pmatrix}$$

Die Zufallsversuche,die zu den Kantenwahrscheinlichkeiten des Graphen gehören,werden jetzt laufend nacheinander ausgeführt (hier das Wechseln der Personen zwischen Nichtrauchern und Rauchern.Die Versuche seien mit 1,2,3...n,n+1.... nummeriert.

Mit $w_{ij}(n)$ und $w_{ij}(n+1)$ seien die (Übergangs-) Wahrscheinlichkeiten zwischen den Knoten k_i und k_j beim n.ten und n+1 .ten Zufallsversuch bezeichnet.

Dann gilt nach der 1. Mittelwertregel:

$w_{ij}(n+1)=\sum_{k=1}^{m} w_{ik}w_{kj}(n) = \sum_{k=1}^{m} w_{ik}(n)w_{kj}$,wobei m die Knotenanzahl des Graphen ist.

Die letzte Gleichung läßt sich als Matrixgleichung schreiben.
W(n) sei dabei die Matrix der der Wahrscheinlichkeiten $w_{ij}(n)$,entsprechend W(n+1).

$$W(n+1)=W(n)W=WW(n)$$

Daraus folgt: $W(n)=W^n$ (mit $W^0=E$)

Es sei $\vec{x}(0) = (x_1,x_2,...,x_m)$ eine Anfangsverteilung und
$\vec{x}(n) = (x_1(n),x_2(n),...x_m(n))$ die Verteilung nach n Zufallsversuchen.

Dann gilt: $\vec{x}(n) = \vec{x}(0)\, W^n$

Die gesuchte Lösung ist jetzt der Grenzwert:

$$\lim_{n\to\infty} \vec{x}(n) = \vec{x}(0) \lim_{n\to\infty} W^n$$

Also ist der Grenzwert $V = \lim_{n\to\infty} W^n$ gesucht.

Antwort auf die Frage,wann dieser Grenzwert existiert,gibt der folgende Satz:

Satz C XII.2 Hauptsatz für reguläre Markov-Ketten:

Eine Matrix W heißt regulär,wenn es ein n gibt,so dass W^n aus lauter positiven Elementen besteht.

Es sei W die Wahrscheinlichkeits-Matrix eines Markovgraphen.W muß dann regulär sein,da die Elemente alle positiv sind.

Für reguläre,stochastische Matrizen W gilt:

a)Der Grenzwert $V = \lim_{n\to\infty} W^n$ existiert.

V ist dann eine Matrix,in der die Zeilen-und Spaltensumen jeweils gleich 1 sind.

b)Die Zeilenvektoren $\vec{v} = (v_1, v_2, ..., v_m)$ der Matrix V sind alle gleich und enthalten nur positive Elemente.

Es gilt: $\vec{v}V = \vec{v}$,

d.h. $\vec{v}$ ist Eigen- bzw. Fixvektor von V.

Außerdem gilt: $\vec{v}W = \vec{v}$,

d.h. $\vec{v}$ ist Eigen- bzw. Fixvektor von W.

Der Vektor $\vec{v}$ ist eindeutig bestimmt.

c)Die mittlere Rückkehrzeit vom Knoten k_i in den Knoten k_i ist m_{ii} $(=m_i) = \frac{1}{v_i}$.

Beweis:

(nach Lit 15, S.182ff.)

zu a) und b):

Es sei j eine feste Spalte in W^n,und das maximale bzw. minimale Element in dieser Spalte sei M_n bzw. m_n .
Es gilt dann $M_n = \max_i w_{i,j}(n)$ und $m_n = \min_i w_{i,j}(n) = w_{xj}(n)$,wobei in der x-ten Zeile das minimale Element steht.
Dann ist:

$$M_{n+1} = \max_i \sum_k w_{i,k} w_{kj}(n) = \max_i (\sum_{k\neq x} w_{ik} w_{kj}(n) + w_{i,x} m_n) \leq$$

$$\max_i (M_n \sum_{k\neq x} w_{i,k} + w_{i,x} m_n) = M_n + \max_i (m_n - M_n) w_{i,x} = M_n - (M_n - m_n) \min_i w_{i,x} \leq$$

$M_n - k(M_n - m_n) \leq M_n$,wobei k das kleinste Element der Matrix W ist.

Analog läßt sich zeigen:

$$m_{n+1} \geq m_n + k(M_n - m_n) \geq m_n$$

Also ist M_n monoton wachsend, und m_n monoton fallend.Subtraktion der beiden Ungleichungen ergibt:

$$M_{n+1} - m_{n+1} \leq (1-2k)(M_n - m_n)$$

Also gilt:

$M_n - m_n \le (1-2k)^{n-1}$ für n größer gleich 1.

Da k als kleinstes Element von W kleiner als 0,5 sein muß,gilt $0<1-2k<1$ und $M_n - m_n$ ist Nullfolge.
Also haben alle Folgenelemente w_{ix} der x.ten Spalte denselben Grenzwert v_x (der existiert),und die Zeilen der Matrix V sind alle gleich.

Ist $\vec{v}$ gleich der i.ten Zeile von V so gilt $\vec{v}V = \vec{v}$,denn

die Summe der Elemente des Vektors $\vec{v}$ ergibt 1.

Die Beziehung $\vec{v}W = \vec{v}$ ergibt sich unmittelbar aus Beziehung $\vec{v}V = \vec{v}$,denn:

$\vec{v}V = \vec{v} \Rightarrow \vec{v}(VW) = \vec{v}W = \vec{v}$,da V=VW ist.Die letztere Beziehung folgt aus $W^{n+1} = W^n W$ für $n \to \infty$: V=VW.

Es sei $\vec{x}(0) = (x_1, x_2, \ldots x_m)$ eine beliebige Anfangsverteilung.

Dann ist

$\vec{x}(0)\,V = (v_1(x_1+x_2+..x_m),\ldots,v_m(x_1+\ldots+x_m)) = (v_1, v_2, \ldots v_m) = \vec{v}$,

weil die Summe der x_i 1 ist, und die Matrix V in allen Spalten aus gleichen Elementen besteht (alle Zeilen sind gleich!).

Also hängt die Grenzverteilung $\vec{v} = \vec{x}_\infty = \lim\limits_{n\to\infty} \vec{x}(n)$ nicht von der Anfangsverteilung ab und ist stets gleich dem Zeilenvektor der Matrix V.

Falls $\vec{v}\,'$ ein weiterer Vektor ist,für den $\vec{v}\,'V = \vec{v}\,'$ gilt,so folgt auch: $\vec{v}\,'V = \vec{v}$,da der Ergebnisvektor nicht von der Anfangsverteilung abhängt.Also gilt $\vec{v} = \vec{v}\,'$ und der Vektor $\vec{v}$ ist eindeutig bestimmt.

Außerdem gilt $v_i = \sum_j v_j v_{ji} \ge k > 0$.Also sind die Elemente von V alle positiv.

zu c):

Es sei $m_{i,j}$ die mittlere Schrittzahl,um von k_i nach k_j zu kommen.

Nach der 2. Mittelwertregel gilt:

$$m_{i,j} = 1 + \sum_{k \ne j} w_{ik} m_{kj}$$

Die Gleichung läßt sich mittels der nxn Matrizen M,D und E schreiben als:

$M = E + W(M-D)$

M besteht aus den Elementen $m_{i,j}$,E ist die Einheitsmatrix, und D ist eine Matrix,deren Diagonalenelemente die Elemente $m_{i,i}$ sind und sonst aus lauter Nullen besteht.

Aus obiger Gleichung folgt:

$M=E+WM-WD$

Durch Multiplikation von links mit $\vec{v}$ folgt:

$\vec{v}D=\vec{v}E$,da $\vec{v}W=\vec{v}$ gilt.

Also gilt für ein Diagonalenelement: $v_{ii}m_{ii}=1$

Daraus folgt $m_{ii}=\frac{1}{v_{ii}}$

oder einfacher geschrieben,da die Matrix V nur aus gleichen Zeilen besteht:

$m_{ii}\ (=m_i)=\frac{1}{v_i}$

Mathematische Überlegungen zur Lösung der gestellten Aufgabe:

Eine Möglichkeit,die gestellte Aufgabe zu lösen,geht von der Gleichung $\vec{v}W = \vec{v}$ aus und löst das zugehörige Eigenwertproblem zum Eigenwert 1 mit der Nebenbedingung,dass die Summe der Komponenten des Vektors $\vec{v}$ gleich 1 sein muß,da diese Summe die Wahrscheinlichkeitssumme der Grenzwahrscheinlichkeit ist.

Mit der Matrix $W=\begin{pmatrix}0,9 & 0,1\\ 0,2 & 0,8\end{pmatrix}$ ergibt sich dann das Gleichungssystem

$\vec{v}(W-E)=0$, wobei E die Einheitsmatrix ist.

Die Gaußmatrix $\begin{pmatrix}-0,1 & 0,2 & 0\\ 0,1 & -0,2 & 0\end{pmatrix}$ läßt sich umformen in $\begin{pmatrix}1 & -2 & 0\\ 0 & 0 & 0\end{pmatrix}$ und ergibt die Lösung $v_1=2v_2$ mit $v_2=r\in\Re$ beliebig.

Aus der Nebenbedingung $2r + r= 1$ folgt $r=\frac{1}{3}$

Also ist die Lösung: $v_1=\frac{2}{3}$ und $v_2=\frac{1}{3}$.

Eine andere Möglichkeit zur Lösung des oben genannten Problems ist die Ermittlung der Grenz-Matrix V,deren jeweils gleiche Zeilen die gesuchte Grenz-Wahrscheinlichkeitsverteilung ist.

Ausgerechnet werden dazu die Matrizen W^1, W^2, W^3, W^4, W^5 und W^{40}:

$$\begin{bmatrix}0.9 & 0.1\\ 0.2 & 0.8\end{bmatrix}$$

$$\begin{bmatrix}0.781 & 0.210\\ 0.438 & 0.562\end{bmatrix}$$

$$\begin{bmatrix}0.83 & 0.17\\ 0.34 & 0.66\end{bmatrix}$$

$$\begin{bmatrix} 0.7467 & 0.2533 \\ 0.5066 & 0.4934 \end{bmatrix}$$

$$\begin{bmatrix} 0.722690 & 0.277309 \\ 0.554619 & 0.445380 \end{bmatrix}$$

$$\begin{bmatrix} 0.676082 & 0.323917 \\ 0.647834 & 0.352165 \end{bmatrix}$$

$$\begin{bmatrix} 0.666666 & 0.333333 \\ 0.666666 & 0.333333 \end{bmatrix}$$

Wie man sieht gilt $V=W^{40}$ auf 6 Stellen genau,und die oben mittels des Algorithmus des Menüs Anwendungen/stationäre Markovkette des Programm Knotengraph gewonnenen Ergebnisse sind bestätigt.

Die Multiplikation der Matrix W mit W^n ist äquivalent zur Anwendung der 1. Mittelwertregeln nach der Ausführung eines erneuten Zufallsversuch.Angewendet auf das Beispiel Raucher und Nichtraucher bedeutet dies,dass sich Personen gemäß den Kantenwahrscheinlichkeiten von einem Knoten zum anderen bewegen.

Die Matrizenmultiplikation ist allerdings eine recht mühsame Angelegenheit und konvergiert evtl. nur langsam,wie das Beispiel zeigt.Eine bessere algorithmische Möglichkeit wird durch die folgenden Überlegungen gegeben:

Wenn man statt der Personen des obigen Beispiels (Raucher und Nichtraucher) wiederum Spielsteine gemäß den Kantenwahrscheinlichkeiten bewegt,wobei zu Anfang eine solche Spielsteineverteilung auf den Knoten gewählt wird,dass man sofort ganze Steine von jedem Knoten aus zu den Nachbarknoten unter Beachtung der Wahrscheinlichkeiten als Brüche ziehen kann,läßt sich die Ermittlung des stationären Verteilungsvektor $\vec{V}$ nach A.Engel (Lit 15, S.215ff.) wiederum als Wahrscheinlichkeitsspiel (Wahrscheinlichkeitsabakus) auffassen:

Der Algorithmus stationäre Markovkette

Verbale Beschreibung:

Die Wahrscheinlichkeitsbewertung der Kanten sei in Form von (gekürzten) Brüchen gegeben.Es sei N_j der Hauptnenner der Nenner $N_{j1}, N_{j2}, ... N_{jk}$ der Brüche der Kantenwahrscheinlichkeiten der ausgehenden Kanten eines inneren Knotens k_j des Markovgraphen.Jedem Knoten wird dann eine Anzahl (Spiel-)Steine zugeordnet.

1)Setze die Zahl der Spielsteine jedes Knotens k_j zu Beginn auf den Wert N_j.Dies bedeutet,dass man von allen Knoten aus ziehen kann.Die Summe der Spielsteine ergibt die Gesamtzahl der Spielsteine.

Spielsteine sollen von einem Knoten zum anderen längs der Kantenrichtungen gezogen werden,wobei man den Graphen als Spielbrett aufzufassen hat.(Die Steine befinden sich stets in den Knoten und werden längs der Kanten gezogen.)

Wichtige Bedingung ist,dass nur ganze Steine gezogen werden können.Dies ist nur dann möglich,wenn sich in einem Knoten k_j genau ein Vielfaches von N_j Steinen befinden.

2)Die Knoten k_j,deren Steinezahl unter einem Vielfachen von N_j liegt,werden auf das nächste Vielfache von N_J Steine aufgefüllt.Alle neu hinzugefügte Spielsteine werden zur Gesamtzahl Spielsteine hinzugezählt.

3)Die Verteilung der Spielsteine auf den Knoten wird als momentane Verteilung gespeichert.

4)Man zieht nun gleichzeitig von jedem Knoten aus,solange die Knoten noch mit der zum Ziehen mindestens benötigten Zahl Steine versehen sind.Gleichzeitig bedeutet,dass die zu anderen Knoten gezogenen Steine erst als hinzugekommene Steine der Nachbarknoten berücksichtigt und hinzuaddiert werden,nachdem das Ziehen bei allen Knoten des Graphen beendet ist,weil auf Grund zu niedriger Steinezahlen nicht mehr gezogen werden kann.

Die Steine werden dabei als ganzahliger Bruchteil $N_j \cdot \frac{1}{N_{jl}}$ längs der l.ten ausgehenden Kante auf die Nachbarzielknoten des Knotens k_l verteilt.

5)Die neue Verteilung der Steine auf den Knoten wird mit der vorigen Verteilung verglichen.Wenn es dieselbe Verteilung ist,gehe nach 6).

Sonst setze den Algorithmus bei 2) fort.

6)Als Ergebnis kann der Fixvektor berechnet werden.Seine Komponenten ergeben sich jeweils als Quotient aus der Anzahl der Steine in den einzelnen Knoten und der Gesamtzahl der Spielsteine.

Die mittlere Schrittzahl (Rückkehrzeit) ergibt sich als Kehrwert dieser Zahl.

Beende dann den Algorithmus.

Der Algorithmus hat wiederum den Vorteil,dass er exakte Ergebnisse liefert und ein definiertes Ende hat.

Bezüglich der Genauigkeit des Ergebnisses bei der Darstellung der Brüche als Realzahlen gilt das beim Algorithmus absorbierende Markovkette Gesagte.

Aufgabe CXII.7:(Fortsetzung)

Beobachtung und problemorientierte Erarbeitung:

Beobachte den Ablauf des Algorithmus Anwendungen/(stat) Markovkette des Programms Knotengraph an Hand des Graphen G076.gra im Demo- oder Einzelschrittmodus,und notiere die Spielsteine-Verteilung nach jedem Schritt in einer Tabelle.Erkläre anschließend die Verteilung an Hand der Spielregeln.Spiele das Spiel dann per Hand nach.(Verfahre analog auch bei anderen Graphen,z.B. G077.gra siehe dazu die nächste Aufgabe)

Tabelle:

Zu G077.gra:

A	B	C	D
10	10	10	10
0	20	10	10
10	30	10	10
10	40	10	20
20	50	10	20
20	60	20	20
20	70	20	30
20	80	20	30
20	90	20	30

Zu G076.gra:

N	R
10	5
10	5

Lösungen: $\frac{2}{3}$; $\frac{1}{3}$ bzw. $\frac{1}{8}$; $\frac{9}{16}$; $\frac{1}{8}$; $\frac{3}{16}$

(Summen:15 bzw. 160)

Der Quellcode im Auszug:

Die Datenstruktur von Knoten und Kanten ist dieselbe wie bei der Anwendung absorbierenden Markovkette.Benutzt werden jetzt bei TMarkovknoten neu die Felder VorigeAnzahl und Delta zur Speicherung der Steine der vorigen Zugfolge und der bei der momentanen Zugfolge neu hinzuzufügenden Steine (siehe weiter oben).

TMarkovstatgraph leitet sich durch Vererbung von TInhaltsgraph ab.Der Datentyp enthält keine neuen Datenfelder:

```
TMarkovstatgraph = class(TInhaltsgraph)
  public
    constructor Create;
    function Fehler:Boolean;
    function AnzahlRandknoten:Integer;
    procedure ZieheSteinestat(Ausgabe:TLabel;Gesamtzahl:Extended;Flaeche:TCanvas);
    procedure LadeKnotenAnzahlmitMinimum;
    procedure BestimmeMinimum(Genauigkeit:Integer);
    procedure ErhoeheAnzahlumDelta;
    procedure SpeichereVerteilung;
    procedure SummiereAnzahlstat(var Gesamtzahl:Extended);
    procedure BestimmeWahrscheinlichkeit(Gesamtzahl:Extended);
    procedure ErzeugeAusgabe(var S:string);
    function VorigeAnzahlgleichAnzahl(Gesamtzahl:Extended):Boolean;
    procedure LadeKnotenAnzahlmitkleinsterZahl;
    procedure Markovstat(Ausgabe1,Ausgabe2:Tlabel;var Gesamtzahl:Extended;
    var Sliste:TStringlist;Flaeche:TCanvas;Genauigkeit:Integer);
    procedure Markovkettestat(Flaeche:TCanvas;Ausgabe1,Ausgabe2:TLabel;
    var Sliste:TStringlist);
  end;
```

Die Methode Markovstat führt die Schritte 1) bis 6) des oben beschriebenen Algorithmus durch.Übergeben wird dabei die Variable Gesamtzahl als Referenzparameter,die zunächst mit 0 belegt ist.

```
procedure TMarkovstatgraph.Markovstat(Ausgabe1,Ausgabe2:Tlabel;var Gesamtzahl:Extended;
var Sliste:TStringlist;Flaeche:TCanvas;Genauigkeit:Integer);
var  Steine,Schrittzahl:Extended;
     St:string;
begin
  Steine:=0;                                                                     1)
  Schrittzahl:=0;
  BestimmeMinimum(Genauigkeit);                                                  2)
  LadeKnotenAnzahlmitMinimum;                                                    3)
  SummiereAnzahlstat(Gesamtzahl);                                                4)
  BestimmeWahrscheinlichkeit(Gesamtzahl);                                        5)
  ErzeugeAusgabe(St);                                                            6)
  Ausgabe2.Caption:=St;                                                          7)
  Ausgabe2.Refresh;                                                              8)
  repeat                                                                         9)
     LadeKnotenAnzahlmitkleinsterZahl;                                          10)
     SpeichereVerteilung;                                                       11)
     SummiereAnzahlstat(Gesamtzahl);
     BestimmeWahrscheinlichkeit(Gesamtzahl);                                    12)
     ErzeugeAusgabe(St);
     Ausgabe2.Caption:=St;
     Ausgabe2.Refresh;
     ZieheSteinestat(Ausgabe1,Gesamtzahl,Flaeche);                              13)
     ErhoeheAnzahlumDelta;                                                      14)
  until VorigeAnzahlgleichAnzahl(Gesamtzahl);                                   15)
  SummiereAnzahlstat(Gesamtzahl);
  BestimmeWahrscheinlichkeit(Gesamtzahl);                                       16)
  ErzeugeAusgabe(St);
  ShowMessage('Wahrscheinlichkeitsverteilung: '+St);
end;
```

Bei 1) werden Variablen initialisiert,u.a. Steine mit Null.Bei 2) wird das Feld (Steine-)Minimum_ mit der minimalen Anzahl Steine geladen,bei der man ziehen kann.(Diese Zahl hängt von der vorgegebenen Genauigkeit ab.) Die Methode BestimmeMinimum wurde schon weiter oben erläutert.Bei 3) wird das Feld (Steine-)Anzahl jedes Knoten mit dem vorher bestimmten Minimum geladen und bei 4) wird die Gesamtzahl aller den Knoten zugeordneter Steine bestimmt.

Die Zeilen 6) bis 8) erzeugen die Ausgabe im Ausgabelabel2.

Die Knoten werden mit der kleinsten Steinezahl N_j,bei der man ziehen kann,wieder aufgeladen (10) und die Verteilung der Steine in den Knoten wird im Feld VorigeAnzahl_ zwischengespeichert (11).Die vorläufige Wahrscheinlichkeit in den Knoten,wie sie sich aus der momentanen Steineverteilung ergibt,wird bestimmt (12).Bei 13) werden die Steine von den Knoten aus zu den Nacharknoten längs der Kanten gezogen.

Erst nachdem alle Züge ausgeführt wurden,werden bei 14) die im Feld Delta gespeicherten Steine zum Feld Anzahl hinzuaddiert.

Bei 9) beginnt eine Repeat-Schleife,die alle schon beschriebenen Anweisungen (Anweisungen 10 bis 14) solange wiederholt ausgeführt,bis bei 15) die vorige Anzahl der Steine mit der momentanen Anzahl bei jedem Knoten übereinstimmt (stationäre Verteilung).

Bei 11) müssen dem Feld SteineAnzahl jedes Knoten soviele Steine hinzugefügt werden,dass das Minimum der Steine bei jedem Knoten erreicht wird.

Zum Schluss wird bei 16) wieder die Gesamtzahl aller Steine in den Knoten bestimmt,und die stationäre Wahrscheinlichkeit für jeden Knoten berechnet, sowie im Feld Anzeige (zwecks Anzeige beim Mausklick auf die Knoten) gespeichert und die Verteilung in einem Ausgabefenster (mittels ShowMessage) ausgegeben.

Die Methode Markovkettestat prüft,ob es sich um eine Markovkette handelt, und ruft dann die Methode Markovstat auf.

```
procedure TMarkovstatgraph.Markovkettestat(Flaeche:TCanvas;Ausgabe1,Ausgabe2:TLabel;
var Sliste:TStringlist);
var Fehlerstri,Str:String;
    Index,Genauigkeit:Integer;
    Gesamtzahl:Extended;
    Eingabe:Boolean;
begin
  if Fehler then                                                                17)
  begin
    ShowMessage('Zuerst Fehler beheben!');
    exit;
  end;
    Str:='5';
  Repeat
    Eingabe:=Inputquery('Eingabe Genauigkeit:','Genauigkeit (1 bis 5 Stellen):',Str);
    if StringtoInteger(Str)<=0
    then
      Genauigkeit:=5
    else
      Genauigkeit:=abs(StringtoInteger(Str));
  until Eingabe and (Genauigkeit<6);
  Ausgabe1.Caption:='Berechnung läuft...';
  Ausgabe1.Refresh;
  Gesamtzahl:=0;                                                                18)
  Markovstat(Ausgabe1,Ausgabe2,Gesamtzahl,Sliste,Flaeche,Genauigkeit);         19)
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      with TMarkovknoten(Knotenliste.Knoten(Index)) do                         20)
      begin
        Ergebnis:=Wert+chr(13)+'Wahrscheinlichkeit: '+
        RundeZahltostring(Wahrscheinlichkeit,Knotengenauigkeit)
         +chr(13)+'Mittlere Schrittzahl: '+
        RundeZahltostring(MittlereSchrittzahl,Knotengenauigkeit);
        Sliste.Add('Knoten: '+Wert+' Wahrscheinlichkeit: '+
        RundeZahltostring(Wahrscheinlichkeit,Knotengenauigkeit)
        +' Mittlere Schrittzahl: '+
        RundeZahltostring(MittlereSchrittzahl,Knotengenauigkeit));
      end;
  Ausgabe1.Caption:='                                             ';
  Ausgabe1.Refresh;
 Ausgabe2.Caption:='                                             ';
 Ausgabe2.Refresh;
end;
```

Bei 17) wird geprüft,ob die Wahrscheinlichkeitssumme der Bewertung aller ausgehenden Kanten jedes Knoten des Graphen gleich 1 ist.Wenn nicht,wird die Ausführung des Algorithmus abgebrochen.Bei 18) wird die Gesamtzahl auf

0 gesetzt und bei 19) die oben besprochenen Methode Markovstat aufgerufen.Schließlich werden bei 20) die stationäre Wahrscheinlichkeitsverteilung und die mittlere Rückkehrschrittzahl für jeden Knoten zur Ausgabe im Ausgabefenster in der Stringliste SListe gespeichert.Der verständliche Quelltext der im obigen Text erwähnten übrigen hier nicht besprochenen Methoden kann im Listing des Programms Knotengraph im Anhang nachgelesen werden.

Aufgabe C XII.8:

Im Land Utopia gibt es vier Parteien A,B,C und D,und es wird jedes Jahr neu gewählt.Zwischen den Parteien bestehen die in der folgenden Tabelle angegebenen Wählerwanderungen (Abgabe der Wähler in Prozent von Partei X zu Partei Y):

Partei:	A	B	C	D
A	60	20	10	10
B	0	80	10	10
C	10	40	30	20
D	20	20	10	50

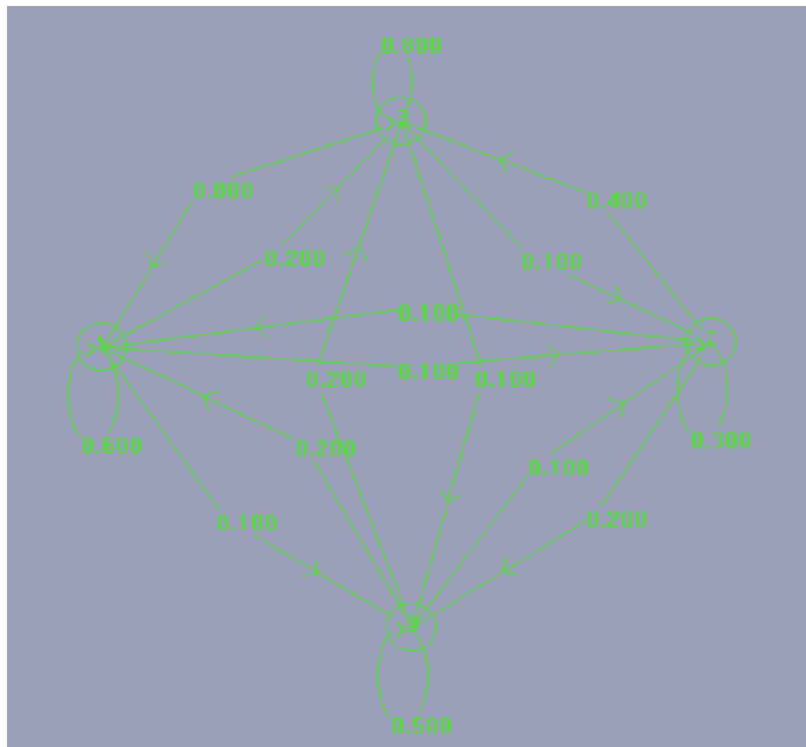
Wie wird die Sitzverteilung (in Prozent) nach vielen Jahren im Parlament von Utopia sein,wenn der momentane Wählertrend konstant anhält?

a)Erstelle einen geeigneten Markovgraphen.

b)Löse die Aufgabe mit Hilfe des Algorithmus Anwendungen/Markovkette (stat) des Programms Knotengraph,und benutze dabei den Demo-oder Einzelschrittmodus,um den Algorithmus zu verfolgen.Nach wieviel Jahren im Durchschnitt kehrt ein Wähler einer (neuen) Partei wieder zu seiner alten Partei als Wähler zurück?

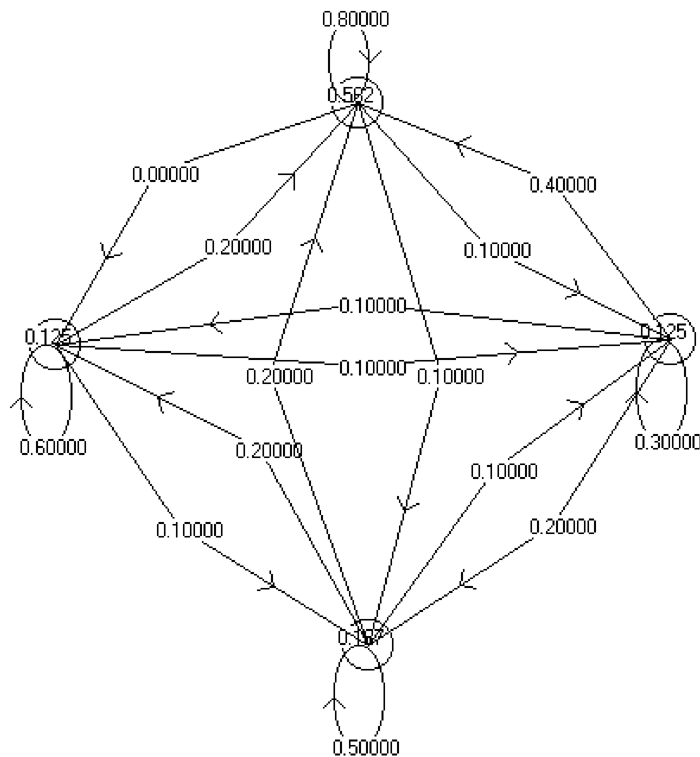
Lösung:

a)Graph:



G077.gra

b)



G077.gra

Knoten: A Wahrscheinlichkeit: 0.125 Mittlere Schrittzahl: 7.993

Knoten: B Wahrscheinlichkeit: 0.562 Mittlere Schrittzahl: 1.778

Knoten: C Wahrscheinlichkeit: 0.125 Mittlere Schrittzahl: 7.998

Knoten: D Wahrscheinlichkeit: 0.187 Mittlere Schrittzahl: 5.332

Also ist die Sitzverteilung nach vielen Jahren im Parlament:

Partei A: 12,5% Partei B: 56,2% Partei C: 12,5% Partei D: 18,7%

Die oben genannten mittleren Schrittzahlen geben an,nach wie vielen Jahren die Partei von einem Wähler jeweils im Durchschnitt wiedergewählt wird.

Zum Schluss dieses Kapitels seien noch einige Beispiele dargestellt,die zeigen,dass auch einfache Wahrscheinlichkeitsprobleme des Mathematikunterrichts der Sekundarstufe I,die auf der Berechnung von Pfadwahrscheinlichkeiten in Wahrscheinlichkeitsbäumen beruhen als auch die Bayesche Regel betreffen,mit dem Programm Knotengraph veranschaulicht werden können und lösbar sind:

Aufgabe C XII.9:

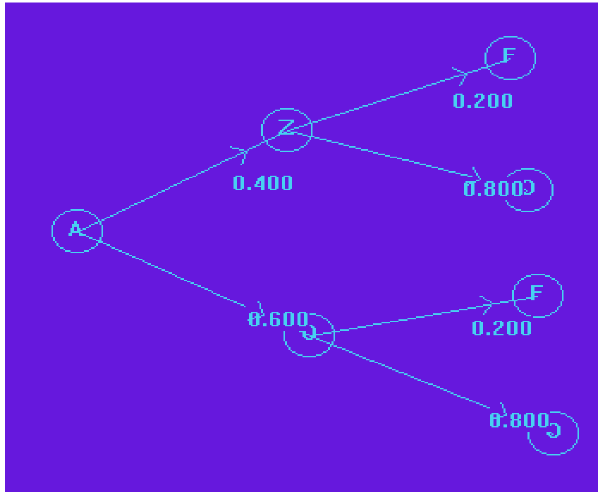
In einem Gefäß (Urne) befinden sich eine Anzahl von Zitronen-und Orangenbonbons.Die Bonbons sind jeweils gefüllt oder ungefüllt.Die Wahrscheinlichkeit ein Zitronenbonbon zu ziehen,beträgt 40% und ein Orangenbonbon 60%.Von den Bonbons sind jeweils 20% gefüllt und 80% ungefüllt.

Wie groß ist die Wahrscheinlichkeit jeweils ein gefülltes Zitronenbonbon,ein ungefülltes Zitronenbonbon,ein gefülltes Orangenbonbon oder ein ungefülltes Orangenbonbon zu ziehen?

Die Lösung beruht auf dem Algorithmus der Tiefensuche,wie er in Kapitel C II beschrieben wurde.

Lösung:

Erzeuge den folgenden Graphen:



G078.gra

Z=Zitronenbonbon,O=Orangenbonbon,F=gefüllt,O=ohne Füllung

Ermittle mit dem Algorithmus des Menüs Pfade/Tiefer Baum (Programm Knotengraph) die Pfadprodukte der Pfade (Produktregel):

A Z Summe: 0.400 Produkt: 0.400

A Z F Summe: 0.600 Produkt: 0.080 gefülltes Zitronenbonbon

A O Summe: 0.600 Produkt: 0.600

A O F Summe: 0.800 Produkt: 0.120 gefülltes Orangenbonbon

A Z O Summe: 1.200 Produkt: 0.320 ungefülltes Zitronenbonbon

A O O Summe: 1.600 Produkt: 0.600 ungefülltes Orangenbonbon

Als weitere Aufgaben dieser Art sind z.B. Veranschaulichungen der Summenregel oder der Bayeschen Regeln denkbar:

Aufgabe C XII.10:

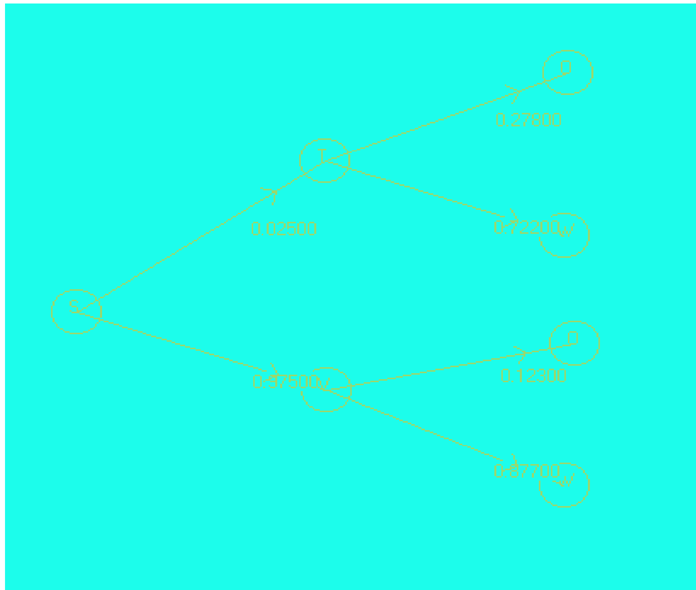
(Vgl. Lit 66,S. 226 ff,dort auf konventionelle Weise gelöst.)

Von den Autounfällen,bei denen nicht nur Sachschäden auftreten,ereigneten sich in der Bundesrepublik im Jahre 1990 2,5% mit getöteten Personen und 97,5% mit Verletzten.

Von den Unfällen in Ostdeutschland ereigneten sich 12,3% mit Verletzten und 27,8% mit Getöteten.

Wie groß ist die Wahrscheinlichkeit für das Auftreten eines Unfalls mit Personenschäden in Ostdeutschland bzw. Westdeutschland?

Das Problem läßt sich als zweistufiger Zufallsversuch auffassen und führt auf folgendes Baumdiagramm (O=Ost,W=West,T=Getötete,V= Verletzte,S=Start):



G079.gra

Wieder lassen sich die Pfadwahrscheinlichkeiten erzeugen (Programm Knotengraph/Menü Pfade/Alle Pfade (von S)):

S T Summe: 0.02500 Produkt: 0.02500

S T O Summe: 0.30299 Produkt: 0.00695

S T W Summe: 0.74700 Produkt: 0.01804

S V Summe: 0.97500 Produkt: 0.97500

S V O Summe: 1.09799 Produkt: 0.11992

S V W Summe: 1.85200 Produkt: 0.85507

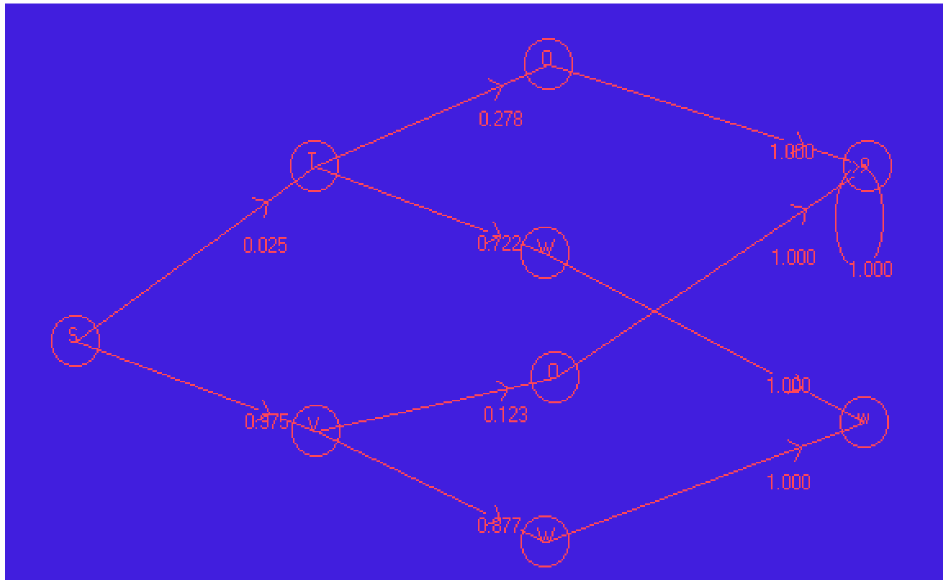
Die Produkte der Pfade mit zwei Kanten (zu den Blättern) sind jeweils die Wahrscheinlichkeiten für die vier Ereignisse Getötete bzw. Verletzte in Ostdeutschland bzw. in Westdeutschland.

Nun ist die Wahrscheinlichkeit für das Auftreten eines Unfalls mit Personenschäden in Ostdeutschland bzw. Westdeutschland zu ermitteln.

Lösung:

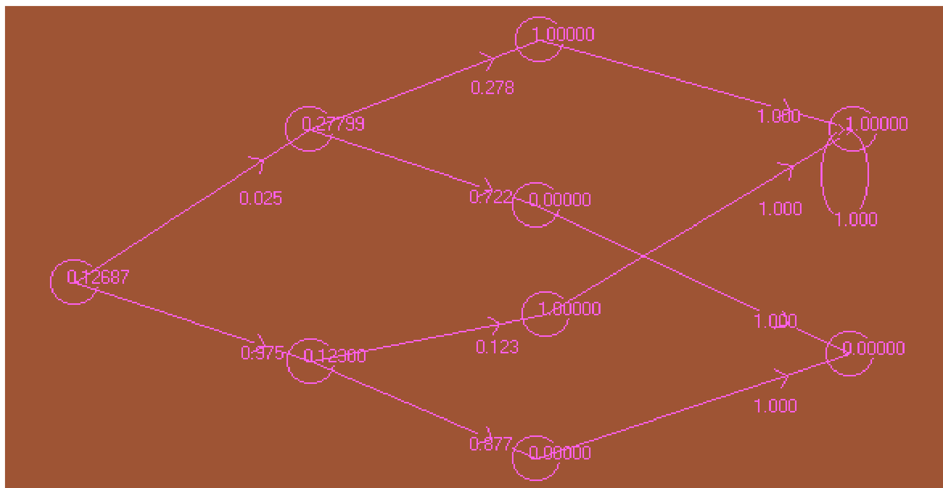
Dazu ist die Summenregel anzuwenden.

Ein entsprechender Graph ergibt sich folgendermaßen durch Zusammenfassung von Ereignissen (o faßt alle Ostdeutschen Unfälle,w alle Westdeutschen Unfälle zusammen):



G080.gra

Durch Anwendung des Algorithmus des Menüs Markovkette (abs) des Programms Knotengraph mit dem Auswahlknoten o ergibt sich für die Übergangswahrscheinlichkeiten:



G080.gra

Als Wahrscheinlichkeit für einen Unfall in Ostdeutschland bzw. in Westdeutschland (mit Personenschaden) ergibt sich also (gerundet): 0,127 bzw. 0,873

Zusatzfrage:

Ist der Anteil der Toten in Ostdeutschland höher als in Westdeutschland?

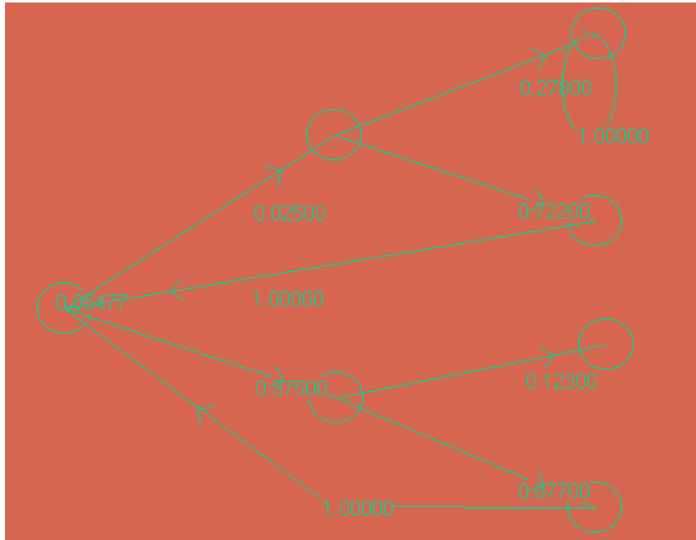
Zur Lösung der Aufgabe ist die Bayesche Regel anzuwenden und eine Umkehrung des obigen Baumdiagramms zu erstellen.(vgl.Lit 66, S. 230)

Lösung mit Hilfe von Knotengraph:

Mit Hilfe des Algorithmus Graph reduzieren des Programms Knotengraph läßt sich ausgehend von dem obigen Baumdiagramm,indem die beiden Blätterknoten W zusätzlich durch eine gerichtete Kante mit der Bewertung 1 mit dem Startknoten S verbunden werden und die beiden Blätterknoten mit O jeweils zum Quellknoten mit dem Fluss 1 gemacht werden,unter Auffassung des so entstan-

denen Graphen als Signalflussgraph,der Fluss durch den Startknoten S jeweils zu (gerundet) 0,055 bzw. 0,120 bestimmen.

Graph zu w=0.055 (0.05477):

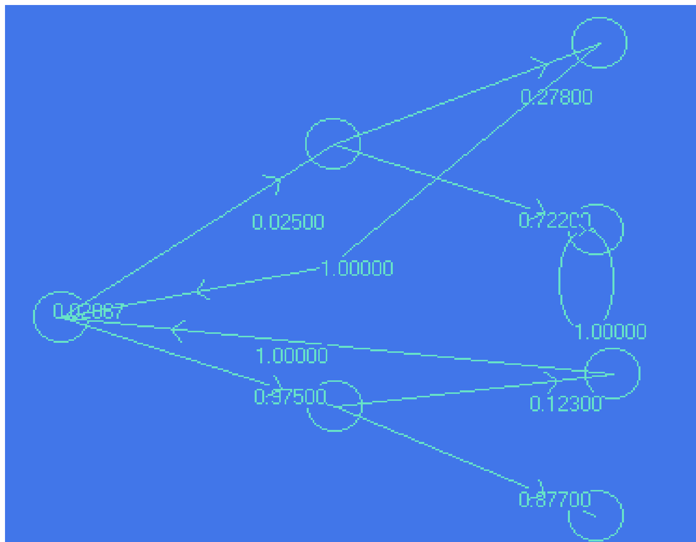


G081.gra

Die Zahlen bedeuten,dass in Ostdeutschland 5,5% aller (Personen-)Unfälle mit Toten und 12,0% mit Verletzten auftreten.

Entsprechend lassen sich durch Verbinden der Blätterknoten O mittels einer Kante der Bewertung 1 mit dem Startknoten S und unter Erzeugen eines Quellenknotens jeweils in den Blätterknoten W die Wahrscheinlichkeiten w= 0,021 bzw. w=0,979 in S erhalten.

Graph zu w=0.021 (0.02087):

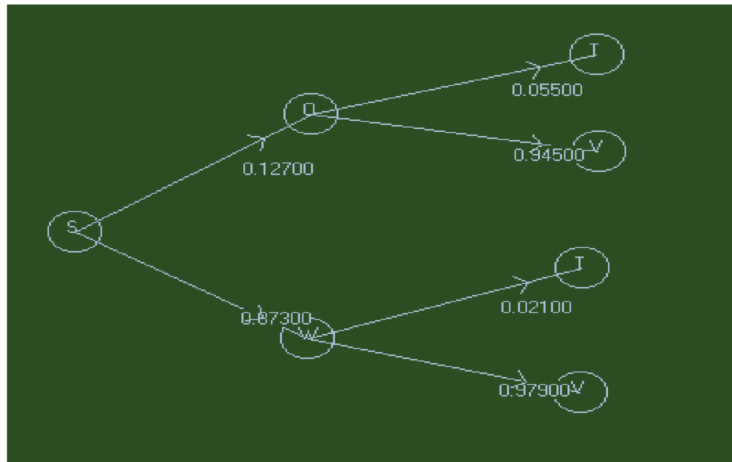


G082.gra

Damit ist die obige Frage beantwortet,da die Wahrscheinlichkeit eines Unfalls in Ostdeutschland mit Toten mit 5,5 % höher ist als die entsprechende Wahrscheinlichkeit in Westdeutschland (2,1%).

Aus den Zahlen läßt sich dann leicht das umgekehrte Baumdiagramm (als auch eine entsprechende Vierfeldertafel) erstellen.

Umgekehrtes Baumdiagramm:



G083.gra

Pfadwahrscheinlichkeiten (zu den Blättern wie oben):

S O Summe: 0.1269 Produkt: 0.1269

S O T Summe: 0.1820 Produkt: 0.0069

S O V Summe: 1.0720 Produkt: 0.1200

S W Summe: 0.8729 Produkt: 0.8729

S W T Summe: 0.8940 Produkt: 0.0183

S W V Summe: 1.8520 Produkt: 0.8546

Allgemein müssen bei dem Verfahren bei beliebigen Pfadbäumen stets die nicht als Quellknoten ausgewählten Blätter(-ereignisse) mittels Kanten der Bewertung 1 mit dem Startknoten verbunden werden,während die ausgewählten Blätterknoten (die das betrachtend Ereignis darstellen) jeweils (nacheinander) als Quellknoten den Fluss 1 erhalten (die anderen sind dabei jeweils Randknoten).

Die Formel von Bayes lautet:

$$w_b(a) = \frac{w(a \cap b)}{w(b)} = \frac{Wahrscheinlichkeit\ für\ a\ und\ b\ (Pfad)}{1 - w(\bar{b})}$$

Das Verfahren beruht auf den oben diskutierten Formel $(1-a_{11})x_1 + (-a_{21})x_2 = c_1$ bzw. $x_a = w_1 x_a + w_2 x_b$,also $x_a = \frac{w_2}{1 - w_1} x_b$,wobei durch durch Erzeugen von Kreisen zum Startknoten mittels Kanten der Bewertung 1 vom Gegenereignis aus der Faktor $1- a_{11}$ bzw. $1-w_1$ hergestellt wird.Der Quellfluss (x_b) muß dabei 1 sein,weil es sich um ein Wahrscheinlichkeitsproblem handelt.Der Fluss durch den Startknoten (z.B. 0.055 bzw. 0.005477) ist dann der Pfadfluss (w_2 z.B. 0.007 bzw. 0.00695) multipliziert mit dem Kehrwert der entsprechenden Wahrscheinlichkeit ($1-w_1$ z.B. 0.127 bzw. 0.12687) der Knoten der ersten Stufe im Umkehrbaum zu dem betrachteten Ereignis,dessen Blattknoten als Quellknoten gewählt wurde (eventuell auch aufgeteilt auf mehrere Pfade).

Durch die Anwendungsaufgaben dieses Kapitels ist gezeigt,dass sich mit dem Programm Knotengraph ein großer Teil der Lösungen von Aufgaben der Wahrscheinlichkeitsrechnung und Statistik didaktisch-methodisch veranschaulichen und erstellen läßt.

C_XIII Kosten-und Transportprobleme,optimales Matching

Ein Ziel des Mathematik- und Informatikunterrichts der Schule sollte die Lösung von Problemen aus der Erfahrungswelt und der realen Umgebung des Schülers sein.Ein wesentlicher Schwerpunkt ist dabei die Erstellung eines abstrakten Modells der Wirklichkeit.Besonders gut eignen sich dazu Aufgaben aus dem Bereich des Operations Research,speziell Kosten- und Transportprobleme.

Die Verkleinerung von Kosten ist eine überall gewünschte Option,deren Sinn sofort einsichtig ist.Die Problemstellung führt sofort auf die Aufstellung einer Kosten-Zielfunktion,die zu minimieren ist.Dabei treten meistens noch Nebenbedingungen auf,wie beispielsweise die Einhaltung bestimmter Transportschranken,die zusätzlich eingehalten werden müssen.

Eine klassische Methode diese Probleme zu lösen bietet der Simplexalgorithmus zur Lösung linearer Optimierungsaufgaben.Er kann als reines Rechenverfahren durch fortlaufende Veränderung der Elemente einer Matrix durchgeführt werden.Eine Veranschaulichung dieses Verfahrens geschieht meistens mit Hilfe der Darstellung linearer Ungleichung als Halbebenen.Meistens wird dieses Thema in einfacher Form schon im Mathematikunterricht der Sekundarstufe I bei der Behandlung linearer Ungleichungen aufgegriffen.

Eine ganz andere Möglichkeit der Veranschaulichung bietet die Graphentheorie mit Hilfe des unmittelbar einsichtigen Algorithmus von Busacker und Gowen,wenn man sich auf spezielle lineare Optimierungsprobleme,die durch Graphen dargestellt werden können,beschränkt.Dazu muß das Problem als Flussproblem formulierbar sein.Es ist dann solange durch den jeweils Kostenminimalsten Pfad ein Fluss maximaler Größe vom Quell- zum Senkenknoten zu schicken,bis bei allen nur möglichen Pfaden von der Quelle zur Senke die oberste Kapazitätsgrenze (Schranke) erreicht ist.

Die Aufgabenstellung kann als Fortsetzung bzw. Spezialisierung der Aufgabe Bestimme den maximalen Fluss gemäß dem Algorithmus von Ford-Fulkerson (vgl. das entsprechende vorige Kapitel C IX) angesehen werden,geht es doch hier darum den maximalen Fluss unter der Auflage der Kostenminimierung zu suchen.Deshalb läßt sich auch die Lösung des maximalen Flusses als Ergebnis des Algorithmus von Busacker und Gowen gewinnen,wenn man einen nicht zu erreichenden zu großen Fluss beim Algorithmus des Menüs Anwendungen/Minimale Kosten des Programms Knotengraph vorgibt.Das Thema dieses Kapitels bietet sich also als Fortsetzung einer Unterrichtsreihe mit den Unterrichtsinhalten Maximaler Fluss und Maximales Matching an (vgl. Kapitel C IX und C X).

Eine Anwendung des Minimalen Kostenalgorithmus auf spezielle Graphen,von deren Quellenknoten Kanten mit speziellen Kapazitätsschranken ausgehen und in deren Senkenknoten Kanten mit speziellen Kapazitätsschranken einlaufen,die jeweils bestimmten Produktions- und Abnahmekapazitäten entsprechen,führt direkt zum (allgemeinen) Transport-bzw. Hitchcockproblem. Spezialisiert man wiederum diese Graphen auf bipartite Graphen,bei denen die genannten Kapazitätschranken auf maximal 1 gesetzt werden,lassen sich sogar optimale Matchingprobleme auf diesen Graphen lösen.Eine weitere Anwendung des Hitchcockproblems auf Graphen,deren Wegstreckenlängen als Kosten angesehen werden,führt dann im nächsten Kapitel zur Lösung des Chinesischen Briefträgerproblems.

So läßt sich um den Algorithmus von Busacker und Gowen eine Unterrichtsreihe mit vielfältigen Themenbereichen und Anwendungsmöglichkeiten,die auseinander entwickelt werden können,arrangieren. Dadurch läßt sich leicht ein problemorientierter Unterricht erreichen.Es kommt noch hinzu,dass sich bei der Bestimmung des minimalen Flusspfades die früher besprochenen Methoden zur Bestimmung des minimalen Pfades zwischen zwei Knoten verwenden lassen,so dass sich unmittelbar die Notwendigkeit objektorientierter Vererbung ergibt und damit ein Zusammenhang zu den Pfadalgorithmen hergestellt wird (vgl. Kapitel C II und C III).

Wiederum ist zwar die Codierung des Algorithmus von Busacker und Gowen nicht schwer zu realisieren,aber leider wiederum relativ umfangreich.

Deshalb gilt hier ebenfalls das schon im letzten Kapitel Gesagte,und es wird wieder empfohlen,ein entsprechendes Programmierprojekt nur in Leistungskursen,bzw. wenn genügend Zeit zur Verfügung steht, durchzuführen.Wenn der Quellcode für das Minimale Kostenproblem allerdings erst einmal erstellt ist,hat man den Vorteil,dass nur noch geringe Änderungen nötig sind,um auch die anderen oben genannten verwandten Probleme damit lösen zu können.Der entsprechende Quellcode wird in diesem Kapitel wieder wie im vorigen Kapitel (nur) als Überblick über die wichtigsten Methoden besprochen.

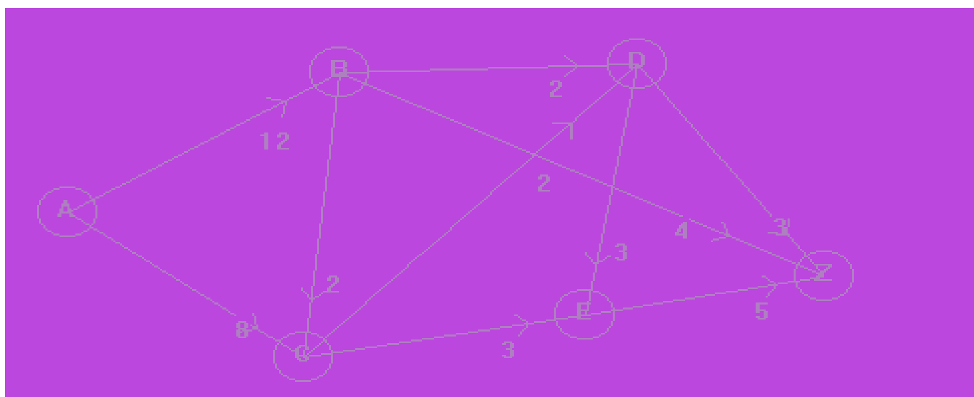
Ansonsten bietet es sich wieder an,das (fertige) Programm Knotengraph als didaktisch-methodisches Werkzeug zu benutzen (bei den Aufgaben dieses Kapitels wird von dieser Option ausgegangen),um im Mathematik- oder Informatikunterricht mittels des Demomodus des Programms Knotengraph und mittels der Menüs Minimale Kosten,Transportproblem,Optimales Matching den Ablauf der entsprechenden Algorithmen zu verdeutlichen und zu erklären (und auch per Hand mit Papier und Bleistift nachzuvollziehen). Beispielsweise könnte sich auf diese Weise,wie schon oben erwähnt,eine Unterichtsreihe zum Algorithmus von Ford-Fulkerson/Maximalflußproblem,das beispielsweise im Informatikunterricht als Programmierungsprojekt durchgeführt wurde,durch die Unterrichtsinhalte dieses Kapitels (ebenso wie auch durch Inhalte aus Kapitel C X/Bestimmung des maximalen Matching in bipartiten Graphen) erweitern und ergänzen lassen.(vgl. dazu auch die Aufgaben CIX.4 und CXIII.4)

(Siehe zu diesem Kapitel auch den entsprechenden Unterrichtsplan im Anhang F IV)

<u>Aufgabe C XIII.1 (Einstiegsproblem)</u>

Gegeben sei der folgende zusammenhängende und gerichtete Graph.Die Knoten des Graphen sollen Städte und die Kanten Transportwege in einem Verkehrsverbund zwischen den Städten per Eisenbahn,Straßenbahn oder Bus bedeuten.Längs jeder Kante fährt jeweils eins der genannten Transportmittel in der durch den Kantenpfeil vorgegebenen Richtung.Vom Knoten (Stadt) A (Quelle) sollen N Personen zum Knoten (Stadt) Z (Senke) transportiert werden.(Der Knoten A hat dabei nur ausgehende Kanten und der Knoten Z nur einlaufende Kanten.)

Die Verkehrsmittel zwischen den Städten haben jeweils eine unterschiedlich große (nichtnegative)) Transportkapazität (in Tausend) pro Tag.Diese Kapazität ist die als Kanteninhalt angezeigte Zahl.



G084.gra

Die Fahrkosten pro Person (in DM oder €) sind auf den verschiedenen Transportwegen verschieden hoch und sind für die verschiedenen Kanten der folgenden Tabelle zu entnehmen:

Kante: A B Kosten: 3
Kante: A C Kosten: 5
Kante: B D Kosten: 5
Kante: B C Kosten: 4
Kante: B Z Kosten: 2
Kante: C D Kosten: 3
Kante: C E Kosten: 5
Kante: D Z Kosten: 3
Kante: D E Kosten: 2
Kante: E Z Kosten: 2

a)Wieviel Personen können maximal pro Tag von Stadt A nach Stadt Z fahren und auf welchen Wegen fahren sie?

(Dabei wird angenommen,dass ein eventuelles Anschlußproblem von einem Verkehrsmittel zum anderen entweder nicht existiert oder aber schon in der Angabe der Transportkapazitäten enthalten ist.)

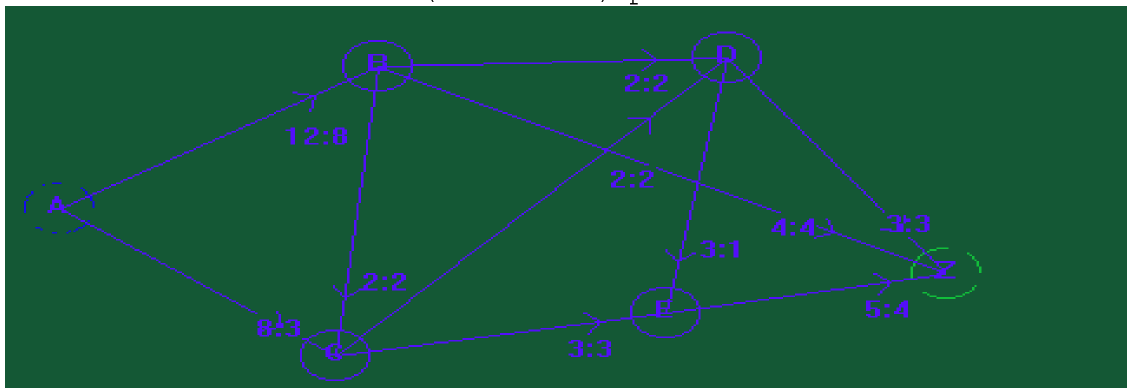
b)Wenn nur 8 Tausend Personen von Stadt A nach Stadt Z fahren wollen,sind Wege gesucht,so dass die Summe der Fahrpreise dieser Personen möglichst gering ist.

Löse die Aufgaben mit den Algorithmen der Menüs Anwendungen/Maximaler Netzfluss und Anwendungen/Minimale Kosten des Programms Knotengraph.

Der Quellen-und Senkenknoten wird automatisch erkannt.Bei der Lösung von Aufgabe b) mit dem Algorithmus Minimale Kosten wird der Benutzer vom Programm aufgefordert durch Mausklick auf die Kanten die jeweiligen Kosten einzugeben.Ergebnisse werden im Graph angezeigt und können nach Ablauf des Algorithmus im Ausgabefenster abgerufen werden.

Lösung:

a)Nach dem Algorithmus von Ford-Fulkerson (vgl. Kapitel C IX) ergibt sich als maximale Personenzahl (in Tausend) pro Verkehrsmittel:



G084.gra

(Angaben in Tausend)

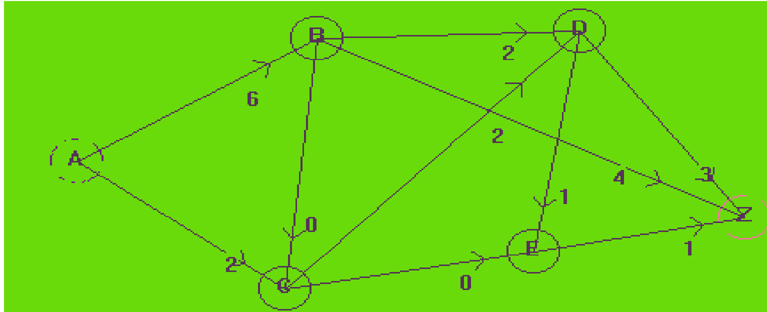
Kanten,Schranken und Fluss:

A->B Schranke: 12 Fluss:8
A->C Schranke: 8 Fluss: 3
B->D Schranke: 2 Fluss: 2
B->C Schranke: 2 Fluss: 2
B->Z Schranke: 4 Fluss: 4
C->D Schranke: 2 Fluss: 2
C->E Schranke: 3 Fluss: 3
D->Z Schranke: 3 Fluss: 3
D->E Schranke: 3 Fluss: 1
E->Z Schranke: 5 Fluss: 4

maximaler Gesamtfluss: 11 (Tausend)

Also können 11 Tausend Personen pro Tag transportiert werden.

b) Kantenfluss, d.h. Personenzahl in Tausend für jedes Verkehrsmittel:



G084.gra

(Angaben in Tausend bis auf Kosten)

Kante: A B Fluss: 6 Flusskosten: 18 Schranke: 12 Kosten: 3
Kante: A C Fluss: 2 Flusskosten: 10 Schranke: 8 Kosten: 5
Kante: B D Fluss: 2 Flusskosten: 10 Schranke: 2 Kosten: 5
Kante: B C Fluss: 0 Flusskosten: 0 Schranke: 2 Kosten: 4
Kante: B Z Fluss: 4 Flusskosten: 8 Schranke: 4 Kosten: 2
Kante: C D Fluss: 1 Flusskosten: 3 Schranke: 2 Kosten: 3
Kante: C E Fluss: 1 Flusskosten: 5 Schranke: 3 Kosten: 5
Kante: D Z Fluss: 3 Flusskosten: 9 Schranke: 3 Kosten: 3
Kante: D E Fluss: 0 Flusskosten: 0 Schranke: 3 Kosten: 2
Kante: E Z Fluss: 1 Flusskosten: 2 Schranke: 5 Kosten: 2

Vorgegebener Fluss durch Quelle oder Senke: 8 (Tausend)

Erreichter Fluss durch Quelle oder Senke: 8 (Tausend)

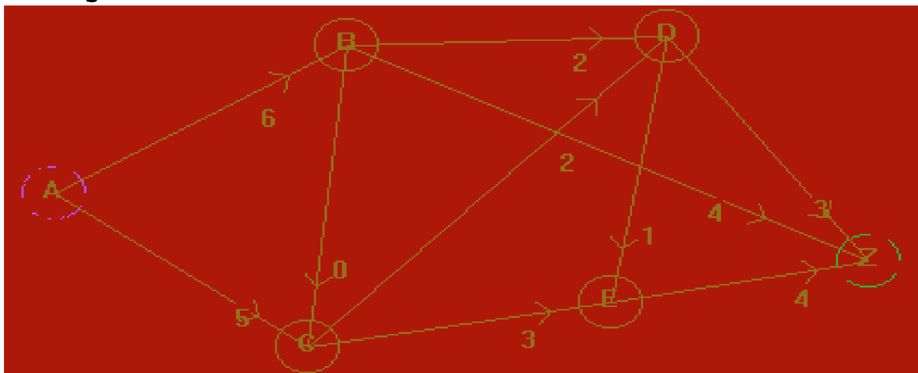
Gesamtkosten: 65 (Tausend) (minimale Kosten für 8 Tausend Personen)

Die Flusskosten bedeuten die auf den einzelnen Kanten (einzelnen Verkehrsmitteln) entstehenden Kosten für (alle damit fahrenden Personen) den gesamten Fluss durch diese Kante.

Zusatz:

Löse die Aufgabe a) auch dadurch, dass für die Personenzahl eine große Zahl N z.B. 1000 mal Tausend Personen vorgegeben wird und statt des Algorithmus Maximaler Netzfluß der Algorithmus Minimale Kosten verwendet wird.

Lösung:



G084.gra
(Angaben in Tausend)

Kante: A B Fluss: 6 Flusskosten: 18 Schranke: 12 Kosten: 3
Kante: A C Fluss: 5 Flusskosten: 25 Schranke: 8 Kosten: 5

Kante: B D Fluss: 2 Flusskosten: 10 Schranke: 2 Kosten: 5
Kante: B C Fluss: 0 Flusskosten: 0 Schranke: 2 Kosten: 4
Kante: B Z Fluss: 4 Flusskosten: 8 Schranke: 4 Kosten: 2
Kante: C D Fluss: 2 Flusskosten: 6 Schranke: 2 Kosten: 3
Kante: C E Fluss: 3 Flusskosten: 15 Schranke: 3 Kosten: 5
Kante: D Z Fluss: 3 Flusskosten: 9 Schranke: 3 Kosten: 3
Kante: D E Fluss: 1 Flusskosten: 2 Schranke: 3 Kosten: 2
Kante: E Z Fluss: 4 Flusskosten: 8 Schranke: 5 Kosten: 2

Vorgegebener Fluss durch Quelle oder Senke: 1000 (Tausend)

Erreichter Fluss durch Quelle oder Senke: 11 (Tausend)

Gesamtkosten: 101 (Tausend)

Es ergibt sich wieder nur die maximale Zahl von 11 Tausend Personen.Die Lösung ist Kostenminimal bezüglich der Gesamtkosten.Eine Personenzahl von 1000 mal Tausend Personen kann nicht pro Tag transportiert werden.Also läßt sich auch auf diese Weise das Maximalflußproblem lösen,ohne den Algorithmus von Ford-Fulkerson zu benutzen.

Aufgabe C XIII.2:

Löse Teil b) der vorigen Aufgabe zunächst durch Probieren per Hand und dann mit Hilfe des Programms Knotengraph mit dem Menü Anwendungen/Minimale Kosten im Demomodus für den folgenden Graphen G085.gra.
(Beispiel aus Lit 10, S.356)

Zusatzaufgaben:

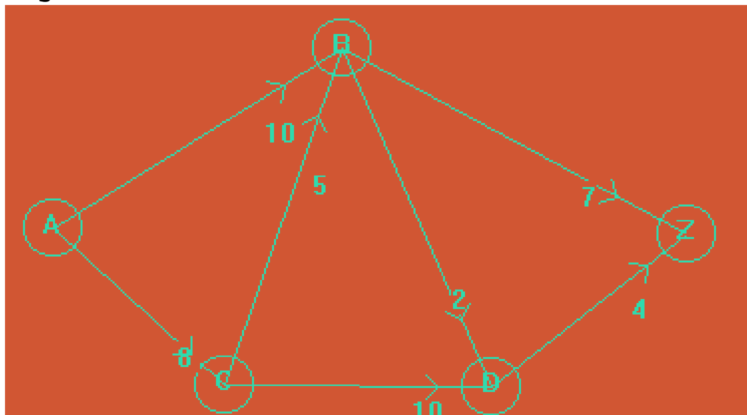
I)Wie viele Personen können hier maximal von A nach Z fahren?

II)Löse die Aufgabe auch für **maximale** Fahrkosten,indem die entsprechende Option im Ablauf des Algorithmus gewählt wird.

Kosten:

Kante: A B Kosten: 4
Kante: A C Kosten: 1
Kante: B Z Kosten: 1
Kante: B D Kosten: 6
Kante: C D Kosten: 3
Kante: C B Kosten: 2
Kante: D Z Kosten: 2

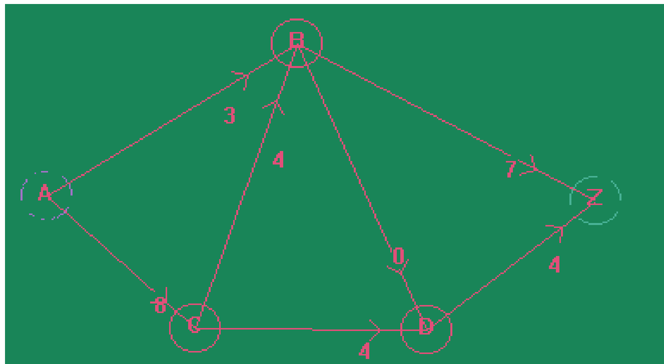
Angaben in Tausend:



G085.gra

Lösung:

Angaben in Tausend:



G085.gra

(Angaben in Tausend)

Kante: A B Fluss: 3 Flusskosten: 12 Schranke: 10 Kosten: 4
Kante: A C Fluss: 8 Flusskosten: 8 Schranke: 8 Kosten: 1
Kante: B Z Fluss: 7 Flusskosten: 7 Schranke: 7 Kosten: 1
Kante: B D Fluss: 0 Flusskosten: 0 Schranke: 2 Kosten: 6
Kante: C D Fluss: 4 Flusskosten: 12 Schranke: 10 Kosten: 3
Kante: C B Fluss: 4 Flusskosten: 8 Schranke: 5 Kosten: 2
Kante: D Z Fluss: 4 Flusskosten: 8 Schranke: 4 Kosten: 2

Vorgegebener Fluss durch Quelle oder Senke: 11 (Tausend)

Erreichter Fluss durch Quelle oder Senke: 11 (Tausend)

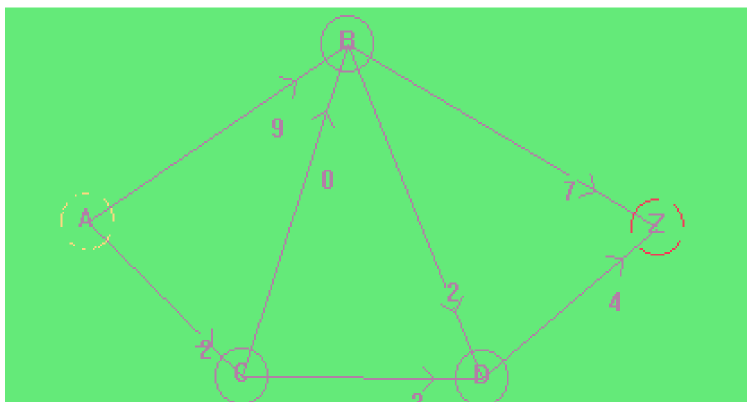
Gesamtkosten: 55 (Tausend) (minimale Fahrkosten)

Lösung der Zusatzaufgaben:

I)Die Personenzahl 11 Tausend ist auch maximal.

II)

Angaben in Tausend:



G085.gra

Angaben in Tausend:

Kante: A B Fluss: 9 Flusskosten: 36 Schranke: 10 Kosten: 4
Kante: A C Fluss: 2 Flusskosten: 2 Schranke: 8 Kosten: 1
Kante: B Z Fluss: 7 Flusskosten: 7 Schranke: 7 Kosten: 1
Kante: B D Fluss: 2 Flusskosten: 12 Schranke: 2 Kosten: 6

Kante: C D Fluss: 2 Flusskosten: 6 Schranke: 10 Kosten: 3
Kante: C B Fluss: 0 Flusskosten: 0 Schranke: 5 Kosten: 2
Kante: D Z Fluss: 4 Flusskosten: 8 Schranke: 4 Kosten: 2

Vorgegebener Fluss durch Quelle oder Senke: 11 (Tausend)

Erreichter Fluss durch Quelle oder Senke: 11 (Tausend)

Gesamtkosten: 71 (Tausend) (maximale Fahrkosten)

Definition C XIII.1:

Gegeben sei ein Flussgraph gemäß Definition C IX.1.Jeder Kante des Graphen sei zusätzlich eine nichtnegative Zahl als Kosten pro Flusseinheit durch diese Kante zugeordnet.Vorgegeben sei außerdem ein Vorgabefluss als nichtnegative Zahl,der kleiner oder gleich dem maximal möglichen Fluss vom Quellen- zum Senkenknoten ist.Unter einem Kostenminimalen (Kostenmaximalen) Fluss zum Vorgabefluß versteht man dann einen bezüglich der Schranken zulässigen Fluss,so dass jeder Kante eine nichtnegative Zahl als Fluss zugeordnet wird,mit der Eigenschaft dass die Summe aller Produkte Fluss der Kante multipliziert mit den Kosten pro Fluss der Kante unter allen Möglichkeiten eines bezüglich der Schranken zulässigen Flusses auf dem Graphen minimal (maximal) ist und dass der Fluss,der vom Quellenknoten ausgeht bzw. in den Senkenknoten mündet,gleich dem Vorgabefluss ist.

Es sei k_{ij} Kante vom i.ten zum j.ten Knoten des Graphen,c_{ij} sei die der Kante zugeordnete Schranke mit $c_{ij} \geq 0$.$K_{ij} \geq 0$ seien die der Kante zugeordneten Kosten.Mit f sei der Fluss durch die Kante als Funktion der Kante bezeichnet

Dann gilt:

I) $0 \leq f(k_{ij}) \leq c_{ij}$ für alle Kanten k_{ij} des Graphen

II) $$\sum_{i=1}^{n} f(k_{ij}) - \sum_{k=1}^{n} f(k_{jk}) = \begin{cases} -v \text{ für } j = \text{Quellknoten} \\ v \text{ für } j = \text{Senkenknoten} \\ 0 \quad \text{sonst} \end{cases}$$,wobei v der Vorgabefluß ist und der Fluss für alle Knotenpaare,zwischen denen sich keine Kanten befinden,Null ist.

k_{ij} sind die in den j.ten Knoten einlaufenden Kanten,k_{jk} sind die aus dem j.ten Knoten auslaufenden Kanten.Die Zahl n ist die Anzahl der Knoten des Graphen.

Die letzte Bedingung kennzeichnet die Eigenschaft eines Flusses,dass soviel Fluss in einen Knoten hineinfließt,wie aus dem Knoten wieder herausfließt (außer beim Quellen-und Senkenknoten).

Aufgabe:

III)Minimiere (Maximiere) die Funktion G= $$\sum_{1 \leq i \leq n, 1 \leq j \leq n, i \neq j} f(k_{ij}) K_{ij}$$

Wie kann nun ein Kostenminimaler Fluss zu einem vorgegebenen Graphen und zu einem Vorgabefluss ermittelt werden?Ein möglicher Algorithmus ist der Maximalfluß-Algorithmus von Busacker und Gowen:

Der Algorithmus von Busacker und Gowen

Eine problemorientierte Erarbeitung des Verfahrens findet sich im entsprechenden Unterrichtsplan des Anhangs FIV.

Verbale Beschreibung:

1)Setze für alle Kanten k_{ij} des Graphen $f(k_{ij})=0$ (Anfangsfluss).

2)Bestimme für jede Kante die modifizierten Kosten K'_{ij},die sich folgendermaßen aus den Kosten K_{ij} für jede Kante k_{ij} berechnen:

a) $K'_{ij} = K_{ij}$ für $f(k_{ij})<c_{ij}$

b) $K'_{ij} = \infty$ für $f(k_{ij})=c_{ij}$

c) $K'_{ji} = -K_{ij}$ für $f(k_{ij})<0$

d) $K'_{ji} = \infty$ für $f(k_{ij})=0$

K'_{ji} sind die Kosten der Kante k_{ji}.

Dabei ist k_{ji} die zu k_{ij} entgegengesetzte Kante,durch die ein Fluss entgegengesetzt der Pfeilrichtung von k_{ij} fließt.Der Sinn eines solchen Flusses wurde schon im Kapitel C IX bei der Lösung des Maximalflußproblems erläutert (Algorithmus von Ford-Fulkerson).Es ist der Fluss,der den momentanen Fluss bei Rückwärtskanten verringert.Dies ist nur möglich,wenn der momentane Fluss größer als Null ist.Eine Erhöhung des Vorwärtsflusses geht nur,wenn der momentane Fluss kleiner als die Kantenschranke ist.Ansonsten wird die Kante für Flüsse durch das Setzen des Wertes ∞ für die Kosten gesperrt.

3)Suche bezüglich der modifizierten Kosten als Kantenbewertung einen minimalen (= kürzesten) Pfad,der aus Vorwärts-und Rückwärtskanten bestehen kann,vom Quellen- zum Senkenknoten.
Falls kein solcher Pfad existiert,ist der Algorithmus beendet,weil kein Fluss zur Größe des vorgegebenen Flusses bezüglich minimaler Kosten gefunden werden kann.
Wenn ein solcher Pfad existiert,setze den Algorithmus bei Schritt 4) fort.

4)Erhöhe den Fluss auf diesem Pfad für alle Vorwärtskanten bzw. vermindere ihn bei Rückwärtskanten um denselben Betrag,der so gewählt wird,dass noch ein zulässiger Fluss längs dieses Pfades entsteht (Beachtung der Schranken und der Nichtnegativität des Flusses) und außerdem bei mindestens einer Vorwärtskante der Fluss gleich der Schranke bzw. bei mindestens einer Rückwärtskante der Fluss gleich Null wird.

5)Ziehe anschließend den Betrag diese Flusses vom vorgegeben Fluss v ab,so dass der neue vorgegebene Fluss nun um diesen Betrag d vermindert ist.

6)Falls der vorgegeben Fluss kleiner oder gleich als der Betrag d ist,erhöhe bzw vermindere den Fluss durch die Kanten auf dem Pfad nur um den Betrag d.Setze dann den neuen vorgegebenen Fluss v auf Null.In diesem Fall ist der Algorithmus beendet,und ein Fluss durch die Kanten zum vorgegebenen Fluss v gefunden,der Kostenminimal ist.

Ansonsten setze den Algorithmus bei Schritt 2) fort.

Der Algorithmus führt also das Problem der Bestimmung minimaler Kosten auf das wiederholte Suchen (und Verändern der Flusswerte) von minimalen Pfaden zwischen zwei Knoten zurück.Dieses Problem wurde schon im Kapitel C III mit dem Algorithmus der Methode BestimeMinimalerPfad erörtert,wobei hier allerdings das Verfahren von Ford (siehe Anhang) statt des von Dijkstra wegen der negativen Kantenbewertung zu wählen ist.

Satz XIII.1:

Der Algorithmus von Busacker und Gowen liefert stets einen Kostenminimalen Fluss auf einem gemäß obiger Definition C IX.1 vorgegebenen Graphen für ganzzahlige Flüsse.

Beweisskizze:

Falls in dem vorgegeben Graphen ein Kreis aus gemäß den modifizierten Kosten zulässigen Rückwärtskanten existiert,deren Fluss jeweils bei allen Kanten um eine Differenz d gesenkt werden kann,heißt ein solcher Kreis ein Kreis negativer Länge.Unter Länge sind hier die Summe der Produkte Modi-

fizierte Kosten der Kanten (alle endlich und negativ) multipliziert mit dem Fluss durch die Kanten zu verstehen.

Es gilt dann der Hilfssatz:

Ein Fluss der vorgegebenen Größe v ist genau dann von minimalen Kosten,wenn in dem Graph bezüglich der modifizierten Kosten kein Kreis negativer Länge (Kosten) existiert.

Zu einem Beweis dieses Hilfssatzes siehe z.B. Busacker,Saaty,Endliche Graphen und Netzwerke,Ouldenbourgh-Verlag,München und Wien,S.353,Theorem 7-8)

Mit diesem Hilfssatz,bei dem der Nachweis der notwendigen Bedingung trivial ist,da man bei Vorliegen eines Kreises negativer Länge den Fluss dort um den Betrag d vermindern könnte und somit die Kosten um den entsprechenden Betrag senken könnte (die hinreichende Bedingung ist allerdings nachzuweisen),läßt sich der Beweis folgendermaßen führen:

Statt den Fluss wie oben beschrieben immer zugleich um den Betrag d zu erhöhen,läßt er sich auch schrittweise jeweils um den Betrag 1 längs eines vorhandenen Minimalen-Kosten-Pfades erhöhen.

Bis zur Erhöhung des Flusses auf den Fluss 1,2.... i mit i<v sei noch kein Kreis negativer Länge im Graphen vorhanden.

Bei der Erhöhung auf r=i+1≤w trete erstmalig ein Kreis K negativer Länge im Graph auf.

Dann muß es eine Kante im negativen Kreis K geben,die zum minimalen Pfad P bezüglich der Erhöhung des Flusses um 1 von i auf i+1 gehört.

Diese Kante wurde in Vorwärtsrichtung durchlaufen,damit sich der Fluss um 1 erhöht.Nur so kann dadurch ein neuer Kreis negativer Länge entstehen.

Diese Kante werde nun aus dem Pfad P herausgenommen und durch die übrigen (Rückwärts-)Kanten des Kreises K ersetzt.Dies ergibt einen erneuten Pfad vom Quellen-zum Senkenknoten P´.
Die Kosten auf diesem Pfad P´ sind kleiner als die Kosten auf dem Pfad P,weil der Fluss in den Rückwärtskanten noch mindestens um eine Flusseinheit gesenkt werden kann,so dass ein PFad mit kleineren modifizierten Kosten entsteht.Das ist ein Widerspruch dazu,dass der Pfad P ein Pfad mit den kleinsten modifizierten Kosten von der Quelle zur Senke war.

Also entstehen bei dem Algorithmus von Busacker und Gowen nur Graphen ohne negative Kreise.Nach dem Hilfssatz (hinreichende Richtung) ist der Fluss demnach Kostenminimal.

Die Datenstruktur:

Die Datenstruktur gilt gleichzeitig für die Algorithmen der Anwendungen Minimale Kosten,Transportprobleme,Optimales Matching und Chinesischer Briefträger.

```
TMinimaleKostenkante = class(TInhaltskante)
  private
    Fluss_:Extended;
    Kosten_:Extended;
    ModKosten_:string;
    KostenWert_:Extended;
    Schranke_:Extended;
    Richtung_:Boolean;
    PartnerKante_:Integer;
    Ergebnis_:string;
    procedure SetzeFluss(Fl:Extended);
    function WelcherFluss:Extended;
    procedure SetzeKosten(Ko:Extended);
    function WelcheKosten:Extended;
    procedure SetzeModKosten(Mko:string);
    function WelcheModKosten:string;
    procedure SetzeKostenWert(Ako:Extended);
```

```
    function WelcherKostenWert:Extended;
    procedure SetzeSchranke(Schr:Extended);
    function WelcheSchranke:Extended;
    procedure SetzeRichtung(Ri:Boolean);
    function WelcheRichtung:Boolean;
    procedure SetzePartnerKante(Ka:TMinimaleKostenkante);
    function  WelchePartnerkante:TMinimalekostenkante;
    procedure SetzeKantenindex(I:Integer);
    function WelcherKantenindex:Integer;
    procedure SetzeErgebnis(Erg:string);
    function WelchesErgebnis:string;
  public
    constructor Create;
    function Wertlisteschreiben:TStringlist;override;
    procedure Wertlistelesen;override;
    property Fluss:Extended read WelcherFluss write SetzeFluss;
    property Kosten:Extended read WelcheKosten write SetzeKosten;
    property ModKosten:string read WelcheModKosten write SetzeModKosten;
    property KostenWert:Extended read WelcherKostenWert Write setzeKostenWert;
    property Schranke:Extended read WelcheSchranke write setzeSchranke;
    property Richtung:Boolean read WelcheRichtung write SetzeRichtung;
    property Partnerkante:TMinimaleKostenkante read WelchePartnerkante
    write SetzePartnerkante;
    property Kantenindex:Integer read WelcherKantenindex write SetzeKantenindex;
    property Ergebnis:string read WelchesErgebnis write setzeErgebnis;
  end;
```

TMinimaleKostenKante leitet sich von TInhaltskante durch Vererbung ab und enthält folgende neue Datenfelder:

Fluss_:Enthält den Fluss durch die Kante.
Kosten_:Enthält die Kosten der Kante pro Flusseinheit.
ModKosten_:Nimmt den Wert der modifizierten Kosten auf.
KostenWert_:Nimmt das Produkt aus Kosten pro Flusseinheit mal Kosten auf.
Schranke_:Nimmt die Kantenschranke auf,die zunächst als Inhalt_ der Kante gespeichert ist.
Richtung_:Gibt an,ob es sich um eine Vorwärts-oder Rückwärtskante handelt.
PartnerKante_:Speichert einen Verweis auf die zugeordnete Vorwärts-bzw. Rückwärtskante (als Index in der Kantenliste).
Ergebnis_: Nimmt einen Ergebnisstring aus Kantenschranke und Kantenfluss zur Anzeige auf der Zeichenoberfläche auf.

Zusätzlich zu den im Graph vorhandenen Vorwärtskanten wird zu jeder Vorwärtskante eine (gerichtete) Rückwärtskante neu erzeugt und in den Graph eingefügt,um die bezüglich der modifizierten Kosten minimalen Pfade zwischen Quellen-und Senkenknoten mittels des Algorithmus von Ford suchen zu lassen.

Auf die Felder wird mittels geeigneter Propertys zugegriffen.

```
TMinimaleKostengraph = class(TInhaltsgraph)
    private
    Gesamtfluss_:Extended;
    VorgegebenerFluss_:Extended;
    Ganzzahlig_:Boolean;
    procedure SetzeGesamtfluss(Gfl:Extended);
    function WelcherGesamtfluss:Extended;
    procedure SetzeVorgegebenenFluss(Vfl:Extended);
    function WelcherVorgegebeneFluss:Extended;
    procedure SetzeGanzzahlig(Gz:Boolean);
    function WelcherWertganzzahlig:Boolean;
    public
    constructor Create;
    function Wertlisteschreiben:TStringlist;override;
    procedure Wertlistelesen;override;
    property Gesamtfluss:Extended read WelcherGesamtfluss write SetzeGesamtfluss;
    property VorgegebenerFluss:Extended read WelcherVorgegebeneFluss
    write SetzeVorgegebenenFluss;
    property Ganzzahlig:Boolean read WelcherWertGanzzahlig write SetzeGanzzahlig;
    procedure SpeichereSchranken;
    procedure SpeichereSchrankenalsKosten;
    procedure SpeichereKosten;
    procedure LadeKosten;
    procedure InitialisiereKanten;
    procedure SetzeVorwaertsKantenRichtungaufvor;
```

```
    procedure ErzeugeRueckkanten;
    procedure EleminiereRueckkanten;
    procedure ErzeugeModKosten;
    function SucheminimalenWegundbestimmeneuenFluss(Quelle,Senke:TInhaltsknoten;
      Flaeche:TCanvas):Boolean;
    procedure EingabeVorgegebenerFluss;
    procedure BestimmeFlussKostenfuerKanten(var Sliste:Tstringlist);
    function  GesamtKosten:Extended;
    procedure BildeinverseKosten;
    function BestimmeQuelleundSenke(var Quelle,Senke:TInhaltsknoten;Flaeche:TCanvas;
    Anzeigen:Boolean):Boolean;
    function ErzeugeQuellenundSenkenknotensowieKanten(var Quelle,Senke:TInhaltsknoten;
      var Fluss:Extended;Art:char):Boolean;
    procedure BestimmeQuelleSenkesowievorgegebenenFluss(var Quelle,Senke:TInhaltsknoten;
      Ausgabe1:TLabel;Flaeche:TCanvas;Art:char);
    procedure SetzeSchrankenMax;
    procedure LoescheQuellenundSenkenknoten(Quelle,Senke:TInhaltsknoten);
    procedure ErzeugeunproduktiveKanten;
    procedure LoescheNichtMatchKanten;
    procedure MinimaleKosten(Flaeche:TCanvas;var G:TInhaltsgraph;
      var Oberflaeche:TForm;Ausgabe1:TLabel;Maximal:Boolean;
      var Quelle,Senke:TInhaltsknoten;
      Art:char;Var Sliste:TStringlist);
  end;
```

TMinimaleKostengraph leitet sich von TInhaltsgraph durch Vererbung ab und beinhaltet folgende neue Datenfelder:

Gesamtfluss_:Speichert den momentanen Gesamtfluss durch Quellen-und Senkenknoten.
VorgegebenerFluss_:Speichert den zu erreichenden Vorgabefluss.
Ganzzahlig_:Speichert,ob nur ganzzahlige Flüsse erzeugt werden sollen.

Auch auf diese Felder wird mittels geeigneter Propertys zugegriffen.

Der Quellcode in Auswahl:

```
procedure TMinimaleKostengraph.ErzeugeRueckkanten;
var Index,Weite:Integer;
    HilfsKantenliste:TKantenliste;
    Ka,Kr:TMinimaleKostenKante;

begin
  Hilfskantenliste:=TKantenliste.Create;
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
      Hilfskantenliste.AmEndeanfuegen(Kantenliste.Kante(Index));
  if not Hilfskantenliste.Leer then
    for Index:=0 to Hilfskantenliste.Anzahl-1 do
    begin
      Ka:=TMinimalekostenkante(Hilfskantenliste.Kante(Index));
      if Ka.Richtung=false then
      begin
        Weite:=Ka.Weite;
        Kr:=TMinimaleKostenkante.Create;
        Kr.Position:=self.Kantenwertposition;
        Kr.Wert:='R';
        Kr.Weite:=-Weite;
        Kr.gerichtet:=true;
        Kr.Richtung:=true;
        Kr.Partnerkante:=Ka;
        Kr.Kosten:=Ka.Kosten;
        Kr.Schranke:=Ka.Schranke;
        Kr.Fluss:=0;
        FuegeKanteein(TInhaltsknoten(Ka.Endknoten),TInhaltsknoten(Ka.Anfangsknoten),true,Kr);
      end;
    end;
  Hilfskantenliste.Free;
  Hilfskantenliste:=nil;
end;
```

Die Methode ErzeugeRueckkanten erzeugt zu jeder Kante des Graphen eine zugehörige Rückwärtskante mit der Bezeichnung (Wert) R.Sie besitzt die gleichen Kosten und Schranke wie die ursprüngliche Kante.

```
function MinimaleKostenKantenWert(x:TObject):Extended;
var Ka:TMinimaleKostenKante;
```

```
begin
  Ka:=TMinimaleKostenKante(x);
  if Ka.Richtung=false then
  begin
    if not TMinimaleKostengraph(Ka.Anfangsknoten.Graph).ganzzahlig then
    begin
      if Ka.Fluss=Ka.Schranke
      then
        MinimaleKostenKantenwert:=1E32
      else
        MinimaleKostenKantenWert:=Ka.Kosten;
    end;
    if TMinimaleKostengraph(Ka.Anfangsknoten.Graph).Ganzzahlig then
    begin
      if Ka.Schranke<1E4 then
      begin
        if trunc(Ka.Fluss)=trunc(Ka.Schranke)
        then
          MinimaleKostenKantenWert:=1E32
        else
          MinimaleKostenKantenWert:=Ka.Kosten;
      end
      else
       MinimaleKostenkantenwert:=Ka.Kosten;
    end;
  end;

  if Ka.Richtung=true then
  begin
    if Ka.Partnerkante=nil then
    begin
      Showmessage('Fehler:Partnerkante existiert nicht!');
      halt;
    end;
    if Ka.Partnerkante.Fluss=0
    then
      MinimaleKostenKantenWert:=1E32
    else
      MinimalekostenKantenwert:=-(Ka.Partnerkante).Kosten;
  end;
end;
```

Die Function MinimaleKostenKantenwert bestimmt die modifizierten Kosten für jede Kante abhängig davon,ob es sich um eine Vorwärts-oder Rückwärtskante handelt und ob der Fluss ganzzahlig oder nicht ganzzahlig sein soll.Die durch diese Function gegebene Bewertung wird dann verwendet,um in der Methode SucheminimalenPfadundbestimmeneuenFluss nach minimalen Pfaden zwischen Quellen-und Senkenknoten suchen zu lassen und den Fluss zu verändern.

```
function TMinimaleKostenGraph.SucheminimalenWegundbestimmeneuenFluss
(Quelle,Senke:TInhaltsknoten;Flaeche:TCanvas):Boolean;
label Endproc;
var MinimalerPfadgraph:TMinimaleKostengraph;
    MinFlussschranke,Differenz:Extended;
    Ka:TminimaleKostenkante;
    Index:Integer;
    Weggefunden:Boolean;

  procedure BestimmeMinimumFlussSchranke(Ka:TMinimaleKostenKante);
  begin
    Weggefunden:=true;
    if Ka.Richtung=false then
      if Ka.Schranke-Ka.Fluss<MinflussSchranke then
        MinFlussSchranke:=Ka.Schranke-Ka.Fluss;
    if Ka.Richtung=true  then if Ka.Partnerkante.Fluss<MinflussSchranke then
      MinFlussschranke:=Ka.Partnerkante.Fluss;
  end;

  procedure ErhoeheFlussumDifferenz(Ka:TMinimalekostenkante);
  begin
    if Ka.Richtung=false then Ka.Fluss:=Ka.Fluss+Differenz;
    if Ka.Richtung=true then Ka.Partnerkante.Fluss:=Ka.Partnerkante.Fluss-Differenz;
  end;

begin
  MinimalerPfadgraph:=
  TMinimaleKostengraph(BestimmeMinimalenPfad(Quelle,Senke,MinimaleKostenKantenWert));    1)
  if (MinimalerPfadgraph.Kantenliste.Leer)                                               2)
  or (Minimalerpfadgraph.Kantenliste.WertsummederElemente(MinimaleKostenKantenWert)>1E31) then
  begin
```

```
    ShowMessage('Vorgegebener Fluss '+RundeZahltoString(VorgegebenerFluss,Kantengenauigkeit)
    +' ist nicht zu erreichen!');
    Weggefunden:=false;
    goto Endproc;
  end;
  Minflussschranke:=1E32;
  if not MinimalerPfadgraph.Kantenliste.Leer then
    for Index:=0 to MinimalerPfadgraph.Kantenliste.Anzahl-1 do
    begin
      Ka:=TMinimaleKostenKante(Minimalerpfadgraph.Kantenliste.Kante(Index));
      BestimmeMinimumFlussschranke(Ka);                                          3)
    end;
  if Gesamtfluss+MinflussSchranke>VorgegebenerFluss then                         4)
    MinFlussSchranke:=VorgegebenerFluss-GesamtFluss;
  Differenz:=MinFlussSchranke;
  if Ganzzahlig then Differenz:=Trunc(Differenz);
  if not MinimalerPfadgraph.Kantenliste.Leer then
    for Index:=0 to MinimalerPfadgraph.Kantenliste.Anzahl-1 do
    begin
      Ka:=TMinimaleKostenKante(Minimalerpfadgraph.Kantenliste.Kante(Index));
      ErhoeheFlussumDifferenz(Ka);                                               5)
    end;
  GesamtFluss:=GesamtFluss+Differenz;                                            6)
  Endproc:
  SucheMinimalenWegundbestimmeneuenFluss:=Weggefunden;
end;
```

Bei 1) wird ein bezüglich der modifizierten Kosten minimaler Pfad als Graph zwischen Quellen-und Senkenknoten gesucht und,falls existent,auf der Variablen MinimalerPfadgraph gespeichert.Wenn bei 2) kein solcher Pfad existiert oder aber der Wert der modifizierten Kosten gleich unendlich ist (1E32),kann der vorgegebene Fluss nicht erreicht werden und der Algorithmus wird beendet.Ansonsten wird bei 3) für den Pfad der Betrag d gesucht,um den der Fluss maximal längs des Pfades bei allen Vorwärtskanten erhöht (unter Beachtung der Schranken) und bei Rückwärtskanten verringert (unter Beachtung nicht negativer Fluss) werden kann.Bei 4) wird geprüft,ob bei Addition des Betrages d zum bisherigen Gesamtfluss schon der vorgegebene Fluss erreicht wird. (Wenn ja,wird d als Differenz zwischen Vorgabefluss und Gesamtfluss festgesetzt.Dann ist der Algorithmus beendet.)Bei 5) wird schließlich der Fluss der Kanten auf dem Pfad vergrößert bzw. verkleinert und bei 6) wird der momentane Gesamtfluss erneut angepasst.

```
procedure TMinimalekostengraph.MinimaleKosten(Flaeche:TCanvas;varG:TInhaltsgraph;
Var Oberflaeche:TForm;Ausgabe1:TLabel;Maximal:Boolean;var Quelle,Senke:TInhaltsknoten;
Art:char;var Sliste:Tstringlist);
label Endproc;
var Gesamtkosten:Extended;
begin
  InitialisiereKanten;                                                           7)
  Kantenwertposition:=1;
  SetzeVorwaertsKantenrichtungaufVor;                                            8)
  Kantenwertposition:=6;
  Kantenwertposition:=0;
  SpeichereSchranken;
  Kantenwertposition:=5;
  BestimmeQuelleSenkesowievorgegebenenFluss(Quelle,Senke,Ausgabe1,Flaeche,Art);  9)
  LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  Ausgabe1.Caption:='Berechnung läuft...';
  Kantenwertposition:=5;
  if Maximal then                                                               16)
  begin
    SpeichereKosten;
    BildeinverseKosten;
  end;
  ErzeugeRueckkanten;                                                           10)
  repeat                                                                        11)
    ErzeugeModKosten;                                                           12)
    Kantenwertposition:=3;
    if not SucheminimalenWegundbestimmeneuenFluss(Quelle,Senke,Flaeche)         13)
    then
    begin
      ShowMessage('Erreichbarer maximaler Fluss:'+
      RundeZahltostring(self.Gesamtfluss,Kantengenauigkeit));
      if Art='b' then ShowMessage('Das Briefträgerproblem ist auf dem gerichteten'
      +chr(13)+'Graphen nicht lösbar!');
      goto Endproc;
    end;
```

```
  until self.VorgegebenerFluss<=self.Gesamtfluss;
  Endproc:
  EleminiereRueckkanten;                                                    14)
  if Maximal then LadeKosten;
  If Art in ['t','b','o'] then
    LoescheQuellenundSenkenknoten(Quelle,Senke);
  BestimmeFlussKostenfuerKanten(Sliste);                                    15)
  Sliste.Add('Vorgegebener Fluss durch Quelle oder Senke: '+
  RundeZahltoString(self.VorgegebenerFluss,Kantengenauigkeit));
  Sliste.Add('Erreichter Fluss durch Quelle oder Senke: '+
  RundeZahltoString(self.Gesamtfluss,Kantengenauigkeit));
  Sliste.Add('Gesamtkosten: '+RundeZahltoString(self.Gesamtkosten,
  Kantengenauigkeit));
  Gesamtkosten:=self.Gesamtkosten;
  if Art<>'b'
  then
    ShowMessage('Gesamtkosten: '+RundeZahltoString(Gesamtkosten,Kantengenauigkeit))
  else
    ShowMessage('Unproduktive Weglängen: '+RundeZahltoString(Gesamtkosten,
    Kantengenauigkeit));
    ShowMessage('Vorgegebener Fluss:'+
  RundeZahltostring(self.VorgegebenerFluss,Kantengenauigkeit));
    ShowMessage('Erreichter Fluss durch Quelle oder Senke: '+
  RundeZahltostring(self.Gesamtfluss,Kantengenauigkeit));
end;
```

Bei 7) werden alle Kantenfelder auf ihre Anfangswerte (Null) gesetzt,und bei 8) werden die momentan (im ursprünglichen Graphen)vorhandenen Kanten als Vorwärtskanten markiert.Bei 9) wird der Quellen-und Senkenknoten bestimmt,sowie der vorgegebene Fluss vom Benutzer in einem Eingabefenster von der Tastatur eingegeben.Bei 10) wird zu jeder Vorwärtskante eine entgegengesetzte Kante mit gleicher Schranke und Kosten erzeugt und in den Graph mit entsprechender Markierung als Rückwärtskante eingefügt.
Bei 11) beginnt eine Schleife,in der zunächst (12) die Modifizierten Kosten zur Ausgabe auf der Zeichenoberfläche gebildet werden und dann solange die Methode SucheminimalenWegundbestimmeneuenFluss (13) aufgerufen wird,bis der momentan erreichte Gesamtfluss gleich oder größer als der Vorgabefluss ist.Danach werden die Rückwärtskanten wieder aus dem Graphen gelöscht (14), und die Gesamtkosten als minimale Kosten berechnet (15).

Soll der Fluss statt zu den minimalen Kosten für die maximalen Kosten gesucht werden,kann dies durch Erzeugung der invertierten Kosten (16) erreicht werden,zu denen jeweils wieder der Fluss für ein MinimalesKosten-Problem bestimmt wird.

```
procedure  TMinimaleKostengraph.BildeinverseKosten;
var Max:Extended;
    Zaehl:Integer;

  function MaxKosten:Extended;
  var Index:Integer;
      MKosten:Extended;

    procedure BestimmeMaxKosten(Ka:TMinimaleKostenkante);
    begin
      if Ka.Kosten>MKosten then
        MKosten:=Ka.Kosten;
    end;

  begin
    MKosten:=0;
    if not self.Kantenliste.Leer then
      for Index:=0 to self.Kantenliste.Anzahl-1 do
        BestimmeMaxkosten(TMinimaleKostenkante(self.Kantenliste.Kante(Index)));
    MaxKosten:=Mkosten;
  end;

  procedure InverseKosten(Ka:TMinimalekostenkante);
  begin
    Ka.Kosten:=Max-Ka.Kosten;
  end;


begin
  Max:=Maxkosten;                                                           17)
  if not Kantenliste.Leer then
    for Zaehl:=0 to self.Kantenliste.Anzahl-1 do
      InverseKosten(TMinimaleKostenkante(Self.Kantenliste.Kante(Zaehl)));   18)
```

end;

Dazu wird bei 17) zunächst das Maximum der Kosten aller Kantenbestimmt.Die invertierten Kosten berechnen sich dann als Differenz des Maximums minus der Kantenkosten für jede Kante (18).Die invertierte Kosten werden dann als Kosten in jeder Kante gespeichert.

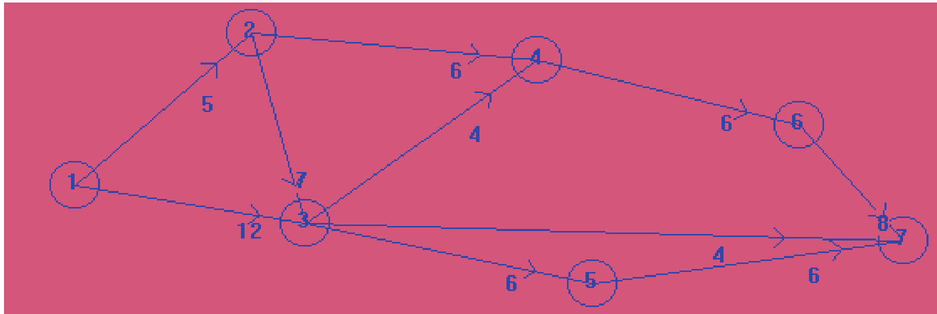
Die Lösung des MinimalenKosten-Problems für die invertierten Kosten ist die Lösung des MaximalenKostenproblems für die ursprünglichen Kosten.Bei 19) werden dann wieder die zwischengespeicherten ursprünglichen Kosten zurück-geladen.

Aufgabe C XIII.3:

Löse zu dem folgenden Graph das MinimaleKosten-Problem zum vorgegebenen Fluss 14 und den nachfolgend angegebenen Kosten der Kanten mit Hilfe des Algorithmus des Menüs Anwendungen/Minimale Kosten des Programms Knotengraph im Demomodus (oder eventuell durch den eigenen erstellten Quellcodes in der Entwicklungsumgebung je nach Konzeption).Die vorgegebenen Kanteninhalte sind die Schranken der Kanten.

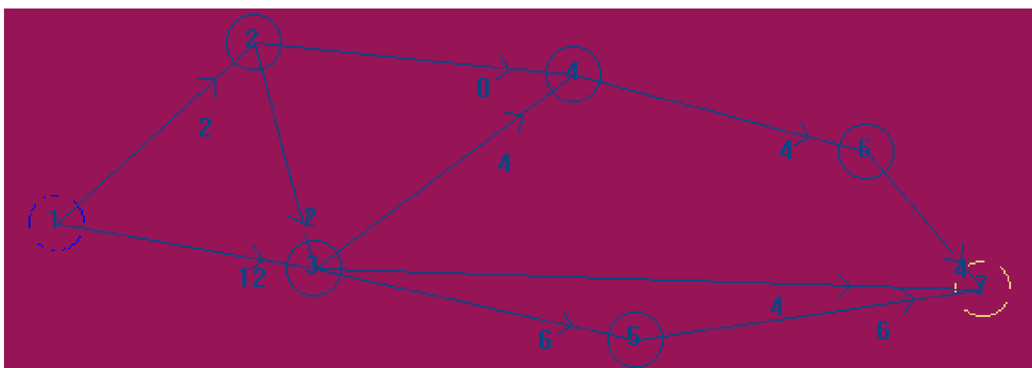
Kosten der Kanten:

Kante: 1 2 Kosten: 5
Kante: 1 3 Kosten: 3
Kante: 2 4 Kosten: 3
Kante: 2 3 Kosten: 0
Kante: 3 7 Kosten: 4
Kante: 3 5 Kosten: 2
Kante: 3 4 Kosten: 1
Kante: 4 6 Kosten: 2
Kante: 5 7 Kosten: 3
Kante: 6 7 Kosten: 4



G086.gra

Lösung:



Kante: 1 2 Fluss: 2 Flusskosten: 10 Schranke: 5 Kosten: 5
Kante: 1 3 Fluss: 12 Flusskosten: 36 Schranke: 12 Kosten: 3
Kante: 2 4 Fluss: 0 Flusskosten: 0 Schranke: 6 Kosten: 3
Kante: 2 3 Fluss: 2 Flusskosten: 0 Schranke: 7 Kosten: 0
Kante: 3 7 Fluss: 4 Flusskosten: 16 Schranke: 4 Kosten: 4

Kante: 3 5 Fluss: 6 Flusskosten: 12 Schranke: 6 Kosten: 2
Kante: 3 4 Fluss: 4 Flusskosten: 4 Schranke: 4 Kosten: 1
Kante: 4 6 Fluss: 4 Flusskosten: 8 Schranke: 6 Kosten: 2
Kante: 5 7 Fluss: 6 Flusskosten: 18 Schranke: 6 Kosten: 3
Kante: 6 7 Fluss: 4 Flusskosten: 16 Schranke: 8 Kosten: 4

Vorgegebener Fluss durch Quelle oder Senke: 14

Erreichter Fluss durch Quelle oder Senke: 14

Gesamtkosten: 120

Definition C XIII.2:

Unter einer Transportaufgabe versteht man folgendes Problem:

Von q Vertriebsstellen (Quellen) aus sind s Abnahmestellen (Senken) mit einer bestimmten Ware zu beliefern,so dass die Gesamtkosten beim Transport dieser Ware minimal unter allen Möglichkeiten sind.Jede Quelle ist dabei mit einer Senke durch einen Transportweg k_{ij} (von Quelle i zu Senke j) verbunden,dem jeweils Kosten K_{ij} für den Transport der Wareneinheit sowie eine nichtnegative Schranke c_{ij} als größtmögliche Warenmenge,die auf dem Weg transportiert werden kann,zugeordnet sind.
An jeder Quelle sei ein Vorrat von d_i Wareneinheiten vorhanden und an jeder Senke ein Bedarf von d_j Wareneinheiten.Der Transport der Ware nämlich die Warenmenge sei durch einen Fluss beschrieben.Der Fluss von Quelle i zu Senke j sei f_{ij}.

Es gilt dann:

1) $\sum_{j=q+1}^{q+s} f_{ij} \leq d_i$ für i=1...q

2) $\sum_{i=1}^{q} (-f_{ij}) \geq d_j$ für j=q+1....q+s; $d_j<0$

3) $0 \leq f_{ij} \leq c_{ij}$ für i=1...q;j=q+1..q+s

4)Minimiere die Funktion $\sum_{i=1}^{q} \sum_{j=q+1}^{q+s} K_{ij} f_{ij}$.

Der einlaufende Fluss (Wareneinheiten) in einen Knoten (Abnahmestelle) erhält dabei ein negatives Vorzeichen.

Das Tansportproblem heißt Hitchcockproblem,wenn alle Werte c_{ij} unendlich groß sind (d.h. es gibt keine Schranken) und außerdem in den Gleichungen 1) und 2) das Gleichheitszeichen gilt.

Das Hitchcockproblem heißt Zuordnungsproblem,wenn q=s ist und $d_i=1$ ist,$d_j=-1$ und zusätzlich f_{ij} entweder 0 oder 1 ist (für alle i=1...q; j=q+1...q+s).

Die genannten Probleme können mit Hilfe eines bipartiten Graphen dargestellt und anschließend mit Hilfe des Algorithmus Minimale Kosten gelöst werden.

Dazu werden die Quellen und die Senken als die verschiedenen Knotenmengen eines bipartiten Graphen aufgefaßt.Die Transportwege sind dann gerichtete Kanten k_{ij} ,die alle Quellenknoten mit allen Senkenknoten verbinden.Ihnen wird als nichtnegative Zahl jeweils die Kosten K_{ij} und die Schranke c_{ij} (als Datenfelder) zugeordnet.Ein Fluss $f_{ij}=f(k_{ij})$ beschreibt dann die Warenmenge,die vom Quellenknoten i zum Senkenknoten j transportiert wird.Gesucht ist ein Fluss,zu dem minimale Kosten gehören.

Um das Problem auf den Algorithmus Minimale Kosten zurückzuführen,ist jedoch ein einziger Quellen-und Senkenknoten erforderlich.Wie lassen sich außerdem die Vorrats- und Bedarfszahlen berücksichtigen?

Das Verfahren sei an Hand des folgenden Beispiels erläutert:

Aufgabe C XIII.4:(Einstiegaufgabe)

Fortgesetzt wird dazu die Anwendungsaufgabe C IX.4.(vgl. auch die vereinfachte Lösung bei Aufgabe C XIII.5.)

Der Aufgabentext wird dazu noch einmal wiederholt:

Gegeben ist folgender Graph,dessen Knoten Städte und dessen Kanten Straßen darstellen sollen.In den Fabriken der Städte A,B,C werden Ersatzteile produziert,die von den Fabriken in den Städten D,E und F benötigt werden.

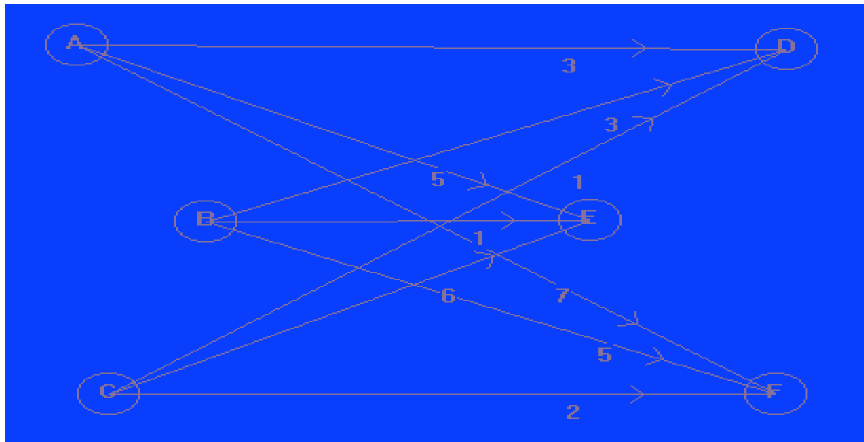
In A werden 5 Tausend,in B 3 Tausend und in C 4 Tausend Ersatzteile pro Tag produziert.In D werden 2 Tausend,in E 4 Tausend und in F 6 Tausend Ersatzteile pro Tag benötigt.Die Abnahmekapazität ist dabei gleich der Produktionskapazität.

Zwischen den Städten A,B und C einerseits und den Städten D,E und F andererseits liegt ein Straßennetz,dessen Straßen jeweils die im folgenden Graphen als Kantenschranken vorgegebenen Transportkapazitäten in Tausend pro Tag haben.

Wie ist der LKW-Verkehr bzw. dessen Ladekapazität zu organisieren,damit die produzierten Ersatzteile bei den Abnehmern in ausreichender Zahl ankommen?

(Für eine problemorientierte Erarbeitung siehe den entsprechenden Unterrichtsplan im Anhang F IV.)

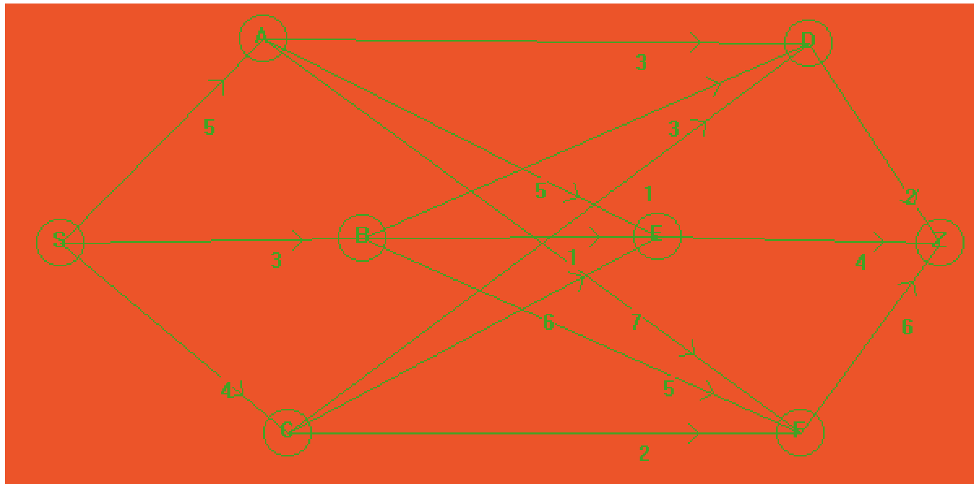
(Bei allen Graphen Zahlenangaben in Tausend)



G087.gra

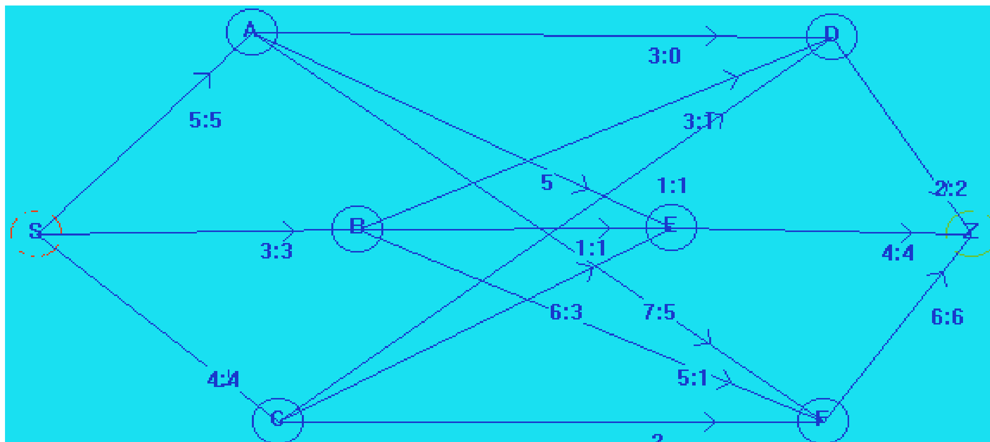
Die dortige Lösung:

Der Graph wird zu einem Netz durch Startknoten S und Zielknoten Z ergänzt.Diese Knoten werden jeweils durch Kanten mit den Knoten A,B und C bzw. D,E und F verbunden,deren Schranken den Produktionskapazitäten bzw. den Verbrauchskapazitäten entsprechen.



G049.gra

Anschließend ergibt der Algorithmus Maximaler Netzfluss im Menü Anwendungen die Lösung:



G049.gra

Der Fluss durch die einzelnen Kanten gibt die Transportkapazität der Wege pro Tag in Tausend zwischen den einzelnen Städten an.

Angabe in Tausend:

Kanten,Schranken und Fluss:

A->F Schranke: 7 Fluss: 5
A->E Schranke: 5 Fluss: 0
A->D Schranke: 3 Fluss: 0
B->E Schranke: 1 Fluss: 1
B->D Schranke: 3 Fluss: 1
B->F Schranke: 5 Fluss: 1
C->D Schranke: 1 Fluss: 1
C->E Schranke: 6 Fluss: 3
C->F Schranke: 2 Fluss: 0
S->A Schranke: 5 Fluss: 5
S->B Schranke: 3 Fluss: 3
S->C Schranke: 4 Fluss: 4
D->Z Schranke: 2 Fluss: 2
E->Z Schranke: 4 Fluss: 4
F->Z Schranke: 6 Fluss: 6

maximaler Gesamtfluss: 12 (Tausend)

Zusätzliche Aufgabenstellung:

Als neue Aufgabenstellung sollen nun den Straßen als Transportwegen noch zusätzliche Kosten (in DM oder €) pro Ersatzteil pro Tag zugeordnet werden:

Kante: A F Kosten: 6
Kante: A E Kosten: 1
Kante: A D Kosten: 2
Kante: B E Kosten: 5
Kante: B D Kosten: 3
Kante: B F Kosten: 4
Kante: C D Kosten: 2
Kante: C E Kosten: 3
Kante: C F Kosten: 4

Gefragt ist jetzt,wieviel Ersatzteile von A,B,C auf welchen Straßen nach D,E und F transportiert werden sollen,damit die Produktions-und Abnahmekapazität jeweils ausgeschöpft werden,die Transportkapazitäten (Schranken) berücksichtigt werden, und noch **zusätzlich die Transportkosten minimal unter allen Möglichkeiten sind.**

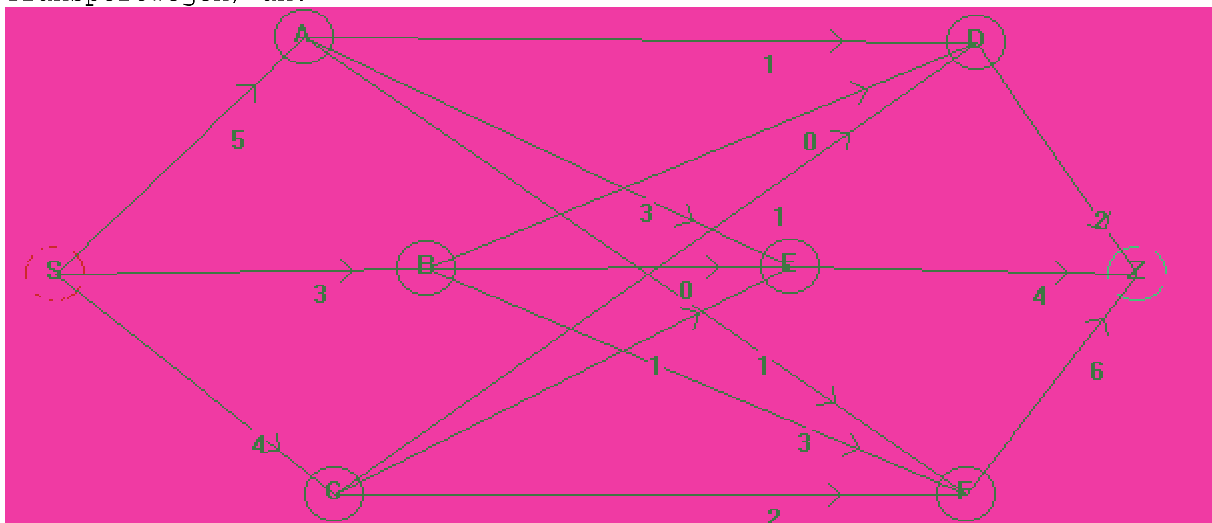
Lösung:

Es wird wieder wie oben ein Graph mit Quellenknoten A und Senkenknoten Z erzeugt,deren (ausgehende bzw. einlaufende) Kanten wie oben zu den Knoten A,B,C bzw. D,E und F führen und mit den obigen Schranken bewertet sind.

Zusätzlich werden jetzt jeder der Kanten des ursprünglichen Graphen die oben genannten Kosten zugewiesen.

Den neu hinzugekommenen Kanten zum Quellen-und Senkenknoten werden jeweils **die Kosten 0** zugeordnet.

Jetzt wird auf dem Graphen ein Fluss gemäß dem Algorithmus Minimale Kosten (Menü Anwendungen/Minimale Kosten) erzeugt.Als Vorgabefluss wird jetzt die Zahl 12 (Tausend) gewählt,die gerade der Summe der Produktions-und Abnahmekapazitäten entspricht.Die ermittelte Lösung für Fluss und Kosten ist die gesuchte Lösung des gestellten Problems,da es sich um einen Fluss handelt,der minimal bezüglich der Kosten ist,und da die zusätzlichen Kanten keine Kosten verursachen.Außerdem werden die Schrankenbedingungen eingehalten und durch die zusätzlichen Kanten zum Quellen-und Senkenknoten fließt gerade soviel Fluss,wie es jeweils den Produktions- und Abnahmekapazitäten in den Knoten (Städten) A,B,C und D,E und F entspricht.Der folgende Graph zeigt den Fluss (d.h. die Anzahl der Ersatzteile pro Tag in Tausend auf den Transportwegen) an:



G049.gra

Angaben in Tausend bis auf Kosten:

Kante: A F Fluss: 1 Flusskosten: 6 Schranke: 7 Kosten: 6
Kante: A E Fluss: 3 Flusskosten: 3 Schranke: 5 Kosten: 1
Kante: A D Fluss: 1 Flusskosten: 2 Schranke: 3 Kosten: 2
Kante: B E Fluss: 0 Flusskosten: 0 Schranke: 1 Kosten: 5
Kante: B D Fluss: 0 Flusskosten: 0 Schranke: 3 Kosten: 3
Kante: B F Fluss: 3 Flusskosten: 12 Schranke: 5 Kosten: 4
Kante: C D Fluss: 1 Flusskosten: 2 Schranke: 1 Kosten: 2
Kante: C E Fluss: 1 Flusskosten: 3 Schranke: 6 Kosten: 3
Kante: C F Fluss: 2 Flusskosten: 8 Schranke: 2 Kosten: 4
Kante: S A Fluss: 5 Flusskosten: 0 Schranke: 5 Kosten: 0
Kante: S B Fluss: 3 Flusskosten: 0 Schranke: 3 Kosten: 0
Kante: S C Fluss: 4 Flussosten: 0 Schranke: 4 Kosten: 0
Kante: D Z Fluss: 2 Flusskosten: 0 Schranke: 2 Kosten: 0
Kante: E Z Fluss: 4 Flusskosten: 0 Schranke: 4 Kosten: 0
Kante: F Z Fluss: 6 Flusskosten: 0 Schranke: 6 Kosten: 0

Vorgegebener Fluss durch Quelle oder Senke: 12 (Tausend)

Erreichter Fluss durch Quelle oder Senke: 12 (Tausend)

Gesamtkosten: 36 (minimale Kosten) (Tausend)

Die allgemeine Lösung des Transportproblems für einen bipartiten Graphen besteht also daraus,einen zusätzlichen (Haupt-)Quellen-und Senkenknoten zu erzeugen,von diesen Knoten ausgehende und eingehende Kanten zu den Quellenknoten und Senkenknoten des vorgegebenen Graphen zu erzeugen,deren Schranken jeweils den Produktions-und Abnahmekapazitäten entsprechen,und deren Kosten Null sind.Außerdem sind den Kanten des ursprünglichen Graphen die vorgegebenen Kosten (und Schranken) zuzuweisen.(Beim Hitchcockproblem sind die Schranken unendlich.)

Danach ist ein Fluss mit minimalen Kosten gemäß dem obigen Algorithmus zu erzeugen.

Bequemer für die Lösung des Problems wäre es,wenn die Erzeugung des neuen Quellen-und Senkenknotens sowie der zusätzlichen Kanten automatisch erfolgen würde.

Dann müßten diesen Kanten nur noch die Produktions-und Abnahmekapazitäten zugeordnet werden.(Die Zuweisung der Kosten Null würde automatisch geschehen.)Den ursprünglichen Kanten des Graphen müßten noch die Kosten zugewiesen werden.

Diese Verbesserungen sind im Algorithmus des Menüs Anwendungen/ Transportproblem des Programms Knotengraph verwirklicht.

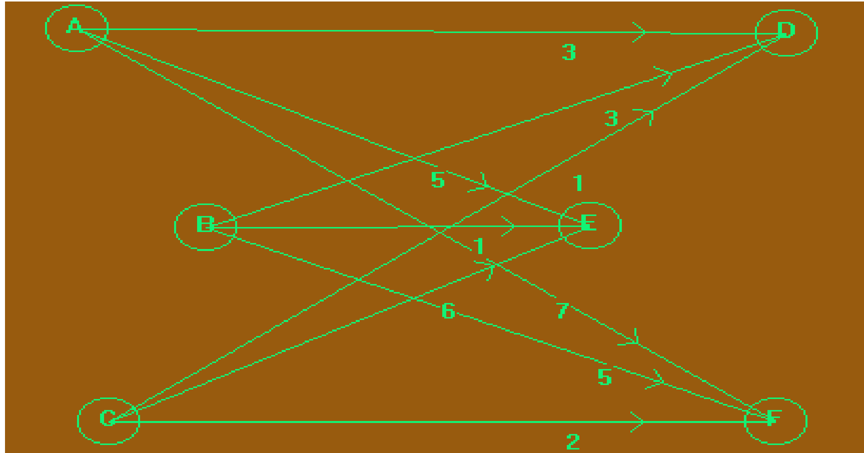
Außerdem kann hier gewählt werden,ob ein Hitchcockproblem gelöst werden soll.Dann werden die Schranken der Kanten nicht berücksichtigt (gleich unendlich 1E32 gesetzt).Ansonsten wird wieder der gleiche Algorithmus nach Busacker und Gowen (dieselben Methoden) wie oben beschrieben zur Lösung benutzt.

<u>Aufgabe C X.III.5:(Einstiegsaufgabe)</u>

Löse die obige gestellte Aufgabe mit dem ursprünglichen Graphen und den oben angegebenen Kosten direkt mit dem Demomodus des Algorithmus des Menüs Anwendungen/Transportproblem,und beobachte die Erzeugung des Quellen-und Senkenknoten sowie zusätzlicher Kanten und den Ablauf des Algorithmus.Gib als Schranken jeweils die Produktions-und Abnahmekapazitäten der verschiedenen Knoten (Städte) (in Tausend) ein.

Ursprünglicher Graph:

Zahlenangaben in Tausend:

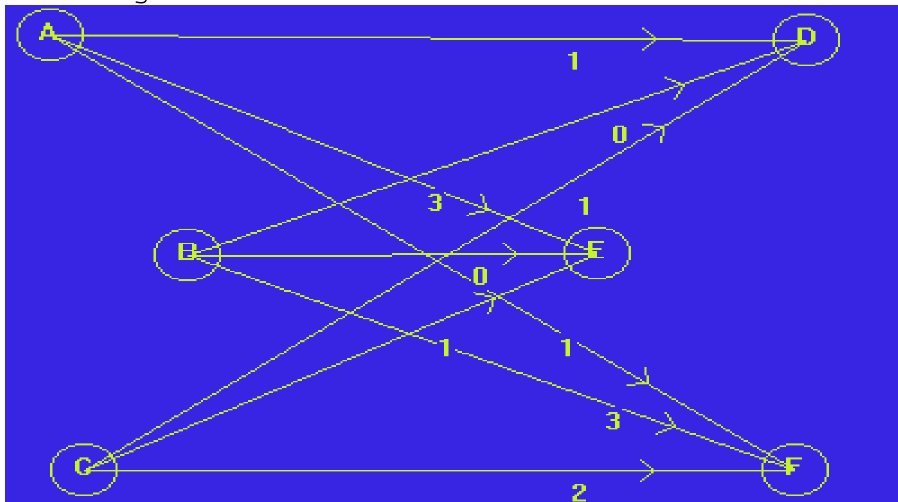


G087.gra

Lösung:wie oben

(Die Lösung ist aber leichter zu überblicken,weil nur die relevanten Zahlen angegeben werden.)

Zahlenangaben in Tausend:



G087.gra

Zahlenangaben in Tausend bis auf Kosten:

Kante: A F Fluss: 1 Flusskosten: 6 Schranke: 7 Kosten: 6
Kante: A E Fluss: 3 Flusskosten: 3 Schranke: 5 Kosten: 1
Kante: A D Fluss: 1 Flusskosten: 2 Schranke: 3 Kosten: 2
Kante: B E Fluss: 0 Flusskosten: 0 Schranke: 1 Kosten: 5
Kante: B D Fluss: 0 Flusskosten: 0 Schranke: 3 Kosten: 3
Kante: B F Fluss: 3 Flusskosten: 12 Schranke: 5 Kosten: 4
Kante: C D Fluss: 1 Flusskosten: 2 Schranke: 1 Kosten: 2
Kante: C E Fluss: 1 Flusskosten: 3 Schranke: 6 Kosten: 3
Kante: C F Fluss: 2 Flusskosten: 8 Schranke: 2 Kosten: 4

Vorgegebener Fluss durch Quelle oder Senke: 12 (Tausend)

Erreichter Fluss durch Quelle oder Senke: 12 (Tausend)

Gesamtkosten: 36 (minimale Kosten) (Tausend)

Aufgabe C XIII.6:

Löse das folgende Hitchcockproblem mit Hilfe des Algorithmus des Menüs Anwendungen/Transportproblem des Programms Knotengraph.

(Beispiel nach Lit. 4,S.110)

Vorgegeben werden q=3 Quellen und s=4 Senken.

Die Schranken sind unendlich (9.999 E31).Die Kosten werden als Matrix folgendermaßen gegeben:

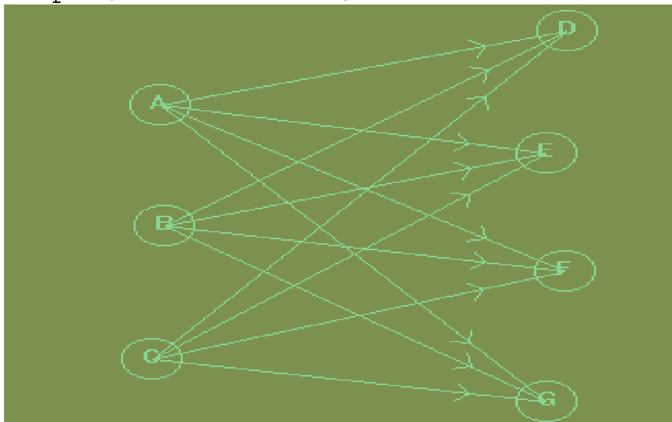
$$K_{ij}=\begin{pmatrix} 0 & -1 & 2 & 4 \\ 1 & 4 & 1 & 0 \\ 0 & 2 & 2 & 3 \end{pmatrix}$$

Die Produktions-und Abnahmemengen sind gegeben durch durch die Vektoren:

$$\vec{d} = \begin{pmatrix} 5 \\ 3 \\ 6 \end{pmatrix} \quad \textbf{und} \quad \vec{e} = \begin{pmatrix} 1 \\ 6 \\ 3 \\ 4 \end{pmatrix}$$

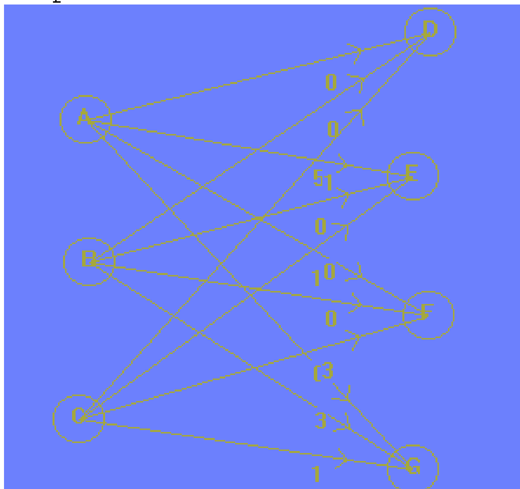
Lösung:

Graph (ohne Schranken):



G088.gra

Graph mit Fluss zu minimalen Kosten:



G088.gra

Kante: A D Fluss: 0.000 Flusskosten: 0.000 Schranke: 9.999E31 Kosten: 0.000
Kante: A E Fluss: 5.000 Flusskosten:-5.000 Schranke: 9.999E31 Kosten: 1.000
Kante: A F Fluss: 0.000 Flusskosten: 0.000 Schranke: 9.999E31 Kosten: 2.000
Kante: A G Fluss: 0.000 Flusskosten: 0.000 Schranke: 9.999E31 Kosten: 4.000
Kante: B D Fluss: 0.000 Flusskosten: 0.000 Schranke: 9.999E31 Kosten: 1.000
Kante: B E Fluss: 0.000 Flusskosten: 0.000 Schranke: 9.999E31 Kosten: 4.000
Kante: B F Fluss: 0.000 Flusskosten: 0.000 Schranke: 9.999E31 Kosten: 1.000
Kante: B G Fluss: 3.000 Flusskosten: 0.000 Schranke: 9.999E31 Kosten: 0.000
Kante: C D Fluss: 1.000 Flusskosten: 0.000 Schranke: 9.999E31 Kosten: 0.000
Kante: C E Fluss: 1.000 Flusskosten: 2.000 Schranke: 9.999E31 Kosten: 2.000
Kante: C F Fluss: 3.000 Flusskosten: 6.000 Schranke: 9.999E31 Kosten: 2.000
Kante: C G Fluss: 1.000 Flusskosten: 3.000 Schranke: 9.999E31 Kosten: 3.000

Vorgegebener Fluss durch Quelle oder Senke: 14.000

Erreichter Fluss durch Quelle oder Senke: 14.000

Gesamtkosten: 6.000 (minimale Kosten)

Wie man sieht,können die Kosten auch negativ vorgegeben werden.

Transport- bzw. Hitchcockaufgaben von der Art der letzten beiden Anwendungsbeispiele sind Spezialfälle der folgenden allgemeineren Aufgabenstellung:

Lineare Optimierungsaufgabe:

Gegeben sei eine Zielfunktion

$$F = \vec{K}\vec{f} + b$$

F soll minimiert werden unter den Nebenbedingungen

1) $H\vec{f} = \vec{d}$

2) $g_i \le f_i \le c_i$; für i=1..n,mit g_i und c_i nichtnegative Schranken

H ist dabei eine Matrix der Dimension von Vektor $\vec{f}$ und $\vec{d}$.

Probleme solcher Art können mit Hilfe des Simplex-Verfahrens der linearen Optimierung mit Hilfe der Umformung von Matrizen gelöst werden.

Die Bedingung 1) muß für Flussprobleme auf Graphen stets umgeformt werden können in die äquivalente Bedingung:

3) $\sum_{j \in E_{+i}} f_j - \sum_{i \in E_{-i}} f_i = d_i$

Dabei bedeutet E_{+i} und E_{-i},die vom i.ten Knoten ausgehenden und in ihn einlaufenden Kantenflüsse.

Falls $d_i>0$ ist,handelt es sich bei dem Knoten um eine Quelle,für $d_i<0$ ($e_i<0$) um eine Senke.Der Fall $d_i=0$ kann als (innerer) Knoten gedeutet werden (Flussbedingung).

Nur diese drei Fälle sind für einen Fluss in einem Graphen möglich.

Aus Bedingung 3) folgt,dass die Zahlen der Matrix H nur die Werte 0,1 oder -1 haben können.Die Matrix H heißt Graphmatrix des Flussgraphen.

Die Bedingung 3) stellt gerade eine Zeile der Matrix dar.Diese Zeile läßt sich dem Knoten mit dem Wert d_i zuordnen.Falls $d_i>0$ ist,ist der Knoten ein

Quellenknoten,bei d_i=0 ein innerer Knoten (Zirkulationsknoten) und bei d_i<0 ein Senkenknoten.
Die Spalten der Matrix H sind gerade den Flussvariablen f_i bzw. f_j zugewiesen, und können den Kanten,durch die dieser Fluss fließt,zugeordnet werden. In jeder Spalte können dann nur außer Nullen entweder eine einzige 1 oder eine einzige -1 oder eine 1 und eine -1 vorhanden sein.Der Fluss f_i fließt nämlich im gerichteten Graph von einem Anfangsknoten zu einem Endknoten.Sind beides innere Knoten (Zirkulationsknoten),erhalten gerade diese beiden Knotenzeilen in der Kantenspalte den Wert 1 (Anfangsknoten) bzw. -1 (Endknoten),da diese Anfang und Ende des Flusses sind.Die Spalte eines Quellenknoten des Graphen enthält nur in der entsprechenden Kantenzeile den Wert 1,weil von ihm nur ein Fluss ausgeht,und ein Senkenknoten enthält dort nur den Wert -1,weil in ihn nur ein Fluss einläuft.

Die Optimierungsaufgabe ist als Flussproblem auf Graphen genau dann stets lösbar,wenn die Summe der Elemente des Vektors $\vec{d}$ gleich Null ist,weil dann der aus den Quellen ausfließende Fluss gleich dem Fluss ist,der in die Senken hineinfließt.

Der Elemente des Vektor $\vec{K}$ entsprechen die Kosten der Kanten.Der Wert für b kann oBdA. gleich Null gewählt werden,da er als additive Konstante lediglich die Höhe der minimalen Kosten verschiebt.

Falls die Schranke g_i=0 ist,entsteht dann das obige Transportproblem bzw. das Hitchcockproblem für $c_i=\infty$ auf einem bipartiten Graphen,falls man die Gleichung 3) in die Gleichungen

$$\sum_{j=q+1}^{q+s} f_{ij} = d_i \quad \text{für } i=1...q$$

$$\sum_{i=1}^{q} (-f_{ij}) = d_j \quad \text{für } j=q+1....q+r \text{ mit } d_j<0$$

separieren kann.

Auf diese Weise können Spezialfälle der linearen Optimierung mit Hilfe des Transportproblems ohne Simplexalgorithmus gelöst werden.

Bemerkung:

Den Simplexalgorithmus gibt es auch als Graphenalgorithmus (mittels Matrixumformung),wobei gemäß der Gleichung 3) ein allgemeiner Graph (nicht unbedingt bipartiter) gemäß der obigen Deutung der d_i nach Knotenarten benutzt werden kann.(Lediglich die Matrix H muß dann nur 0,1 oder -1 enthalten.) (vgl. Lit 4)

Aber auch ohne Simplexalgorithmus mittels des Algorithmus des Menüs Anwendungen/Transportproblem des Programms Knotengraph lassen sich solche Aufgaben mit allgemeineren Graphen lösen,wenn man einen geeigneten Flussgraph konstruiert,wie die folgende Aufgabe zeigt:

(Beispiel aus Lit 4, S.34,dort mittels Simplexalgorithmus für Graphen)

Aufgabe C XIII.7:(Einführungsaufgabe)

Die Funktion F= $f_1 + 3f_2 + f_3 + 2f_4 + 2f_5$ soll minimiert werden,unter den Nebenbedingungen:

A)

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 \end{pmatrix} \vec{f} = \begin{pmatrix} 3 \\ 0 \\ -1 \\ -2 \end{pmatrix}$$

Eventuelle Ungleichungen an dieser Stelle können mittels Schlupfvariablen, wie üblich,in Gleichungen umgeformt werden.

Außerdem:

B)

$0 \le f_1 \le 2$
$0 \le f_2 \le 3$
$0 \le f_3 \le 3$
$0 \le f_4 \le 2$
$0 \le f_5 \le 2$
$0 \le f_6 \le \infty$

Lösung:

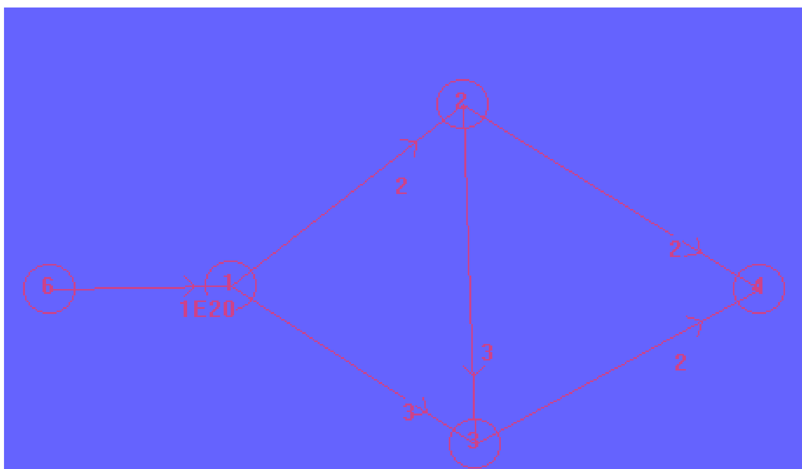
Auf Grund den oben erwähnten Eigenschaften der Graphmatrix,hat der entsprechende Flussgraph 4 Knoten 1,2,3, und 4:

Knoten 1 ist ein Quellenknoten mit dem Fluss 3.
Knoten 2 ist ein innerer Knoten (Zirkulationsknoten).
Knoten 3 ist ein Senkenknoten mit dem Fluss -1.
Knoten 4 ist ein Senkenknoten mit dem Fluss -2.

Die Summe aller Senken-und Quellenflüsse ist gleich Null.Also gibt es eine Lösung des Flussproblems.

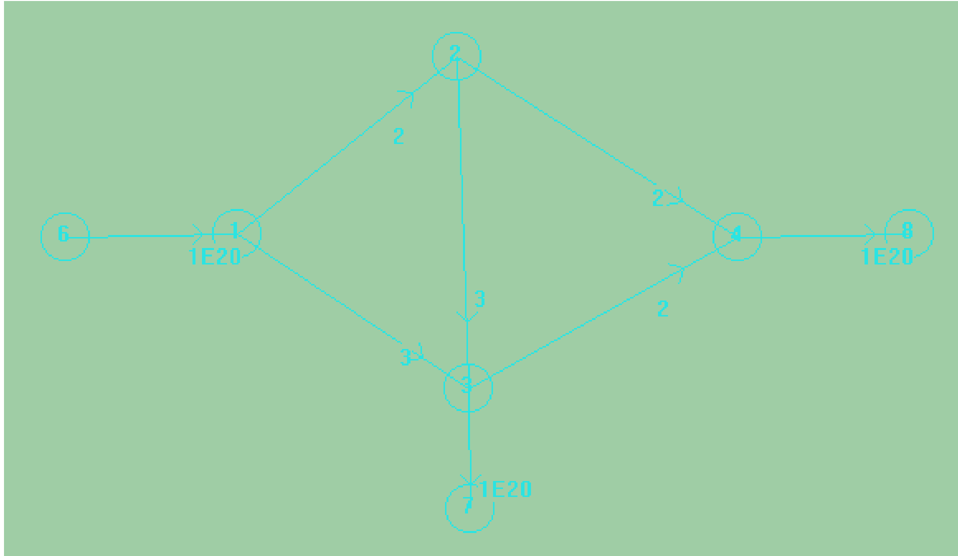
Wenn man den Zeilen der Matrix jeweils die Knoten 1 bis 4 und den Spalten die Flusskanten 1 bis 6 zuordnet,ist die folgende Struktur des Graphen festgelegt,wobei mit der Bedingung B auch noch die Schranken der Kanten berücksichtigt werden können.
(Eine Kante verläuft immer vom Anfangsknoten,dem die 1 zugeordnet ist,zum Endknoten,dem die -1 zugeordnet ist.Sonst sind es Quellen oder Senkenknoten.)

Graph:



Um ein Transportproblem auf diesem Graphen lösen zu können,werden jetzt den Senken 3 und 4 noch zusätzliche Senkenknoten 7 und 8 zugeordnet,deren

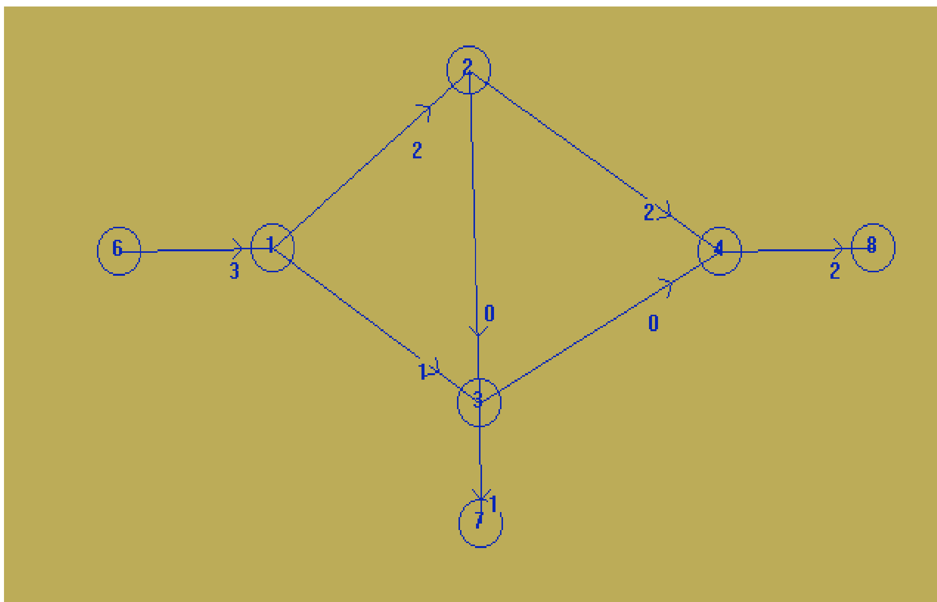
Schranken wie die Schranke der Kante 6->1 als unendlich (hier 1E20=9.9999.. E19) gewählt werden:
(Der Quellenknoten ist mit Knoten 6 schon vorhanden und braucht nicht mehr extra erzeugt zu werden.Falls zu f_6 zusätzlich eine Schranke oder Kosten berücksichtigt werden müssen,wird ein zusätzlicher Knoten 5 als Quellenknoten mit einer Kante nach Knoten 6 und der Schranke 1 E20,d.h. unendlich benötigt.)



G090.gra

Jetzt kann das Transportproblem für diesen Graphen gelöst werden.Dem Knoten 6 (bzw. 5 s.o.) wird dabei der Fluss 3 als Schranke,den Knoten 7 und 8 die Flüsse 1 und 2 als Schranken zugeordnet.Als Vorgabefluss wird der (maximale) Fluss 3 (als Summe der Flüsse aller Quellen) gewählt.
Außerdem müssen die Kosten der Kanten des ursprünglichen Graphen eingegeben werden.Diese sind die Koeffizienten der zu minimierenden Funktion F und betragen für die einzelnen Kanten 1;3;1;2;2 und für die 6.Kante 0,da f_6 nicht in der Funktion f als Variable vorkommt.
Die Kosten der neu erzeugten Kanten 3->7 und 4->8 werden 0 gesetzt.

Die Lösung des Transportproblems durch Start des Menü Anwendungen/Transportproblem des Programms Knotengraph ergibt dann:



G090.gra

Kante: 1 2 Fluss: 2 Flusskosten: 2 Schranke: 2 Kosten: 1
Kante: 1 3 Fluss: 1 Flusskosten: 3 Schranke: 3 Kosten: 3
Kante: 2 4 Fluss: 2 Flusskosten: 4 Schranke: 2 Kosten: 2
Kante: 2 3 Fluss: 0 Flusskosten: 0 Schranke: 3 Kosten: 1
Kante: 3 4 Fluss: 0 Flusskosten: 0 Schranke: 2 Kosten: 2
Kante: 6 1 Fluss: 3 Flusskosten: 0 Schranke: 9E19 Kosten: 0
Kante: 3 7 Fluss: 1 Flusskosten: 0 Schranke: 9E19 Kosten: 0
Kante: 4 8 Fluss: 2 Flusskosten: 0 Schranke: 9E19 Kosten: 0

Vorgegebener Fluss durch Quelle oder Senke: 3

Erreichter Fluss durch Quelle oder Senke: 3

Gesamtkosten: 9 (Minimaler Wert für F)

Auf diese Weise lassen sich also lineare Optimierungsaufgaben,deren Matrix als Fluss-Graphmatrix gedeutet werden kann,auf allgemeinen Flussgraphen,die nicht unbedingt bipartit sein müssen,(ohne Simplexalgorithmus) lösen.

Als Spezialfall des Hitchcockproblems kann wiederum das Zuordnungsproblem bzw. das Problem,ein optimales Matching herzustellen,angesehen werden.
Hierbei ist die Anzahl der Quellen gleich der Anzahl der Senken q=s,die Schranken auf dem ursprünglichen bipartiten Graph sind ohne Bedeutung (unendlich groß) und die Schranken für die zusätzlich erzeugten Kanten für den zusätzlichen Senken-und Quellenknoten werden stets auf 1 gesetzt (Produktions-bzw. Abnahmemengen).

Der Begriff des Matching wurde in Kapitel X erläutert.(vgl. die dortigen Definitionen).

Die dortige Aufgabenstellung wird jetzt erweitert,und es geht darum ein optimales (und perfektes) **Matching** auf einem bipartiten Graphen,dessen Kanten durch Kosten bewertet sind,zu finden,das **noch zusätzlich Kostenmaximal oder Kostenminimal ist.**

Definition C XIII.2:

Ein maximales Matching auf einem Graphen,dessen Kanten eine Kosten-Bewertung tragen,heißt optimal bezüglich der Kostenbewertung,wenn die Summe der Kosten dieser Kanten unter allen Möglichkeiten ein maximales Matching zu erzeugen entweder maximal oder minimal ist.

Bemerkung:

Im folgenden (wie schon beim Transportproblem) ist jeweils zu unterscheiden zwischen den (Gesamt-) Quellen- bzw. Senkenknoten,die durch den Algorithmus des Hitchcockproblems zusätzlich erzeugt werden und den durch den gerichteten,bipartiten Graph schon vorgegebenen Quellen-und Senkenknoten,die durch die Kantenrichtungen des Graphen von selbst bestimmt sind.

Aufgabe C XIII.8:(Einführungsaufgabe)

Drei Schüler sollen drei Mathematikaufgaben aus den unterschiedlichen Bereichen Analysis (A),Lineare Algebra (L) und Stochastik (S) bearbeiten.Dabei soll jeder Schüler eine Aufgabe zur Lösung erhalten.Die Kenntnisse der Schüler sind in den drei Gebieten unterschiedlich gut und schnell.Die Güte und Schnelligkeit kann vom Lehrer auf Grund des Lösens von Aufgaben in der Vergangenheit durch eine Wertungszahl angegeben werden.
Welche Aufgabe sollte jetzt welchem Schüler zur Lösung zugeteilt werden,damit alle drei Aufgaben (z.B. in einem Wettbewerb) von den drei Schülern möglichst richtig und schnell gelöst werden?

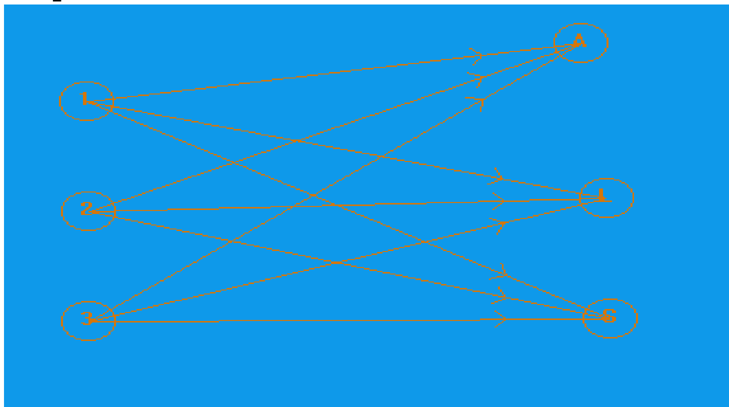
Gütezahlen:	Aufgabe A	L	S
Schüler 1:	12	6	7
Schüler 2:	2	5	10
Schüler 3:	8	5	6

Erstelle einen bipartiten Transportgraphen mit der obigen Gütezahl als Kosten-Kantenbewertung,setze die Schranken der Kanten,die vom durch den Algorithmus zusätzlich erzeugten Gesamt-Quellenknoten ausgehen bzw. in den durch den Algorithmus zusätzlich erzeugten Gesamt-Senkenknoten führen auf 1 (der Wert der übrigen Schranken ist unendlich),löse danach das Hitchcockproblem auf dem erstellten Graphen mit den Optionen Maximale Kosten und ganzzahlig,und ermittle die Kanten im ursprünglichen Graph,deren Fluss gleich 1 sind.

Kosten der Kanten:

Kante: 1 L Kosten: 6
Kante: 1 A Kosten: 12
Kante: 1 S Kosten: 7
Kante: 2 A Kosten: 2
Kante: 2 L Kosten: 5
Kante: 2 S Kosten: 10
Kante: 3 A Kosten: 8
Kante: 3 L Kosten: 5
Kante: 3 S Kosten: 6

Graph:



G091.gra

Lösung:

G091.gra

Kante: 1 L Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 6
Kante: 1 A Fluss: 1 Flusskosten: 12 Schranke: 9E31 Kosten: 12
Kante: 1 S Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 7
Kante: 2 A Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 2
Kante: 2 L Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 5
Kante: 2 S Fluss: 1 Flusskosten: 10 Schranke: 9E31 Kosten: 10
Kante: 3 A Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 8
Kante: 3 L Fluss: 1 Flusskosten: 5 Schranke: 9E31 Kosten: 5
Kante: 3 S Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 6

Vorgegebener Fluss durch Quelle oder Senke: 3

Erreichter Fluss durch Quelle oder Senke: 3

Gesamtkosten: 27

Die Kanten mit dem Fluss 1 sind die Kanten des optimalen Matching.

Verbale Beschreibung:

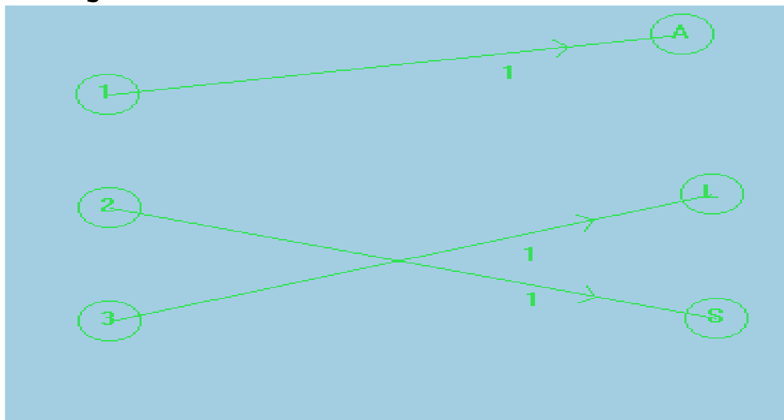
Da der Fluss von/zu jedem Quellen-und Senkenknoten des gegebenen paaren Graphen jeweils nur 1 beträgt,kann und muß nur jeweils eine ausgehende bzw. einlaufenden Kante den Fluss 1 erhalten.(Option ganzzahliger Fluss) Dadurch wird jeweils ein Matching erzeugt,und es werden genau drei Kanten ausgewählt,weil als Vorgabefluß (automatisch) 3 gewählt wurde.Das Matching ist also maximal (und perfekt).Außerdem ist es maximal,weil ein Kostenmaximaler Fluss erzeugt wurde,und die Kanten zum Quellen-und Senkenknoten die Kosten Null tragen.

Löse dieselbe Aufgabe jetzt mittels des Algorithmus des Menüs Anwendungen/ Optimales Matching des Programms Knotengraph.

Der Algorithmus enthält jetzt noch zusätzlich folgende Vereinfachungen:

Die Schranken für die Kanten,die zu den Quellen- und von den Senkenknoten des Graphen vom (Gesamt-)Quellenknoten bzw. zum (Gesamt-)Senkenknoten verlaufen,brauchen jetzt nicht mehr eingegeben zu werden und werden automatisch 1 gesetzt,es wird stets die ganzzahlige Flussoption gewählt, und als Ergebnis wird jetzt ein Graph angezeigt,der nur noch aus den Kanten des Matchings besteht.(Die anderen Kanten werden eliminiert.Der urspüngliche Graph wird allerdings zwischengespeichert und anschließend wieder hergestellt.)

Lösung:



G092.gra

Kante: 1 L Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 6
Kante: 1 A Fluss: 1 Flusskosten: 12 Schranke: 9E31 Kosten: 12
Kante: 1 S Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 7
Kante: 2 A Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 2
Kante: 2 L Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 5
Kante: 2 S Fluss: 1 Flusskosten: 10 Schranke: 9E31 Kosten: 10
Kante: 3 A Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 8
Kante: 3 L Fluss: 1 Flusskosten: 5 Schranke: 9E31 Kosten: 5
Kante: 3 S Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 6

Vorgegebener Fluss durch Quelle oder Senke: 3

Erreichter Fluss durch Quelle oder Senke: 3

Gesamtkosten: 27

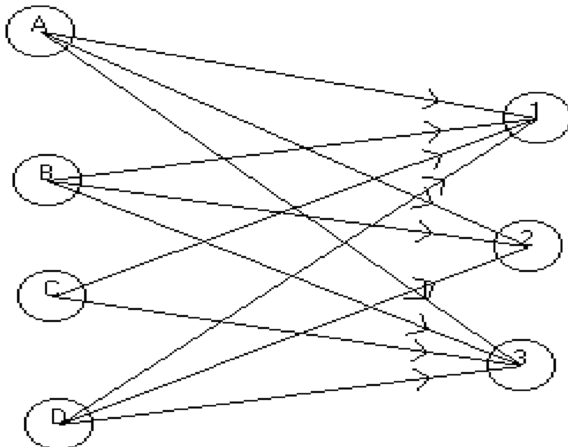
Schüler 1 erhält die Aufgabe A,Schüler 2 die Aufgabe S und Schüler 3 die Aufgabe L.

Aufgabe C XIII.9:

Eine Firma hat 3 Aufträge für Maschinenteile auszuführen.Dazu stehen 4 Maschinen A,B,C und D zur Verfügung.Jeder Auftrag kann nur auf einer Maschine durchgeführt werden.Der Auftrag 2 kann nicht auf Maschine C gefertigt werden.
Bei der Fertigung der Aufträge ergeben sich auf den verschiedenen Maschinen verschiedene Kosten.Welche Auftrag muß welcher Maschine zugeteilt werden,damit die Gesamtkosten minimal sind und die Aufträge parallel bearbeitet werden können?
Löse die Aufgabe mit dem Algorithmus des Menüs Anwendungen/Optimales Matching von Knotengraph mit der Option Minimale Kosten im Demomodus.Erzeuge einen geeigneten Graphen und berücksichtige dabei die folgenden Kosten:

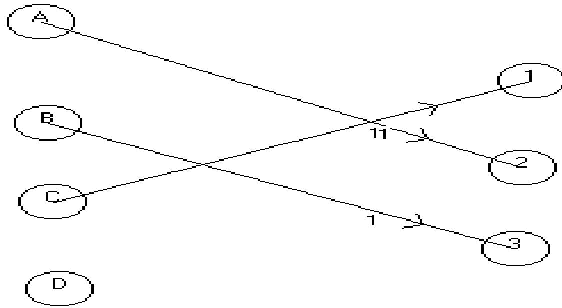
Maschine: / Auftrag:	1	2	3
A	50	66	81
B	55	70	78
C	42	-	72
D	57	68	80

Graph:



G093.gra

Lösung:



G094.gra

Kante: A 1 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 50
Kante: A 2 Fluss: 1 Flusskosten: 66 Schranke: 9E31 Kosten: 66
Kante: A 3 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 81
Kante: B 1 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 55
Kante: B 2 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 70
Kante: B 3 Fluss: 1 Flusskosten: 78 Schranke: 9E31 Kosten: 78
Kante: C 1 Fluss: 1 Flusskosten: 42 Schranke: 9E31 Kosten: 42
Kante: C 3 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 72
Kante: D 1 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 57
Kante: D 2 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 68
Kante: D 3 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 80

Vorgegebener Fluss durch Quelle oder Senke: 4

Erreichter Fluss durch Quelle oder Senke: 3

Gesamtkosten: 186 (minimale Kosten)

A bearbeitet 2,B bearbeitet 3 und C bearbeitet 1.

Hier liegt der Fall vor,dass das Matching nicht perfekt sein kann,weil die Quellenzahl ungleich der Senkenzahl ist.Dann ist auch der Fluss durch die Quellen ungleich dem Fluss durch die Senken.Auch dieser Fall wird vom Algorithmus gelöst.Ebenso kann beim Transportproblem mit verschiedenen Flüssen gearbeitet werden.Als Vorgabefluss wird dann immer der kleinste Fluss benutzt.

Bei den Algorithmen Transportproblem und Optimales Matching müssen jeweils ein zusätzlicher (Gesamt-)Quellen-und Senkenknoten sowie die mit diesen Knoten inzidenten Kanten samt Schranken zusätzlich erzeugt werden.Das Gleiche gilt für das im nächsten Kapitel zu besprechende Chinesische Briefträger-Problem.

Die folgende Methode ErzeugeQuellenundSenkenknotensowieKanten leistet das Verlangte:

<u>Der Quellcode:</u>

```
function TMinimaleKostengraph.ErzeugeQuellenundSenkenknotensowieKanten
(var Quelle,Senke:TInhaltsknoten;var Fluss:Extended;Art:char):Boolean;
label NaechsteKante,Marke1,Marke2;
var Ka:TMinimaleKostenKante;
    Index:Integer;
    Kno:TInhaltsknoten;
    Str:String;
    FlussQuellenzaehler,FlussSenkenzaehler:Extended;
```

```
   FlussOk,Eingabe:Boolean;

begin
  FlussOK:=true;
  Quelle:=TInhaltsknoten.Create;                                              20)
  Quelle.X:=0;
  Quelle.Y:=10;
  Quelle.Wert:='+';
  Senke:=TInhaltsknoten.Create;
  Senke.X:=595;
  Senke.Y:=10;
  Senke.Wert:='-';
  FuegeKnotenein(Quelle);
  FuegeKnotenein(Senke);
  FlussQuellenzaehler:=0;                                                     21)
  Flusssenkenzaehler:=0;
  if not Self.Knotenliste.Leer then
    for Index:=0 to self.Knotenliste.Anzahl-1 do
    begin
      Kno:=TInhaltsknoten(self.Knotenliste.Knoten(Index));
      if (Kno=Quelle) or (Kno=Senke) then goto NaechsteKante;
      if (((Art='t') or(Art='o')) and Kno.EingehendeKantenliste.Leer)         22)
        or ((Art='b') and (Kno.Grad(true)<0))                                 23)
      then
      begin
        Ka:=TMinimaleKostenkante.Create;
        Ka.Kosten:=0;
        Ka.Weite:=0;
        Ka.Wert:='U';
        Ka.Richtung:=false;
        case Art of
          't':begin                                                           24)
                Marke1:
                Eingabe:=true;
                repeat
                  Str:='1';
                  Eingabe:=Inputquery('Eingabe Schranke Knoten: '+Kno.Wert,
                  'Eingabe Schranke:
                  ',Str);
                until Str<>'';
                if (not StringistRealZahl(Str)) or (StringistRealZahl(Str) and
                (Abs(StringtoReal(Str))<1.0E30))
                then
                  Ka.Schranke:=Abs(StringtoReal(Str))
                else
                begin
                  ShowMessage('Fehler! Eingabe nicht im zulässigen numerischen
                  Bereich!');
                  Eingabe:=false;
                end;
                {Ka.Schranke:=Abs(StringtoReal(str));}
                FlussQuellenzaehler:=FlussQuellenzaehler+Ka.Schranke;
                if not Eingabe then
                begin
                  ShowMessage('Wiederholung der Eingabe der Quellen-Schranken!');
                  FlussQuellenzaehler:=0;
                  goto Marke1;<<
                end;
              end;
          'o':begin
                Ka.Schranke:=1;                                               25)
                FlussQuellenzaehler:=Flussquellenzaehler+1;
              end;
          'b':begin
                Ka.Schranke:=abs(Kno.Grad(true));                             26)
                Flussquellenzaehler:=FlussQuellenzaehler+abs(Kno.Grad(true));
              end;
        end;
        FuegeKanteein(Quelle,Kno,true,Ka);                                    27)
      end;
    if (((Art='t') or(Art='o')) and Kno.AusgehendeKantenliste.Leer)           28)
       or ((Art='b') and (Kno.Grad(true)>0))
    then
    begin
      Ka:=TMinimaleKostenkante.Create;
      Ka.Kosten:=0;
      Ka.Weite:=0;
      Ka.Wert:='U';
      Ka.Richtung:=false;
      case Art of
        't':begin
          Marke2:
```

```
        Eingabe:=true;
        repeat
          Str:='1';
          Eingabe:=Inputquery('Eingabe Schranke Knoten: '+Kno.Wert,'Eingabe Schranke:
          ',Str);
        until Str<>'';
        if (not StringistRealZahl(Str)) or (StringistRealZahl(Str) and
        (Abs(StringtoReal(Str))<1.0E30))
        then
          Ka.Schranke:=Abs(StringtoReal(Str))
        else
        begin
          ShowMessage('Fehler! Eingabe nicht im zulässigen numerischen Bereich!');
          Eingabe:=false;
        end;
        FlussSenkenzaehler:=FlussSenkenzaehler+Ka.Schranke;
        if not Eingabe then
        begin
          ShowMessage('Wiederholung der Eingabe der Senken-Schranken!');
          FlussSenkenzaehler:=0;
          goto Marke2;
        end;
      end;
      'o': begin
             Ka.Schranke:=1;
             Flusssenkenzaehler:=Flusssenkenzaehler+1;
           end;
      'b': begin
             Ka.Schranke:=abs(Kno.Grad(true));
             Flusssenkenzaehler:=Flusssenkenzaehler+abs(Kno.Grad(true));
           end;
    end;
    FuegeKanteein(Kno,Senke,true,Ka);
  end;
  NaechsteKante:
 end;
 Fluss:=FlussQuellenzaehler;
 if FlussQuellenzaehler<>Flusssenkenzaehler then                          (29)
 begin
   Flussok:=false;
   ShowMessage('Fluss durch Quelle ist ungleich Fluss durch Senke');
 end;
 ErzeugeQuellenundSenkenknotensowieKanten:=FlussOk;
end;
```

Bei 20) werden zunächst der Qellen-und der Senkenknoten zusätzlich erzeugt, und bei 21) werden zwei Zähler,die den Fluss durch diese Knoten zählen sollen,mit Null initiiert.

Bei 22) wird dann für jeden Knoten,dessen eingehende Kantenliste leer ist (Quelle) bzw. beim Briefträgerproblem (23) für jeden Knoten,dessen gerichteter Knotengrad kleiner als Null ist,Kanten erzeugt,deren Schrankenwerte beim Transportproblem bei 24) eingegeben werden können,und beim Optimalen Matching bei 25) zu Eins gesetzt werden,sowie beim Briefträgerproblem gleich dem (absoluten) gerichteten Knotengrad gesetzt werden (26).Die Kanten werden anschließend in den Graph eingefügt (27).Der Flussquellenzähler wird jeweils um den entsprechenden Betrag erhöht.
Ab 28) werden alle Schritte des letzten Absatzes noch einmal in gleicher Art für jeden Knoten,dessen ausgehende Kantenliste leer ist (Senke) bzw. beim Briefträgerproblem für jeden Knoten,dessen gerichteter Knotengrad kleiner als Null ist,wiederholt.Hier wird der Flusssenkenzähler entsprechend erhöht.

Zum Schluß wird überprüft,inwieweit die beiden Zähler übereinstimmen (29).Bei Nichtübereinstimmung wird eine Meldung ausgegeben,und der Rückgabewert der Function ist false (sonst true).

C XIV Das Chinesische Briefträgerproblem

Als Erweiterung einer Unterrichtsreihe zum Minimalen Kostenproblems bzw. Hitchcockproblems bietet sich die Lösung des Chinesischen Briefträgerproblems an.Gleichzeitig bildet dieses Problem auch die Fortsetzung einer Unterrichtsreihe zum Thema Pfade und Linien,speziell den geschlossenen Eulerlinien.

Wenn nämlich ein Graph eine geschlossene Eulerlinie besitzt,dann ist dieser Pfad gleichzeitig schon die Lösung des Chinesischen Briefträgerproblems,bei dem es darum geht,eine Wegstreckenminimale Lösung zum Besuch aller Kanten eines Graphen,dessen Kanten mit ihrem Kanteninhalt als Zahl (Feld Inhalt_ als Teilwegstrecke aufgefaßt) bewertet werden,so zu suchen,so dass eine geschlossene Linie erreicht werden kann.
Wenn es keine geschlossene Eulerlinie gibt,müssen einige Kanten mehrfach durchlaufen werden (bzw. in einer geschlossenen Linie mehrfach vorhanden sein),und die Auswahl dieser Kanten sollte Wegstreckenminimal sein.

Wählt man nun die durch Zahlen (Wegstrecken) vorgegebene Kantenbewertung als Kosten,läßt sich mit geeigneten Quellen-und Senkenknoten (Knoten mit positivem und negativem Knotengrad) ein (ganzzahliges) Hitchcockproblem lösen,dessen Fluss die Anzahl der Zusatzkanten im vorgegebenen Graphen bestimmt.Ergänzt man den Graph um diese Zusatzkanten,existiert auf diesem erweiterten Graph dann eine geschlossene Eulerlinie,der sich mit Hilfe des in Kapitel C V beschrieben Algorithmus (Eulerlinie) bestimmen läßt

Damit läßt sich die Aufgabe alleine mit Hilfe der Algorithmen der Menüs Anwendungen/Transportproblem (die sich wiederum auf des Minimale Kostenproblem stützen) und Anwendung/Eulerlinie des Programms Knotengraph lösen.

Der Algorithmus des Menüs Anwendungen/Chinesischer Briefträger kombiniert beide Algorithmen und läßt sie nacheinander ablaufen.Eine Lösung des Chinesischen Briefträgerproblems ergibt sich so allerdings nur für **gerichtete** Graphen,weil das Kostenproblem bzw. das Hitchcockproblem einen gerichteten Graphen voraussetzen.

Insgesamt kann also die Behandlung des Chinesischen Briefträgerproblems eine interessante anwendungsorientierte Erweiterung und Abrundung der Unterrichtsweihe der Kapitel C V und/oder C XIII sein,bei der die wesentlichen Eigenschaften der Algorithmen nochmals unter neuem Aspekt vertieft wiederholt werden können.

Aufgabe C XIV.1:(Einstiegsproblem)

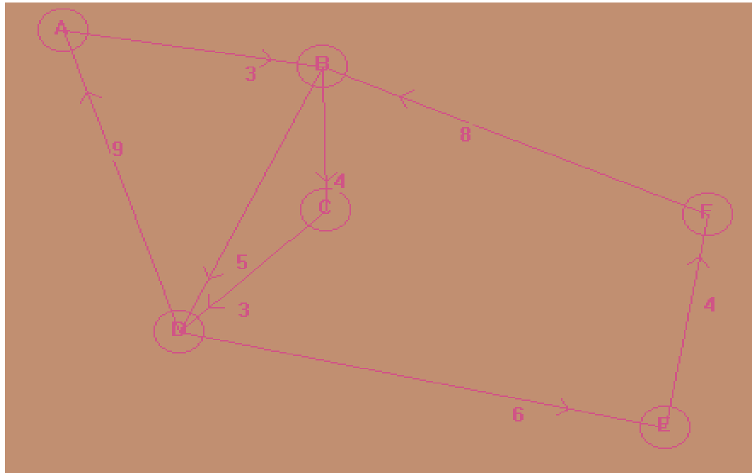
Ein Briefträger (Zusteller mit Auto) soll die Briefe an die Haushalte A,B,C,D,E und F verteilen.Der folgende Graph gibt an,wie die Haushalte durch Straßen miteinander verbunden sind.Die Knoten des Graphen sind die Haushalte,die gerichteten Kanten sind die Verbindungsstraßen.Sie dürfen nur in Pfeilrichtung durchfahren werden (Einbahnstraßen).

(Der nachfolgend beschriebene Algorithmus funktioniert,wie oben erwähnt, nur bei gerichteten Graphen,weil er auf das Minimalkostenproblem zurückgeführt wird."Nicht-Einbahnstraßen" müssen durch zwei antiparallele Pfeile dargestellt werden.Allerdings hat dies den Nachteil,dass diese Straßen als zwei getrennte Straßen angesehen werden.Zur Lösung des Briefträgerproblems auf ungerichteten Graphen siehe z.B.:Lit 45,S.173)

Den Kanten wird eine nichtnegative Bewertung als Kanteninhalt (Feld Inhalt_) zugeordnet,die die Straßenlänge (z.B. in km) darstellen soll.

Der Briefträger möchte die Briefe auf einem Rundweg austragen,der wieder zum Ausgangspunkt zurückführt,wobei eine möglichst geringe Strecke zurückgelegt werden soll.

Graph:



G095.gra

Lösung:

Die minimale Stecke eines Rundweges liegt dann vor,wenn eine Eulerlinie durchlaufen werden kann,bei dem jede Kante genau einmal benutzt wird.

(Zum Thema Eulerlinien siehe Kapitel V)

Die Voraussetzung für die Existenz einer geschlossenen Eulerlinie ist gegeben,da jeder Knoten des gerichteten Graphen genau soviele einlaufende wie ausgehende Kanten besitzt,d.h. der (gerichtete) Knotengrad jedes Knoten ist Null.

Benutze den Algorithmus des Menüs Anwendungen/Eulerlinie des Programms Knotengraph und ermittle eine Lösung.

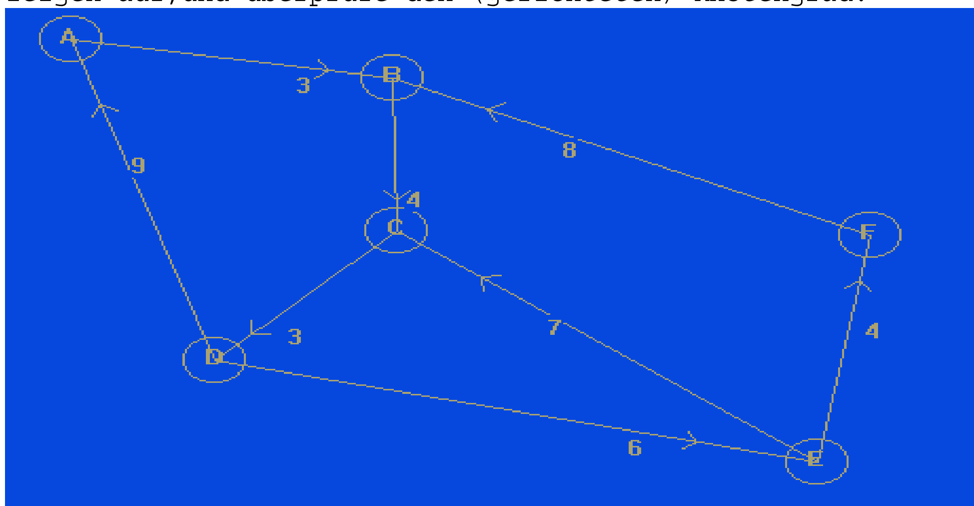
Lösung:

B C D E F B D A B Summe: 42 Produkt: 311040

Also ist 42 die minimale Strecke des Rundwegs des Briefträgers.

Zusatz:Änderung des Graphen:

Der Graph wird nun so abgeändert,dass die Kante B->D gelöscht wird und zusätzlich eine Kante E->C mit der Bewertung 7 eingefügt wird.
Überprüfe die Existenz einer Eulerlinie mit dem Algorithmus des Menüs Eigenschaften/Eulerlinie.Benutze danach das Menü Eigenschaften/Knoten anzeigen auf,und überprüfe den (gerichteten) Knotengrad.



G096.gra

A Koordinaten: x=241 y=91 Knotengrad (gerichtet/ungerichtet): 0/2 Typ:
B Koordinaten: x=393 y=116 Knotengrad (gerichtet/ungerichtet): -1/3 Typ:
C Koordinaten: x=395 y=215 Knotengrad (gerichtet/ungerichtet): -1/3 Typ:
D Koordinaten: x=309 y=299 Knotengrad (gerichtet/ungerichtet): 1/3 Typ:
E Koordinaten: x=594 y=365 Knotengrad (gerichtet/ungerichtet): 1/3 Typ:
F Koordinaten: x=619 y=218 Knotengrad (gerichtet/ungerichtet): 0/2 Typ:

Ergebnis:

Der Graph besitzt jetzt keine Eulerlinie mehr.Der Grund liegt darin,dass die Knoten B und C den Grad -1 und die Knoten D und E den Grad 1 haben.Lediglich die Knoten A und F haben den Grad 0.

Erläuterung des Briefträgerproblems:

Also ist es jetzt nicht mehr möglich,alle Kanten des Graphen so auf einem Rundweg zu durchlaufen,dass jede Kante nur genau einmal besucht wird.

Der Briefträger muß deshalb einige Kanten (Straßen) mehrfach benutzen.Unter diesen Straßen möchte er eine Auswahl treffen,dass die Gesamtstrecke unter allen Möglichkeiten der Rundwege minimal ist.

Dieses Problem heißt Problem des Chinesischen Briefträgers.Es wird im Falle der Existenz einer Eulerlinie durch die Eulerlinie gelöst.

Wenn keine Eulerlinie existiert,kann das Problem auf ein Minimales Kostenproblem,das bestimmt,welche Kanten (Straßen) zusätzlich durchlaufen werden sollen,zurückgeführt werden.

Der Algorithmus Chinesisches Briefträgerproblem

Ein Rundweg des Briefträgers auf dem Graphen,der mindestens jede Kante einmal besucht,kann als (ganzzahliger) Fluss aufgefaßt werden,wobei jeder Kante die Anzahl der Durchläufe dieser Kante als ganze Zahl zugeordnet wird.Für jeden Knoten ist dann nämlich die Anzahl der Läufe,bei denen „in den Knoten hineingegangen“ (der Zustellungsort angelaufen) wird,gleich der Anzahl,bei denen sich der Briefträger vom Knoten (vom Zustellungsort) entfernt.Dieses ist die Flussbedingung für jeden Knoten.

Umgekehrt läßt sich jeder (ganzzahlige) Fluss auf dem Graphen als Rundweg des Briefträgers deuten,wobei der (ganzzahlige) Flusswert in jeder Kante die Anzahl angibt,die diese Kante (Straße) durchlaufen wird.

Besser ist es gleich diesen Fluss um 1 zu vermindern.Dadurch entsteht auch ein Fluss nämlich der Zusatzfluss f,der angibt,wieviel mal eine Kante zusätzlich durchlaufen werden muß.

Wenn jetzt ein Rundweg mit minimaler Streckenlänge gesucht wird,ist also ein minimaler Fluss gesucht,der bezüglich der Länge als Kantenbewertung,die als Kosten der Kante aufgefaßt wird,Kostenminimal ist.

Bei den Knoten B und C des obigen Graphen fließt zuviel Fluss in die Knoten hinein,weil zwei Kanten auf sie zu führen und nur eine Kante hinausläuft.Also sind diese Knoten zusätzlich zu Quellen des Flusses mit dem Quellenfluss 1 zu machen,damit noch zusätzlich die Flusseinheit 1 (Zusatzfluss) aus ihnen herausfließen kann.Eine Einheit Fluss muß noch durch das Erzwingen des Erzeugens einer zusätzlichen auslaufenden Kante aus dem Knoten herausfließen,damit die Flussbilanz stimmt.

Ebenso sind die Knoten D und E zusätzlich zu Senken des Flusses mit dem Fluss -1 zu machen,weil zwei Kanten von ihnen wegführen und nur eine Kante hineinläuft,damit eine Einheit Fluss noch zusätzlich durch Erzwingen des Erzeugens einer zusätzlichen einlaufenden Kante in die Knoten hineinfließen kann und damit die Flussbilanz stimmt.

Wenn man nun diese Knoten als Quellen und Senken auffaßt,und auf diesem Graphen ein minimales Hitchcockproblem mit den Streckenlängen als Kosten löst,entsteht ein Kostenminimaler Fluss,dessen Werte angeben,wie oft eine Kante vom Briefträger durchlaufen werden sollte.

Um die mehrfach durchlaufenden Kanten zu veranschaulichen,werden dann f viele Parallelkanten zu ihnen zusätzlich erzeugt,wobei f der ganzzahlige Fluss durch die Kante ist.

Der so entstandene Graph muß dann eine geschl. Eulerlinie enthalten,weil in jeden Knoten gemäß der Flussbedingung soviele einlaufende Kanten hineingehen,wie er auslaufende Kanten hat.Also ist der gerichtete Knotengrad jedes Knoten gleich Null.

Auf diesem Graph kann dann also eine Eulerlinie gesucht werden,dessen Summe aller Kantenbewertungen (Streckenlänge) wegen der Lösung des minimalen Kostenproblems minimal unter allen Möglichkeiten ist.

Verbale Beschreibung:

Verallgemeinerte Beschreibung des Algorithmus:

1)Bestimme im gerichteten Graphen die Knoten mit positiven und negativem (gerichtetem) Knotengrad.Lege die Knoten mit negativem Knotengrad als Quellen und die Knoten mit positivem Knotengrad als Senken jeweils mit einem ganzzahligen Fluss fest,der gleich dem Knotengrad ist.

2)Löse auf dem so entstandenen Graph das minimale Hitchcockproblem mit ganzzahligem Fluss und der Streckenlänge jeder Kante als Kostenbewertung.

3)Erzeuge zu jeder Kante f Parallelkanten mit derselben Steckenlängen-Bewertung,wobei f der Fluss durch diese Kante ist.

4)Bestimme auf dem so entstandenen Graphen eine Eulerlinie.Diese Eulerlinie ist die Lösung des Chinesischen Briefträgerproblems.

Die Summe der Kantenbewertungen (Gesamtstreckenlänge) ist der Gesamtweg, den der Briefträger zurücklegen muß.

Die minimalen Kosten der Lösung des Hitchcockproblems ist die Summe der zusätzlichen Kantenstrecken,die mehrfach durchlaufen werden (der sogenannte unproduktive Weg).

Die Differenz beider Summen ist die Gesamtsumme aller Kantenbewertungen des Graphen (die Summe der Strecken des nur einmal durchlaufen Straßennetzes).

Fortsetzung der Aufgabe C XIV.1:

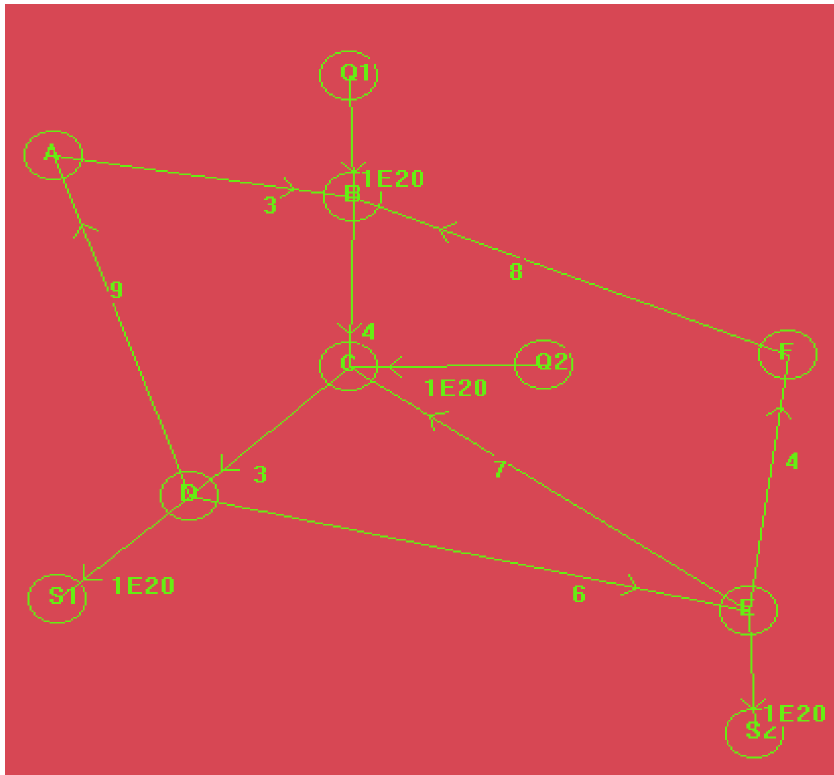
Beobachtung und problemorientierte Erarbeitung:

a)Löse die oben gestellte Aufgabe mittels des Algorithmus des Menüs Anwendungen/Transportprobleme als minimales Hitchcockproblem mit der Option ganzzahliger Fluss mit den Streckenlängen der Kanten als Kosten,indem zu den Knoten B und C je ein zusätzlicher Quellknoten Q_1 und Q_2 erzeugt wird,mit Kanten die nach C und D führen und die eine unendlich große Schranke (1E20) haben.Den Knoten wird der Quellenfluß 1 zugeordet.Erzeuge ferner zwei Senkenknoten S_1 und S_2,zu denen jeweils eine Kante von den Knoten D und E aus mit der Schranke unendlich (1E20) führt.Diesen Knoten wird als Senkenfluss der Fluss 1 ($=|-1|$) zugeordnet.Ermittle dann zum so veränderten Graphen den Fluss bezüglich des minimalen Hitchcockproblems und ermittle damit die mehrfach zu durchlaufenden Kanten.

b)Löse die Aufgabe mittels des Algorithmus des Menüs Anwendungen Chinesischer Briefträger im Demomodus.Wähle dabei die Option (nur) **einen** Eulerkreis suchen.Verfolge und erkläre den Ablauf des Algorithmus.

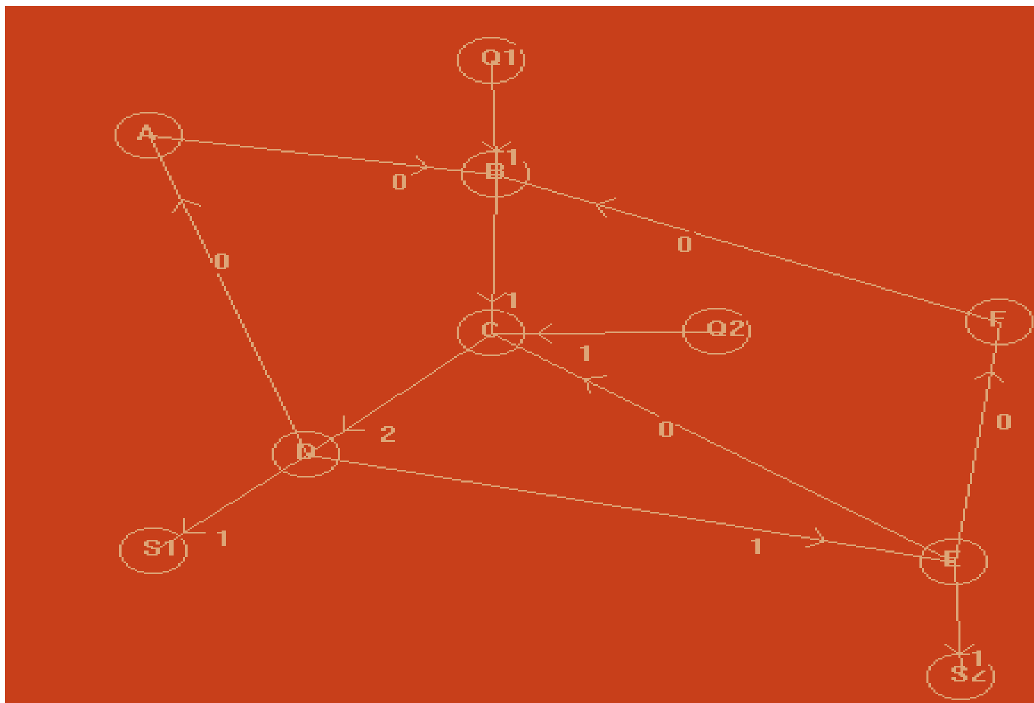
Lösungen:

a)Graph mit zusätzlichen Quellen-und Senkenknoten:



G097.gra

Graph mit minimalen Kostenfluß:



G097.gra

Kante: A B Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 3
Kante: B C Fluss: 1 Flusskosten: 4 Schranke: 9E31 Kosten: 4

Kante: C D Fluss: 2 Flusskosten: 6 Schranke: 9E31 Kosten: 3
Kante: D E Fluss: 1 Flusskosten: 6 Schranke: 9E31 Kosten: 6
Kante: D A Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 9
Kante: D S1 Fluss: 1 Flusskosten: 0 Schranke: 9E31 Kosten: 0
Kante: E F Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 4
Kante: E C Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 7
Kante: E S2 Fluss: 1 Flusskosten: 0 Schranke: 9E31 Kosten: 0
Kante: F B Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 8
Kante: Q1 B Fluss: 1 Flusskosten: 0 Schranke: 9E31 Kosten: 0
Kante: Q2 C Fluss: 1 Flusskosten: 0 Schranke: 9E31 Kosten: 0

Vorgegebener Fluss durch Quelle oder Senke: 2

Erreichter Fluss durch Quelle oder Senke: 2

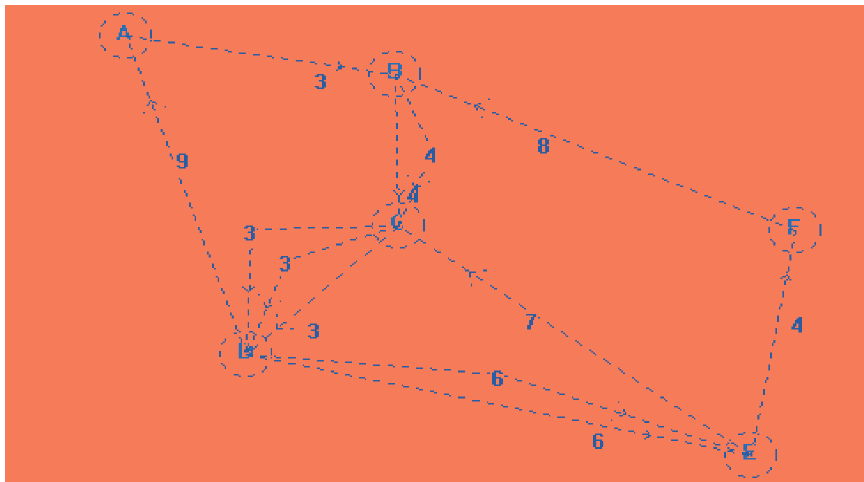
Gesamtkosten: 16 (minimal)

Also müssen die Kanten BC und DE einmal,die Kante CD zweimal zusätzlich durchlaufen werden.Dadurch entsteht eine zusätzliche Wegstrecke von 16 (km).

Beobachtung und Deutung:

Wie man sieht,verläuft die aus dem Quellknoten Q_2 bzw. C austretende Flusseinheit über Knoten D direkt in den Senkenknoten E bzw. S_2,während die von Q_1 bzw. B zusätzlich erzeugte Flusseinheit über Knoten C in den Knoten D bzw. S_1 läuft.

b)Graph mit Zusatzkanten und Eulerlinie:



G098.gra

A B C D E F B C D E C D A Summe: 60 Produkt: 94058496
(Eulerlinie mit Pfadlänge 60)

Kante: A B Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 3
Kante: B C Fluss: 1 Flusskosten: 4 Schranke: 9E31 Kosten: 4
Kante: C D Fluss: 2 Flusskosten: 6 Schranke: 9E31 Kosten: 3
Kante: D E Fluss: 1 Flusskosten: 6 Schranke: 9E31 Kosten: 6
Kante: D A Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 9
Kante: E F Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 4
Kante: E C Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 7
Kante: F B Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 8

Vorgegebener Fluss durch Quelle oder Senke: 2

Erreichter Fluss durch Quelle oder Senke: 2

Gesamtkosten: 16 (Unproduktive Weglänge durch die Zusatzstrecken/Kanten)

Der Quellcode im Auszug:

Das Erzeugen der zusätzlichen Quell-und Senkenknoten sowie der zugehörigen Kanten mit Schranken wurde schon im vorigen Kapitel bei der Erläuterung der Methode ErzeugeQuellenundSenkensowiekanten beschrieben.Der Algorithmus benötigt dann lediglich noch eine Methode,die die Kantenwerte als Kosten speichert:

```
procedure TMinimalekostengraph.SpeichereSchrankenalsKosten;
var Index:Integer;
    Ka:TMinimaleKostenkante;
begin
  if not self.Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
    begin
      Ka:=TMinimaleKostenkante(Self.Kantenliste.Kante(Index));
      Ka.Kosten:=StringtoReal(Ka.Wert);
    end;
end;
```

Außerdem müssen die Zusatzkanten gemäß dem Fluss erzeugt werden:

```
procedure TMinimaleKostengraph.ErzeugeunproduktiveKanten;
var Index:Integer;
    Ka,Kup:TMinimalekostenkante;

  procedure BildeunproduktiveKanten(Ka:TMinimaleKostenkante);
  var Zaehl:Integer;
  begin
    Kantenwertposition:=0;
    for Zaehl:=1 to Round(Ka.Fluss) do
    begin
      Kup:=TMinimaleKostenKante.Create;
      Kup.Wert:=Ka.Wert;
      Kup.Typ:='r';
      Kup.gerichtet:=true;
      Kup.Weite:=Ka.Weite+Zaehl*20;
      FuegeKanteein(TInhaltsknoten(Ka.Anfangsknoten),TInhaltsknoten(Ka.Endknoten),true,Kup);
    end;
  end;

begin
  if not self.Kantenliste.Leer then
    for Index:=0 to self.Kantenliste.Anzahl-1 do
    begin
      Ka:=TMinimaleKostenkante(self.Kantenliste.Kante(Index));
      Ka.Wert:=Realtostring(Ka.Kosten);
      if (not Ka.KanteistSchlinge) and (Ka.Fluss>=1) then
        BildeunproduktiveKanten(Ka);
    end;
end;
```

Schließlich wird die im Kapitel C V beschriebene Methode Eulerlinie auf eine Kopie (Inhaltskopie) des Graphen ausgehend vom zuletzt mit der Maus angeklickten Knoten als Startknoten (oder dem zuerst in den Graphen eingefügten Knoten) angewendet:

```
M:=TEulergraph(F.InhaltskopiedesGraphen(TEulergraph,TInhaltsknoten,TInhaltskante,false));
M.Eulerlinie(Paintbox.Canvas,Ausgabe1,Esliste,GraphH.LetzterMausklickknoten,GraphH.LetzterMaus
klickknoten);
```

(siehe Menümethode TKnotenform.ChinesischerBrieftraegerClick)

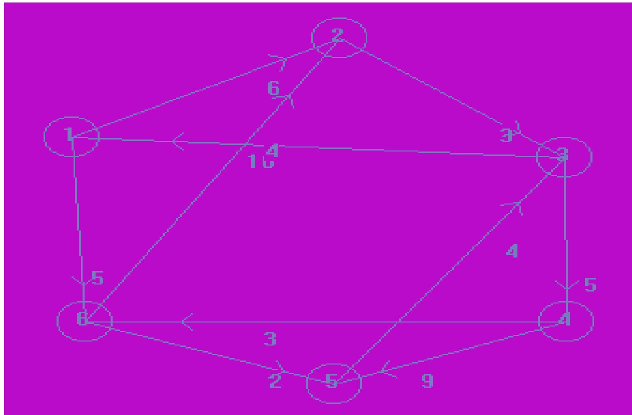
Aufgabe C XIV.2:

(Aufgabe aus Lit 45, S.171)

Löse das Chinesische Briefträgerproblem auf dem folgenden Graphen mit Hilfe des Algorithmus des Programms Knotengraph,Menü Anwendungen/Chinesischer

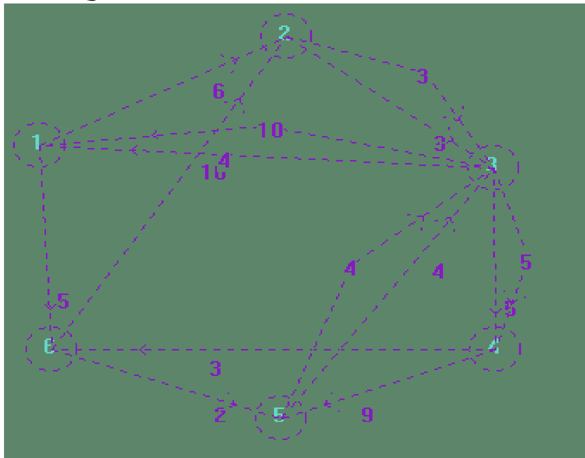
Briefträger (oder durch den eigenen mit der Entwicklungsumgebung erzeugten Quelltextes je nach Konzeption).

Graph:



G099.gra

Lösung:



G100.gra

1 2 3 4 5 3 1 6 5 3 4 6 2 3 1 Summe: 73 Produkt: 2332800000
(Eulerlinie Gesamtstrecke: 73)

Kante: 1 2 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 6
Kante: 1 6 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 5
Kante: 2 3 Fluss: 1 Flusskosten: 3 Schranke: 9E31 Kosten: 3
Kante: 3 4 Fluss: 1 Flusskosten: 5 Schranke: 9E31 Kosten: 5
Kante: 3 1 Fluss: 1 Flusskosten: 10 Schranke: 9E31 Kosten: 10
Kante: 4 5 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 9
Kante: 4 6 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 3
Kante: 5 3 Fluss: 1 Flusskosten: 4 Schranke: 9E31 Kosten: 4
Kante: 6 5 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 2
Kante: 6 2 Fluss: 0 Flusskosten: 0 Schranke: 9E31 Kosten: 4

Vorgegebener Fluss durch Quelle oder Senke: 2

Erreichter Fluss durch Quelle oder Senke: 2

Gesamtkosten: 22 (Unproduktive Weglänge der Zusatzkanten/ Strecken)

D Zusammenfassung

Mathematik erhält ihre Bedeutung aus der Fähigkeit Objekte der realen Umwelt und des Denkens sowie ihre gegenseitige Beziehungen als Symbole abstrakt darzustellen und mit diesen Symbolen nach festgelegten Regeln zu operieren.

Ist die Anzahl dieser Objekte endlich (diskrete Mathematik),ist die Veranschaulichung dieser Symbole in der Form eines Graphen sehr oft die geeignete Darstellungsart,wobei die Objekte als Knoten und ihre gegenseitigen Beziehungen als Kanten realisiert werden.

Wahrscheinlichkeitsrechnung und Geometrie sind bisher die Teilgebiete im Mathematikunterricht der Schule,die als Unterrichtsinhalte die Beziehung zwischen **endlichen** Objektmengen thematisieren.Bei der Wahrscheinlichkeitsrechnung sind die Objekte die Ereignisse in einem (endlichen) Wahrscheinlichkeitsraum und bei der Geometrie die geometrischen Figuren der Anschauung.Im Gegensatz dazu spielen bei der Infinitesimalrechnung,dem dritten Teilgebiet der Schulmathematik fast immer unendliche Objektmengen eine Rolle.

Während sich die Wahrscheinlichkeitsrechnung in großem Umfang der Darstellung ihrer Objekte durch Graphen bedient (Wahrscheinlichkeitsbäume und Markovketten) spielen Graphen in der Schulgeometrie kaum eine Rolle.Dies liegt an der Behandlung dieser Objekte als Dinge der Anschauung,wodurch auch ihre gegenseitigen Beziehungen (Parallelität,Orthogonaltät,Schnitte usw.) anschaulich dargestellt werden können.

Abstrahiert man jedoch und betrachtet die Relationen zwischen geometrischen Objekten nur noch als Regeln eines Axiomensytems oder als Invarianten unter bestimmten Abbildungen (Kongruenzabbildungen,Ähnlichkeitsabbildungen,topologischen Abbildungen),gewinnt auch die Darstellung als Graph wieder mehr an Bedeutung,wie dies z.B. beim verwandten Gebiet der Topologie (z.B. bei der Darstellung von topologischen Netzen) der Fall ist.

Einerseits ist also ein Graph das Endergebnis einer komplexen Abstraktion des Denkens (Modellbildung).Andererseits ergibt sich die Darstellung eines Problems als Graph bei vielen Problemen fast wie von selbst und bietet sich nahezu sofort an,wie z.B. bei Zuordnungsproblemen,bei der Darstellung eines Stadt-/Strassennetzes oder von Rohrleitungssystemen mit den dazugehörigen Aufgabenstellung des Suchen von Pfaden jedweder Art,speziell beim Euler-und Hamiltonproblem,bei Flussproblemen oder aber bei der zeichnerischen Darstellung eines maximalen Matching als Graph (z.B. welcher Lehrer welchem Unterrichtsfach zugeordnet werden sollte).

Diese Probleme und viele Aufgabenstellungen aus dem Bereich des Operations Research wurden bisher im Schulunterricht vernachlässigt bzw. überhaupt nicht behandelt,weil sie keinem der oben genannten Gebiete der Schulmathematik zugeordnet sind,obwohl ihre Bedeutung immer größer wird.Sie erfordern zu ihrer Lösung das Ausführen eines Algorithmus,der auf einem Graphen operiert,und sind durch den Einsatz von Computern einer schnellen Lösung zugänglich,indem der Algorithmus in eine für Computer auszuführende Form mittels einer Programmiersprache übersetzt wird.

Daraus resultiert die Notwendigkeit der Benutzung von Methoden der Informatik zur Lösung von Graphenproblemen,wobei sich die Informatik selber schon zur Darstellung von Datenstrukturen der Graphentheorie bedient.

Die Beschäftigung mit Problemen der Graphentheorie vereinigt also in sich das Erkennen von Relationen zwischen Objekten,die Darstellung dieser Objekte in Form eines Graphen,das Entwerfen von geeigneten Algorithmen,die auf dieser Graphenstruktur operieren und die Codierung in einer geeigneten Programmiersprache.Hierbei erweist es sich als vorteilhaft,um die Objekt-Struktur des Graphen nachbilden zu können,eine objektorientierte Programmiersprache zu benutzen,um die Beziehungen der Objekte untereinander möglichst 1 zu 1 modellieren zu können.Also sollten bei der Lösung von Proble-

men der algorithmischen Graphentheorie die Gebiete Mathematik und Informatik zusammenwirken.

Die vorliegende Arbeit hat gezeigt,wie Probleme der algorithmischen Graphentheorie im Unterricht mit Hilfe des (fertigen) Programms Knotengraph (DWK) bzw. der vorgestellten Entwicklungsumgebung Knotengraph (EWK) und der textorientierten Knotengraphversion (CAK) behandelt werden können,wobei vorteilhaft objektorientierte Datenstrukturen benutzt wurden,so dass einerseits die Graphenstruktur in der Programmierungsumgebung weitgehend nachgebildet werden kann,andererseits unnötiger Programmierungsaufwand durch das Prinzip der Vererbung,das sich sogar auf ganze Benutzeroberflächen erstrekken kann (visuelle Vererbung),vermieden wird.

Kapitel C erörterte an Hand von konkreten Unterrichtsskizzen und Beispielen von Graphen,welche neuen Inhalte methodisch-didaktisch in welcher Weise im Unterricht behandelt werden können.Dabei besteht die Möglichkeit,Knotengraph sowohl als fertiges Programm als auch als Programmierungs-Entwicklungsumgebung unter Einsatz objektorientierter Datenstrukturen zu verwenden.

Die dazu benötigten Datenstrukturen als auch die Bedienung des Programms Knotengraph werden im Anhang F erörtert.Erst durch den Einsatz von objektorientierten Strukturen ist es möglich,im Unterricht durch Bereitstellung von geeigneten Softwaretools in Form von Objekten und visuellen Komponenten die komplexe Darstellung der Veränderung eines Graphen infolge des Ablaufs eines Algorithmus darzustellen,sowie eine geeignete Graphenstruktur bereitzustellen,so dass die Programmierungstätigkeit von Schülern schon auf einer höheren Ebene beginnen und sich so ganz auf die Erstellung des eigentlichen Graphenalgorithmus beschränken kann,ohne sich in aufwendigen Nebentätigkeiten verlieren zu müssen.

In Kapitel B wurde begründet,dass ein Unterrichtsgang zum Thema Graphentheorie in gleicher Weise wie andere Inhalte des Mathematik- oder Informatikunterichts geeignet ist,um sowohl die spezifischen Lernziele der Fächer als auch die allgemeinen Bildungsziele zu erfüllen,wobei in einer solchen Unterrichtsreihe alle bekannten, bewährten Unterrichtsmethoden eingesetzt werden können.Darüberhinaus wurde gezeigt,dass die Graphentheorie durch ihre besonderen Eigenschaften geeignet ist,spezielle Lernziele in besonders guter Weise zu erreichen.

Zu diesen Eigenschaften gehört die Anschaulichkeit,die es ermöglicht dass die Problem-und Aufgabenstellungen der Graphentheorie sehr oft ohne spezielles mathematisches Vorwissen sofort evident sind und sogar Personen ohne mathematische Vorbildung sofort plausibel gemacht werden können.Aber auch der Lösungsgang kann sehr oft anschaulich dargestellt werden,so dass viele Lösungen intuitiv durch entdeckendes Lernen von Schülern selbstständig gefunden werden können und auf diese Weise das Mathematisieren erlernt werden kann.

Die Lösungsverfahren operieren immer auf den Grundelementen Knoten und Kanten,indem sie entweder nach geeigneten Pfaden suchen,oder aber die den Kanten oder Knoten zugeordneten Inhalte (Datenfelder) in systematischer Weise verändern sowie in selteneren Fällen auch die Graphenstruktur durch Hinzufügen oder Löschen von Knoten und Kanten umgestalten (oder mehrerer dieser Operationen gleichzeitig).Bei der Erstellung entsprechender Algorithmen spielt die gegenseitige Beziehung der Knoten und Kanten als verknüpfte Objekte im Graphennetz die entscheidende Rolle.

Diese Beziehungen sind,mathematisch betrachtet mehrstellige Relationen,so dass Graphentheorie hier einen wichtigen Beitrag zur abstrakten Strukturmathematik,nämlich der Eigenschaften von Relationen leistet.Betrachtet man solche Relationen auf bipartiten Graphen und stellt sich die Aufgabe,ein geeignetes Matching zu suchen,führt dies direkt zum wichtigen Begriff der Funktion.

Die Art der Lösungsverfahren als Operationen auf den Objekten Knoten und Knoten,die durch sich selber durch gegenseitige Relationen verknüpft

sind,legt die Anwendung von objektorientierten Programmierungstechniken nahe,so dass diese wichtige Art der Programmerstellung an Hand der Graphentheorie als geeignetes Beispiel in allen seinen Möglichkeiten eingeübt werden kann,ohne künstlich Aufgabenstellungen oder unrealistische Beispiele erfinden zu müssen.

Die Aufgaben der Graphentheorie entstehen fast immer durch Abstraktionen der Wirklichkeit,so dass hier das wichtige Prinzip des Abstrahierens und des Modellierens von der Realität in die Form eines Graphen verlangt wird.Dabei können Problemstellungen aus verschiedenen Bereichen als Endergebnis den gleichen Graphentyp und zu ihrer Lösung denselben Graphenalgorithmus erfordern.Dies zeigt,dass sich die Bedeutung der Graphentheorie nicht nur auf ein Gebiet - etwa nur im innermathematischen Bereich - erstreckt,sondern dass sie auch Fächerübergreifend von Wichtigkeit ist.

Daraus ergibt sich wiederum der Ansatz des projektorientierten Arbeitens,dessen Merkmal es gerade ist,sich nicht nur auf ein Fach zu beschränken.Eine andere Begründung für diesen Ansatz ergibt sich durch die Notwendigkeit der Aufteilung und Bearbeitung umfangreicherer Graphenalgorithmen auf und durch verschiedene Programmiergruppen (Schülergruppen),die auf Grund der vorhandenen Komplexität das Gesamtprogramm zunächst als Einzelmodule erstellen und anschließend daraus zusammensetzen müssen.

Insgesamt erweisen sich,wie die vorliegende Arbeit zeigt,Unterrichtsinhalte der algorithmischen Graphentheorie.

a)im Mathematikunterricht als besonders geeignet zur exemplarischen Behandlung von Verfahren der diskreten Mathematik und des Operations Research,wobei durch diese neuen Unterrichtsthemen dieselben allgemeinen mathematischen Lernziele wie mit den bisher konventionell behandelten Stoffgebieten erreicht werden können,aber andererseits diese wichtigen neuen Inhalte und Verfahrensweisen endlich im Mathematikunterricht berücksichtigt werden können.

b)im Informatikunterricht als nichttriviale,anwendungsbezogene Beispiele, um Verfahren der objektorientierten Programmierung tiefergehend darstellen und vermitteln zu können.

c)als geeignete den Mathematik-und Informatikunterricht verbindende Themen,so dass Schüler an Hand von anwendungsbezogenen Aufgabenstellungen neben der formal statischen Vorgehensweise der Mathematik die algorithmisch dynamische Seite der Informatik kennenlernen.

d)als geeignete Beispiele zur Abstraktion und Modellbildung,wobei die realen Ausgangssituationen den Problemstellungen verschiedener Fächern entnommen werden können (fächerübergreifend).

e)als geeignete Themen für einen projektorientierten Unterricht.

f)als Mittel,um den Relations-und den Funktionsbegriff darstellen und einüben zu können.

g)durch ihre Anschaulichkeit als besonders geeignet für selbständiges entdeckendes Lernen und unterstützen dadurch den Prozess des Mathematisierens in besonders guter Weise.

Durch die Benutzung des Programms Knotengraph (DWK) oder durch die Benutzung der objektorientierten Entwicklungsumgebung Knotengraph (EWK) sowie durch die Benutzung der textorientierten Consolenanwendung Knotengraph (CAK) können die eben genannten speziellen und die in Kapitel B genannten allgemeinen Lernziele von drei unterschiedlichen Ansätzen ausgehend realisiert werden,nämlich einmal innerhalb des Faches Mathematik unter Zuhilfenahme des Programms Knotengraph als Demonstrationsprogramm zur Veranschaulichung von Algorithmen,zum zweiten mittels einer die Fächer Mathematik und Informatik verbindenden Unterrichtsreihe,in der Graphenalgorithmen von den Schülern unter Benutzung einer vorgegebenen Datenstruktur und Oberfläche selber programmiert werden und schließlich als Programmierungsprojekt in-

nerhalb des Faches Informatik mit dem Ziel den objektorientierten Aufbau der Datenstruktur eines Graphen von einer objektorientierten Listenstruktur ausgehend kennenzulernen.

So stellen die in dieser Arbeit beschriebenen und als Anlage auf CD bereitgestellten Lern-und Softwaretools ein allgemeines Werkzeug dar,um Probleme der algorithmischen Graphentheorie im Schulunterricht geeignet anzugehen und zu veranschaulichen sowie problemorientiert behandeln und lösen zu können.

E Literaturverzeichnis

1)Amtsblatt Ministerium für Schule und Weiterbildung,Wissenschaft und Forschung NRW,Teil I,Nr. 12,15.12.1998,S.222 ff

2)Baumann Rüdiger,Didaktik der Informatik,Klett-Verlag,Stuttgart,2. Auflage,1996

3)Baumann Rüdiger,Algorithmen wozu,LOG IN,14b (1994)

4)Beisel Ernst-Peter,Mendel Manfred,Optimierungsmethoden des Operations-Research,BD1 Optimierungsmethoden des Operations Research,Vieweg-Verlag,Braunschweig,1987,Bd 2 Optimierung in Graphen,Vieweg-Verlag, Wiesbaden, 1991

5)Bielig-Schultz,Einige erstaunliche Graphenalgorithmen,in LOG-IN 14,1994, Heft 4,S. 38 -42

6)Bieß Günter,Graphentheorie,Teubner-Verlag,3. Auflage,Leipzig, 1988

7)Bodendiek Rainer in Leppig Manfred,Neue Aspekte Zum Mathematikunterricht Der Sekundarstufen,Graphentheorie in den Sekundarstufen,S. 33 ff.,Ferdinand Schöningh Verlag,Paderborn,1976

8)Boehm, Franz,Graphen in der Datenverarbeitung,Verlag Harri Deutsch,Thun und Frankfurt/M,1981

9)Boer H.,Krane R. und Wiggermann H.,Anwendungen der Graphentheorie im Hinblick auf die Konstruktion von Unterrichtseinheiten für den Mathematikunterricht,1. Staatsexamensarbeit,Universität Münster,1977

10)Busacker Robert G,Endliche Graphen und Netzwerke,R. Ouldenbourg Verlag,München und Wien,1968.

11)Claus Heinz Jörg,Einführung in die Didaktik der Mathematik,Wissenschaftliche Buchgesellschaft Darmstadt,1. Auflage 1989

12)Doerfler Willibald in Graf Klaus-D.,Computer in der Schule,S. 122 Computer im Mathematikunterricht-Beispiele aus der Stochastik,B.G. Teubner Verlag,Stuttgart,1985.

13)Doerfler Willibald,Graphentheorie für Informatiker,Sammlung Göschen, Walter de Gruyter Verlag,Berlin New York,1973

14)Domschke Wolfgang,Kürzeste Wege in Graphen Algorithmen Vefahrensvergleiche,Verlag Anton Hain,Meisenheim am Glan,1972

15)Engel Arthur,Wahrscheinlichkeitsrechnung und Statistik,Band 2,Klett Verlag,1. Auflage,1976

16)Fletcher T.J.,Exemplarische Übungen zur modernen Mathematik,Herder Verlag,2. Auflage, 1967

17)Heesch Heinrich,Untersuchungen zum Vierfarbenproblem,Bibliographisches Institut,Mannheim Wien Zürich,1969

18)Hoppe H.U. und Luther W.J.,ein Fach im Spiegel neuerer Entwicklungen der Fachdidaktik,Log-IN 16,1996,Heft 1,S. 8-14

19)Horowitz Ellis Sahni Sartaj,Algorithmen,Springer Verlag,Berlin Heidelberg New York,1981

20)Röhrl Emanuel,Der Mathematikunterricht,Jahrgang 16,Elementare Topologie und Unterricht,Ernst Klett Verlag,1970

21)Foulds L.R.,Graph Theory Applications,Springer Verlag, New York,1992

22)Kaiser-Messmer,JMD 10 (89) 4,S.309-347

23)Röhrl Emanuel,Der Mathematikunterricht,Jahrgang 19,Graphentheorie,Ernst Klett Verlag,Stutgart,1973.

24)Röhrl Emanuel,Der Mathematikunterricht,Jahrgang 20,Graphentheorie II, Ernst Klett Verlag,Stutgart,1974.

25)Griesel H.,Postel H.,Elemente der Mathematik 10.Schuljahr,Schroedel-Verlag,Braunschweig,S. 226 ff

26)Harary Frank,Graphentheorie,R. Ouldenbourg Verlag,München Wien,1974.

27)Hässig Kurt,Graphentheoretische Methoden des Operations Research,Teubner Verlag,1979

28)Heesch Heinrich,Untersuchungen zum Vierfarbenproblem,1969,Informatik

29)Jungnickel Dieter,Graphen,Netzwerke und Algorithmen,B.I. Wissenschafts-verlag,Mannheim Wien Zürich, 1987

30)Kaufmann, Arnold,Einfuehrung in die Graphentheorie,R. Ouldenbourg Verlag,München und Wien,1971

31)Klingen Leo H.,Elementare Algorithmen,Herder-Verlag,Freiburg Basel Wien,1981

32)Knoedel Walter,Graphentheoretische Methoden und ihre Anwendungen,Springer Verlag,Berlin Heidelberg New York,1969

33)Knuth Donald E.,The Stanford GraphBase,Addison-Weseley New York,1993

34)Künzi H.P.,Müller O.,Nievergelt E.,Einführungskursus in die dynamische Programmierung,Springer Verlag,Berlin Heidelberg New York,1968

35)Küpper Willi,Lüder Klaus,Streitferdt Lothar,Netzplantechnik,Physica-Verlag,Würzbug Wien, 1975

36)Läuchli Peter,Algorithmische Graphentheorie,Birkhäuser Verlag,Basel Boston Berlin,1991

37)Laue Reinhard,Elemente der Graphentheorie und ihre Anwendung in den biologischen Wissenschaften,Vieweg-Verlag,Braunschweig,1971

38)Lenne Helge,Analyse der Mathematikdidaktik in Deutschland,Ernst Klett Verlag Stuttgart,1969

39)Leßner Günther,Elemente der Topologie und Graphentheorie,Herder-Verlag, Freiburg Basel Wien,1980

40)Liebling Thomas M.,Graphentheorie in Planungs-und Tourenproblemen, Springer Verlag,Berlin Heidelberg New York,1970

41)Ministerium für Schule und Weiterbildung,NRW,Wissenschaft und Forschung,50. Jahrgang,N 12,NRW,Einführungserlass zur Verordnung über den Bildungsgang und die Abiturprüfung in der gymnasialen Oberstufe,S. 222 ff,1998

42)Modrow Eckhard,Zur Didaktik des Informatikunterrichts,Bausteine Informatik,Band 2,Dümmler-Verlag

43)Modrow Eckhard,Automaten,Schaltwerke,Sprachen,Dümmler-Verlag

44)Müller Kurt Peter,Wölpert Heinrich,Anschauliche Topologie,B.G. Teubner, Stuttgart,1976

45)Noltemeier Hartmut,Graphentheorie mit Algorithmen und Anwendungen,Walter de Gruyter-Verlag,Berlin New York,1976

46)Ottmann Thomas,Widmayer Peter,Algorithmen und Datenstrukturen,B.I.- Wissenschaftsverlag,Mannheim Wien Zürich,1990

47)Ottmann Thomas,Automaten,Sprachen und Maschinen für Anwender,B.I.-Wissenschaftsverlag,Mannheim Wien Zürich,1990

48)Perl Juergen,Graphentheorie,Akademische Verlagsgesellschaft,Wiesbaden, 1981

49)Quibeldey-Cirkel Klaus,Das Objekt-Paradigma in der Informatik,B.G. Teubner Verlag,Stuttgart,1994

50)Sachs, Horst,Einführung in die Theorie der endlichen Graphen,Carl Hanser Verlag,München,1971.

51)Schaerer Daniel,Listen,Bäume und Graphen als Objekte,Springer Verlag,Wien New York,1994

52)Schneider Gerhard,Mikolcic Hrvatin,Einführung in die Methode der dynamischen Programmierung,R. Ouldenbourg Verlag,München Wien,1972

53)Schwarze, Jochen,Netzplantechnik für Praktiker,Verlag Neue Wirtschafts-Briefe,Herne/Berlin,1970

54)Tinhofer Gottfried,Methoden der angewandten Graphentheorie,Springer Verlag,Wien New York,1976

55)Turau,Algorithmische Graphentheorie,Addison-Weseley,1996

56)Volkmann Lutz,Graphen und Digraphen,Springer Verlag,Wien New York,1991

57)Völzgen Hubert,Stochastische Netzwerkverfahren und deren Anwendungen,Walter De Gruyter & Co-Verlag,Berlin New York,1971

58)Walther Hansjoachim,Anwendungen der Graphentheorie,Vieweg-Verlag,Braunschweig Wiesbaden,1978

59)Walther Hansjochim, Nägler Günter,Graphen-Algorithmen-Programme,Springer-Verlag,Wien New York,1987

60)Walther Gerd,Bodendiek Rainer,Schuhmacher Heinz,Graphen im Mathematikuntericht-eine Analyse der derzeitigen Curriculumsituation von Hannelore Pieper und Gerd Walther in Graphen in Forschung und Unterricht,Verlag Babara Franzbecker,Bad Salzdetfurth und Hildesheim,1985

61)Warken Elmar,Delphi 2,Addison Wesley Verlag,1996

62)Wippermann,Heinz,Graphen Im Unterricht, Georg Kallmeyer Verlag,Göttingen,1976

63)Wirth,Nikolaus,Algorithmen und Datenstrukturen,Teubner-Verlag

64)Zech Friedrich,Grundkurs Mathematikdidaktik,Beltz Verlag,Weinheim und Basel,2.Auflage, 1978

65)Zuchovickij,Semen I,Radtschik I. A.,Mathematische Methoden Der Netzplantechnik,B. G. Teubner Verlag,1969

66)Schulbuch Elemente der Mathematik,10.Schuljahr,Schroedel-Verlag,3-507-83710-2

67)Schriftenreihe Schule in NRW,NR 4725,Sekundarstufe II,Gymnasium/Gesamtschule,Richtlinien und Lehrpläne,Informatik,MSWF,NRW,Ritterbach-Verlag GmbH,Frechen,1.Auflage 1999

Algorithmische Graphentheorie im Unterricht unter Verwendung objektorientierter Datenstrukturen

Anhang

F Anhang

I Beschreibung der Datenstruktur 430

II Die Bedienung des Programms Knotengraph 454

III Eigenschaften der verwendeten Programmiersprache Delphi 485

IV Unterrichtspläne zu ausgewählten Unterrichtsstunden 511

V Ergebnisse und Antworten von Fragen an Schülern zu den Unterrichtsreihen,Beispiel einer Facharbeit,UML-Diagramme,Update 542

VI Quelltextlisting der verschiedenen Programme,Lösungen der Klausuraufgaben,Beschreibung der Objektmethoden im PDF-Format auf CD

F I Beschreibung der Datenstruktur

Die normale in den Lehrbüchern der Graphentheorie beschriebene Methode,einen Graph darzustellen und zu speichern,ist die Darstellung des Graphen in Form einer Matrix.Hierzu gibt es die Möglichkeiten der Darstellung als Inzidenzmatrix und als Adjazenzmatrix.Beide Möglichkeiten werden auch oft bei Bedarf zusammen angewendet.

Es sei n die Anzahl der Knoten und m die Anzahl der Kanten eines Graphen.

Die Indzidenzmatrix ist dann eine nxm Matrix,in der Elemente $a_{i,j}$ definiert sind durch:

$$a_{i,j} = \begin{cases} 1, wenn\, der\, Knoten\, x_i\, und\, die\, Kante\, k_j\, inzident\, \sin d. \\ 0, sonst. \end{cases}$$

(Inzident bedeutet,dass Knoten x_i Anfangs- oder Endknoten von k_j ist.)

Somit entsprechen die Zeilen der Matrix den Knoten und die Spalten den Kanten.In jeder Spalte befinden sich genau zwei Einsen,um eine Kante dem Anfangs-und Endknoten zuzuordnen.

Die Adjazenzmatrix ist eine nxn Matrix,in der die Elemente $a_{i,j}$ definiert sind durch:

$$a_{i,j} = \begin{cases} 1, wenn\, Knoten\, x_i\, und\, Knoten\, x_j\, adjazent\, \sin d. \\ 0, sonst. \end{cases}$$

(Adjazent bedeutet,dass die Knoten x_i und x_j Anfangs-und Endknoten einer Kante sind.)

Falls die Kanten mit Inhalten z.B. Zahlen versehen sind,können statt den Einsen auch die Inhalte in der Matrix gespeichert werden.Die Matrix ist symmetrisch (ungerichtete Kanten) zur Hauptdiagonalen,so dass nur die entsprechende Dreiecksmatrix gespeichert werden muß.

(Leider lassen sich dann den Knoten keine Inhalte mehr auf diese Weise zuordnen,sondern man benötigt ein zusätzliche Struktur.)

Die Darstellungsweise eines Graphen in Form einer Matrix hat den Vorteil,dass auf die Struktur des Graphen direkt der mathematische Formalismus des Matrizenkalküls anwendbar ist,wobei anschließend an der Struktur der Ergebnismatrix wiederum neue Eigenschaften des Graphen abgelesen werden können.

Die Darstellungsweise hat den weiteren Vorzug,dass die Datenstruktur des Arrays oder Feldes benutzt werden kann,um eine Matrix zu speichern.Diese universelle Datenstruktur ist auch in einfacheren Programmiersprachen vorhanden und erlaubt es,Algorithmen der Graphentheorie auch auf Computersystemen mit nur geringerem Haupt- und Festplattenspeichen ablaufen zu lassen.

Der Nachteil der Darstellungsweise als Matrix ist die statische Art der Speicherung.Bei der Festlegung einer Matrix oder eines Arrays,das einen Graph darstellen soll,muß die Anzahl bzw. Maximalzahl der Knoten bzw. Kanten von vornherein feststehen.Die Datenstruktur ist sowohl,was die Anzahl bzw. Maximalzahl von Knoten und Kanten als auch was zusätzliche Eigenschaf-

ten der Knoten und Kanten betrifft (,wie z.B. bei einem Flußproblem die Zuordnung von Schranken und Kosten zusätzlich zu dem Fluss zu jeder Kante) nur schlecht erweiterbar.Man müßte hier weitere Matrizen definieren,um die Daten separat abzuspeichern,oder aber bei verschachtelter Struktur wäre jedes Matrixelement wiederum als Matrix darzustellen.

Die statische Art der Speicherung erlaubt es leider nicht oder nur in sehr geringen Maße,den Komfort,den moderne Programmiersprachen wie Delphi (oder C++ bzw. Java) durch das Mittel der objektorientierten Programmierung bieten,auszunutzen.Durch die objektorientierte Vererbung wird es einem Benutzer ermöglicht,vorhandene Datenstrukturen an seine Bedürfnisse anzupassen und dabei aber auf die schon vorhandenen,vorgegebenen Algorithmen (in Form von Methoden) zugreifen zu können.Dazu darf die ursprünglich vorhandene Graphenstruktur aber nicht schon allzu starr vorgegeben sein.Es empfiehlt sich daher die Objekte Knoten und Kanten als jeweils getrennte Objekte vorzugeben und nicht schon in der Verbundform,wie sie in den oben genannten Matrizen direkt miteinander verknüpft sind.

Zwecks jederzeitiger Erweiterbarkeit der Anzahl der Objekte sollten dynamische Datenstrukturen gewählt werden.Es bietet sich hierzu als einfachste dieser Strukturen die lineare Liste an.Damit ein späterer Benutzer die Knoten und Kanten sowohl von den Dateninhalten (Werten),die den Knoten bzw. Kanten zugeordnet werden sollen,als auch von den Methoden,die diese Daten verändern und miteinander verknüpfen sollen,unter Ausnutzung der Vorteile objektorientierter Vererbung erweitern und verändern kann,ist es vorteilhaft,die genannten Listen als Liste von Zeigern auf Objekte zu konstruieren,wobei die einzelnen Objekte schon jeweils einen allgemeinen Grundvorrat an Methoden (Algorithmen) mitbringen,die die Programmierung von erweiterten Algorithmen als Software-Bausteine unterstützen.

Im folgenden soll die angedeutete Datenstruktur gemäß den Möglichkeiten,die die Programmiersprache Delphi bietet,nun näher beschrieben werden:

Ein Graph besteht aus den verschiedenen Objekten Knoten und Kanten.Nach den obigen Ausführungen ist es sinnvoll,für diese Objekte jeweils die (dynamische) Datenstruktur der Liste zu wählen.Um die Datenstrukturen als Objekt (class) organisieren zu können,empfiehlt es sich,sie als Nachfolger eines in Delphi vordefinierten Class-Datentyps zu wählen.

Als Liste bietet sich hierzu der Datentyp TList an.Das Objekt TList wird zur Verwaltung von Objektlisten benutzt.Die Property List besteht aus einer Liste von Zeigern aller Objekte der Liste.Auf die einzelnen Elemente kann mittels der Property Items zugegriffen werden.Die Elemente der Liste werden mittels der TList-Methoden Add,Delete,Insert,Remove,Move und Exchange angefügt,gelöscht,einfügt,entfernt,verschoben und vertauscht.Die Property Count gibt die Anzahl der Listenelemente an.Die Nummerierung beginnt mit 0 beim 1. Element.Die Methoden (Functionen) First und Last weisen auf das erste bzw. letzte Listenobject.

Von TList wird als Nachfolger das Objekt TListe definiert durch:

```
TListe=class(TList)
```

Der Datentyp verwendet die eben genannten Methoden von TList und stellt (unter Benutzung dieser Methoden) in der Wirkungsweise entsprechende mit deutschen Namen bezeichnete Methoden bereit (wie z.B. Einfuegen,Loeschen usw.) und erweitert die Verwaltungsmöglichkeiten der Objektliste beträchtlich durch das Bereitstellen von auf die spätere Verwendung als Kanten-und Knotenliste,deren Elementen Inhalte bzw. Werte z.B. in Form von Zahlen zugeordnet sind,ausgerichteten Methoden wie z.B. Wertsumme,Sortieren,ErsterichtigePosition,FuerjedesElement bzw. FueralleElemente (usw.).

Der Datentyp definiert (rekursiv) die Property function Element (Index: Integer):TElement als Methode,wobei TElement gleich dem Datentyp class(TListe) ist,um unter Ausnutzung der Index-Eigenschaft von TListe auf die Elemente der Liste mittels Element(Index) (wie in einem Array) zugreifen zu können.

Von TListe leiten sich durch Vererbung die Objekttypen TKnotenliste=class(TListe),TKantenliste=class(TListe) und TPfadliste=class(TListe) ab.

Der Knotentyp wird dann definiert als TKnoten=class(TKnotenliste) unter Vorgabe der (rekursiv definierten) Property function Knoten(Index:Integer):TKnoten von TKnotenliste,um unter Ausnutzung der Index-Eigenschaft von TKnotenliste auf die einzelnen Knoten mittels Knoten(Index) (wie in einem Array) zugreifen zu können.

In gleicher Weise wird dann der Kantentyp definiert als TKante= class(TKantenliste) unter Vorgabe der (rekursiv definierten) Property function Kante(Index:Integer):TKante von TKantenliste,um unter Ausnutzung der Index-Eigenschaft von TKantenliste auf die einzelnen Kanten mittels Kante(Index) (wie in einem Array) zugreifen zu können.

Die gleichen Definitionen findet sich letztlich auch bei Pfad und Pfadliste vom Typ TPfad und TPfadliste:TPfad=class(TPfadliste) mit der function Pfad(Index:Integer):TPfad.

Die Kanten und die Knoten eines Graphen sind nun keine voneinander unabhängige Objekte,sondern eine Kante ist das Verbindungselement zwischen zwei Knoten.Umgekehrt kann ein Knoten das Ausgangsobjekt oder das Zielobjekt für Kanten sein.Außerdem kann eine Kante gerichtet oder ungerichtet sein.Diese Eigenschaften müssen bei der Definition der Datentypen TKante und TKnoten zusätzlich berücksichtigt werden.

TKante erhält deshalb die zusätzlichen Datenfelder Anfangsknoten_:TKnoten und EndKnoten_:TKnoten; Pfadrichtung_: TKnoten und Gerichtet_:Boolean. Anfangsknoten_ und Endknoten_ sind die beiden zu der Kante gehörenden Randknoten,Gerichtet_ beschreibt,ob es sich um eine gerichtete oder ungerichtete Kante handelt,und die Pfadrichtung zeigt bei gerichteten Kanten auf den Endknoten.Bei ungerichteten Kanten kann diese Eigenschaft dazu verwendet werden,um z.B. die Richtung eines durchlaufenden Pfades während des Ablaufs eines Pfadalgorithmus zu beschreiben,indem einer der beiden Randknoten als Zielknoten der Kante als Pfadrichtung gespeichert wird.Die Datenfelder Besucht_:Boolean und Erreicht_:Boolean dienen dazu,Markierungen beim Suchen von Pfaden setzen zu können.

Dem Objekt TKnoten werden gemäß den obigen Überlegungen die zusätzlichen Datenfelder EingehendeKantenliste_:TKantenliste und AusgehendeKantenliste_: TKantenliste zugeordnet.In diesen Kantenlisten werden die in diesen Knoten eingehenden Kanten und die von diesem Knoten ausgehenden Kanten (als Objekt-Zeigerverweise) gespeichert.Auf diese Weise werden alle gerichteten Kanten jeweils in den Kantenlisten von zwei Knoten einmal als eingehende Kanten und einmal als ausgehende Kanten gespeichert.Gerichtete Schlingen,d.h. Kanten mit dem selben Anfangs-und Zielknoten,werden jeweils in die ein-und ausgehenden Kantenliste desselben Knotens eingefügt.
Die Eigenschaft einer Kante ungerichtet zu sein,kann als ein-und ausgehend aufgefaßt werden.Es ist also naheliegend,eine ungerichtete Kante sowohl in der ein- als auch in der ausgehenden Kantenliste eines Knoten zu speichern.Daher befindet sich eine ungerichtete Kante insgesamt in vier Kantenlisten,nämlich jeweils in der ein- und ausgehenden Kantenliste des Anfangs- und des Endknotens.Die Datenfelder Besucht_:Boolean und Erreicht_ eines Knoten dient dazu,um Markierungen bei Suchpfadalgorithmen setzen zu können.

Die Ergebnisse von Suchpfadalgorithmen sind Pfade und können als (geordnete) Folge von Kanten aufgefaßt werden.Es bietet sich daher an,Pfade als Datentyp der Kantenliste nämlich TKantenliste zu speichern.Dieser Datentyp oder einer seiner Nachfolger sollte dann auch Methoden zur Verfügung stellen,um Pfad- bzw. Kantenlisten zeichnerisch darzustellen.Methoden,um Kanten und Knoten zeichnerisch darzustellen,werden aber auch für den Datentyp des Graphen TGraph benötigt.Um doppelte Methoden zu vermeiden,die fast dasselbe bewirken,empfiehlt es sich daher Pfade als spezielle Graphen,nämlich solche mit einer Kantenliste,deren Kanten zum Pfad gehören aufzufassen.Die in diesem Graph enthaltenen Knoten sind dann gerade die Knoten des Pfades (als

Liste,in der durch den Pfad vorgegebenen Reihenfolge).In diesem Sinne ist der Datentyp TPfad als Graph aufzufassen,dessen Kanten der Kantenliste und dessen Knoten der Knotenliste den Pfad darstellen.(Im Grunde können also allgemein Graphen als TPfad gespeichert werden.Der Begriff des Pfades ist in der Literatur nicht eindeutig definiert.In einigen Literaturstellen (vgl. z.B. Lit 51,S.133) ist ein Pfad gleich einem Walk (vgl. C I S.155) und ein Weg ist dann ein **einfacher** Pfad.In diesem Sinne kann also der Typ TPfad dann auch für Walks benutzt werden.)Da bei Suchalgorithmen oft viele Pfade ermittelt werden,deren Eigenschaften miteinander verglichen werden sollen,ist es notwendig,die verschiedenen Pfade vom Typ TPfad in einer Pfadliste von Typ TPfadliste zu speichern,deren Elemente gerade Zeiger auf die verschiedenen Pfade sind.Die genauen Definitionen lauten:

```
TPfadliste  = class(TListe)

TPfad   = class(TPfadliste)
```

Mittels der (rekursiv definierten) Property-Function function Pfad(Index:Integer):TPfad kann auf die Pfade mittels Pfad(Index) durch die Index-Eigenschaft der Pfadliste (wie in einem Array) zugegriffen werden.Durch die Functionsmethoden function Knotenliste:TKnotenliste und function Kantenliste:TKantenliste kann auf die Knoten und Kanten von TPfad zugegriffen werden.Damit Pfade beim Ablauf eines Suchalgorithmus sehr flexibel gehandhabt werden können,enthält jeder Knoten des Graphen vom Typ TKnoten eine Pfadliste vom Typ TPfadliste,in der mögliche Pfade,die für diesen Knoten von Bedeutung sind,gespeichert werden können.Bei der Initialisierung des Graphen wird die Pfadliste der Knoten als leere Pfadliste erzeugt.

Wie anfangs erwähnt,besteht ein Graph aus den verschiedenen Objekten Knoten und Kanten.Es ist daher konsequent,den Typ TGraph als ein Objekt aufzufassen,das primär aus einer Knotenliste und einer Kantenliste besteht.

Die Definition von TGraph lautet also:

```
TGraph = class(TObject)
  Knotenliste_:TKnotenliste;
  Kantenliste_:TKantenliste;
  .
  weitere Definitionen
  .
  end;
```

Somit wird eine gerichtete Kante in drei Kantenlisten geführt,nämlich in der Kantenliste des Graphen,in der ausgehenden Kantenliste des Knotens,von dem die Kante ausgeht und in der eingehenden Kantenliste des Knotens,der der Zielknoten der Kante ist.Eine ungerichtete Kante ist sogar in 5 Kantenlisten vorhanden,nämlich zusätzlich zu den eben genannten Listen noch in der ausgehenden Kantenliste des Zielknoten und in der eingehenden Kantenliste des Startknotens.(s.o.)Beim Einfügen und beim Löschen von Kanten sind also alle eben genannten Listen betroffen.Jede Kante ist als Objekt im Graphen dabei nur einmal vorhanden,das Einfügen und Löschen in den Listen geschieht lediglich durch das Setzen oder Löschen von Verweisen (Zeigern) auf sie.(besondere,spezielle objektorientierte Erweiterung von Adjazenzlisten)

Bei vielen Algorithmen erweist es sich als sehr zweckmäßig,von einem Knoten des Graphen aus Zugriff auf den gesamten Graphen d.h. auf alle Knoten und Kanten haben zu können.Deshalb besitzt der Datentyp TKnoten noch zusätzlich einen Verweis auf den Graphen in Form der Function Graph_ vom Typ TGraph.So ist es außerdem auch möglich von jeder Kante aus durch deren Zeiger auf Anfangsknoten oder Endknoten die gesamte Graphenstruktur erreichen zu können.

Der Vorteil von objektorientierter Programmierung liegt darin,dass ein Benutzer eigene Datentypen als Nachkomme der vorgegeben Datentypen durch Vererbung erzeugen kann und mit diesen neuen Datentypen die vorhandenen Methoden der bisherigen Datentypen benutzen kann.Deshalb sollte es möglich sein,dass ein Benutzer sich einen Graph definieren kann,der von TGraph vererbt wird und dessen Knotentypen und Kantentypen Nachfolger von TKnoten und TKante sind.Um in diesem Graph die Methoden von TKnoten bzw. TKante und neu für diesen Datentyp definierte Methoden benutzen zu können,ist es notwendig

dass das Erzeugen von Knoten des Datentyps TKnoten und des Datentyps TKante durch virtuelle Constructoren geschieht.(Auch TGraph besitzt einen virtuellen Constructor.)Der Vorteil der Programmiersprache Delphi liegt u.a. darin,dass sie die Möglichkeit solcher virtuellen Constructoren bereitstellt.

Beim Speichern und Laden eines Graphen auf bzw. von einem Datenträger tritt bei solchen durch Vererbung erzeugten Graphen mit Knoten und Kanten als Nachfolger der vorhandenen Datentypen TKnoten und TKante das Problem auf,dass der Inhalt neu hinzugekommener Datenfelder von den in TGraph vorgegebenen Methoden nicht mit abgespeichert bzw. nicht mit eingelesen wird.Es müßten also jeweils neue Methoden für diese Datentypen erstellt werden,die die bisherigen Methoden überschreiben.

Um dieses Problem zu umgehen,enthält jeder der oben genannten Datentypen TElement,TKnoten,TKante,und TGraph ein zusätzliches Datenfeld Wertliste_ vom Typ TStringlist.TStringlist ist ein in Delphi vordefinierter Datentyp und ist eine Liste von strings.Die Inhalte der Datenfelder von Nachkommen der Typen TKnoten,TKante und TGraph sollten,nachdem sie in den Datentyp string konvertiert wurden,in der Reihenfolge ihres Vorkommens in der Objekttypdeklaration in diese Stringliste eingefügt werden.Geschieht dies für alle zu speichernden Datenfelder der von einem Benutzer durch Vererbung erstellten Datentypen,können die von TGraph vorgegebenen Methoden SpeichereGraph und LeseGraph die Inhalte der neu hinzugekommenen Datenfelder berücksichtigen,indem sie die strings der Wertliste auf einem Datenträger abspeichern bzw. wieder einlesen.Daher sollten die Datentypen TKnoten,TKante und TGraph und auch alle von diesen Datentypen vererbten Typen die folgenden beiden virtuellen Methoden enthalten.:

```
function Wertlisteschreiben:TStringlist;
procedure Wertlistelesen;
```

Am Beispiel des Datentyps TInhaltsknoten als Nachfolger von TKnoten sowie an Nachfolgerobjekten von TKante mit zusätzlichen Datenfeldern wird im folgenden der prinzipielle Aufbau der Methoden erläutert.

Um den Knoten des Graphen einen Wert (z.B. einen Bezeichner oder einen Zahlenwert) zuordnen zu können,wird zunächst das Objekt TInhaltsknoten deklariert,das als zusätzliches Datenfeld das Feld Inhalt_ vom Typ string enthält:

```
TInhaltsknoten = class(TKnoten)
private
    .
    Inhalt_:string;
    .
public
    constructor Create;override;
    procedure Free;
    procedure Freeall;
    function Wertlisteschreiben:TStringList;override;
    procedure Wertlistelesen;override;
    .
    .
    .
  end;
```

Die Deklaration enthält die Methoden Wertlisteschreiben und Wertlistelesen:

```
function TInhaltsknoten.Wertlisteschreiben:TStringlist;
begin
  Wertliste.Clear;
  Wertliste.Add(Inhalt_);
  Wertlisteschreiben:=Wertliste;
end;

procedure TInhaltsknoten.Wertlistelesen;
begin
  Inhalt_:=Wertliste.Strings[0];
end;
```

Methode Wertlisteschreiben:

Es wird zunächst eine leere Wertliste erzeugt.Danach werden die Inhalte aller Datenfelder des Datentyps einschließlich der Datenfelder der Vorgänger (wenn nötig durch Typkonversion) als strings umgewandelt in die Wertliste eingefügt.Zum Schluß wird diese Liste als Wert der Function zurückgegeben.

Methode Wertlistelesen:

Die strings der Stringliste Wertliste werden der Reihenfolge nach (evtl. mittels Konversion in den definierten Datentyp) als Inhalte der Datenfelder des Datentyps gespeichert.

Entsprechend ist auch für Nachfolger des Objekts TKante zu verfahren.Von TKante leitet sich zunächst das Objekt TInhaltskante ab,das wiederum ein Feld Inhalt_ vom Datentyp string zur Aufnahme von Bezeichner oder Zahlenwerten enthält.

Die Methoden Wertlistelesen und Wertlisteschreiben lauten für TInhaltskante:

```
function TInhaltskante.Wertlisteschreiben:TStringList;
begin
  Wertliste.Clear;
  Wertliste.Add(Inhalt_);
  Wertlisteschreiben:=Wertliste;
end;

procedure TInhaltskante.Wertlistelesen;
begin
  Inhalt_:=Wertliste.Strings[0];
end;
```

Das Objekt TMinimalekostenkante ist definiert durch:

```
TMinimaleKostenkante = class(TInhaltskante)
  private
    Fluss_:Extended;
    Kosten_:Extended;
    ModKosten_:string;
    KostenWert_:Extended;
    Schranke_:Extended;
    Richtung_:Boolean;
    PartnerKante_:Integer;
    Ergebnis_:string;
    procedure SetzeFluss(Fl:Extended);
    function WelcherFluss:Extended;
    procedure SetzeKosten(Ko:Extended);
    function WelcheKosten:Extended;
    procedure SetzeModKosten(Mko:string);
    function WelcheModKosten:string;
    procedure SetzeKostenWert(Ako:Extended);
    function WelcherKostenWert:Extended;
    procedure SetzeSchranke(Schr:Extended);
    function WelcheSchranke:Extended;
    procedure SetzeRichtung(Ri:Boolean);
    function WelcheRichtung:Boolean;
    procedure SetzePartnerKante(Ka:TMinimaleKostenkante);
    function  WelchePartnerkante:TMinimalekostenkante;
    procedure SetzeKantenindex(I:Integer);
    function WelcherKantenindex:Integer;
    procedure SetzeErgebnis(Erg:string);
    function WelchesErgebnis:string;
  public
    constructor Create;
    function Wertlisteschreiben:TStringlist;override;
    procedure Wertlistelesen;override;
    property Fluss:Extended read WelcherFluss write SetzeFluss;
    property Kosten:Extended read WelcheKosten write SetzeKosten;
    property ModKosten:string read WelcheModKosten write SetzeModKosten;
    property KostenWert:Extended read WelcherKostenWert Write setzeKostenWert;
    property Schranke:Extended read WelcheSchranke write setzeSchranke;
    property Richtung:Boolean read WelcheRichtung write SetzeRichtung;
    property Partnerkante:TMinimaleKostenkante read WelchePartnerkante
      write  SetzePartnerkante;
    property Kantenindex:Integer read WelcherKantenindex write SetzeKantenindex;
    property Ergebnis:string read WelchesErgebnis write setzeErgebnis;
end;
```

Die Methoden Wertlisteschreiben und Wertlistelesen enthalten dann folgenden Quelltext:

```
function TMinimaleKostenKante.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;
  Wertliste.Add(Realtostring(Fluss));
  Wertliste.Add(Realtostring(Kosten));
  Wertliste.Add(ModKosten);
  Wertliste.Add(Realtostring(KostenWert));
  Wertliste.Add(Realtostring(Schranke));
  if Richtung then Wertliste.Add('true') else Wertliste.Add('false');
  Wertliste.Add(Integertostring(Kantenindex));
  Wertliste.Add(Ergebnis);
  Wertlisteschreiben:=Wertliste;
end;

procedure TMinimaleKostenKante.Wertlistelesen;
begin
  inherited Wertlistelesen;
  Fluss:=StringtoReal(Wertliste.Strings[Wertliste.Count-8]);
  Kosten:=StringtoReal(Wertliste.Strings[Wertliste.Count-7]);
  ModKosten:=Wertliste.Strings[Wertliste.Count-6];
  KostenWert:=StringtoReal(Wertliste.Strings[Wertliste.Count-5]);
  Schranke:=StringtoReal(Wertliste.Strings[Wertliste.Count-4]);
  if Wertliste.Strings[Wertliste.Count-3]='true'
  then
    Richtung:=true
  else
    Richtung:=false;
  Kantenindex:=StringtoInteger(Wertliste.Strings[Wertliste.Count-2]);
  Ergebnis:=Wertliste.Strings[Wertliste.Count-1];
end;
```

Die Anweisung inherited Wertlisteschreiben speichert die Datenfelder der Vorgängerobjekte.Ebenso liest inherited Wertlistelesen die Datenfelder der Vorgängerobjekte wieder ein.Wie man sieht,werden die Datenfelder der Reihe nach konvertiert als strings in die Stringliste eingefügt bzw. wieder ausgelesen.Als Besonderheit wird hier die Partnerkante (Datentyp Zeiger auf ein Objekt) in Form ihres Kantenindex (umgewandelt in einen string) gespeichert und entsprechend wieder zurückverwandelt.

Die Objekte TKnoten,TKante und TGraph (von dem sich das Objekt TInhaltsgraph ableitet) enthalten die Methoden Wertlisteschreiben und Wertlistelesen mit der Deklaration abstract (kein Quelltext),weil sie noch keine Datenfelder enthalten,jedoch schon virtuelle Methoden für von ihnen vererbte Objekte bereitstellen.

Jeder der drei Datentypen TKnoten,TKante und TGraph definiert eine Property Wert vom Typ string und außerdem eine Property Position vom Typ Integer mittels der Methoden SetzeWertposition und LoescheWertposition.Position gibt dabei eine mögliche Elementposition der Wertliste an.Die Property Wert gibt dann das Element der Wertliste,das durch Position bestimmt ist,als string zurück.Dadurch ist es möglich den zurückzugebenden oder den zuzuweisenden Wert eines dieser Objekte mittels Änderung von Position leicht zu variieren.

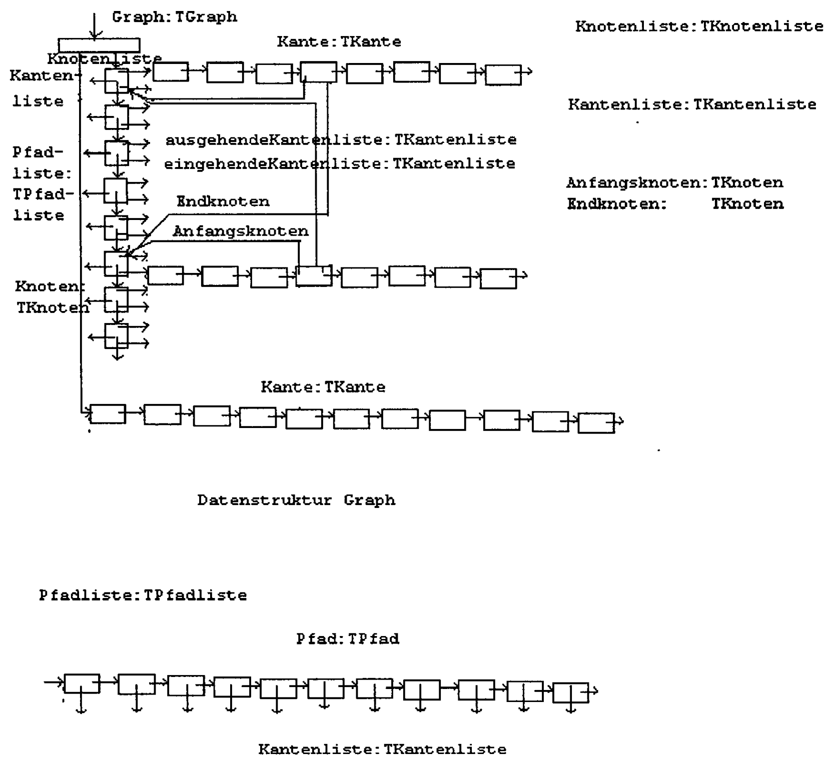
Da die Methode ZeichneGraph prinzipiell die Werte der Knoten und Kanten auf der Zeichenoberfläche ausgibt,die durch die Property Wert bestimmt sind,genügt es also lediglich die (Wert-) Position der Knoten und Kanten des Graphen zu ändern,um den Inhalt beliebiger Datenfelder von Nachfolgern von TKnoten und TKante darstellen zu können.
(Da Wert vom Typ string ist,weil nur strings auf der Zeichenoberfläche dargestellt werden können,ist die oben beschrieben Typkonversion der Datenfelder in die strings der Wertliste notwendig und keine Einschränkung.)

Der Datentyp TGraph stellt mit den Datentypen TKnoten und TKante (als Elemente der Knoten-und Kantenliste) gerade solche Methoden zur Verwaltung eines Graphen bereit,die sich beschreiben lassen,ohne auf die diesen Objekten zugeordnete Werte wie (Knoten-)Inhalt,(Kanten-)Inhalt,Koordinaten,Typ, zeichnerische Kanten-und Knotendarstellung,wie z.B. Farbe oder (Zeichen-)

Stil,Knotenradius oder Kantenweite zurückgreifen zu müssen.(TGraph als ADT,d.h. als reine Beschreibung der Knoten-Kantenrelation.)

Die Units UListe und UGraph enthalten die bisher besprochene Datenstruktur.Die folgende Graphik stellt diese Datenstruktur des Graphen als Zeichnung dar.UML-Diagramme zur Gesamtstruktur befinden sich am Anhangsende.

Graphische Übersicht über die Datenstruktur:



Datenstruktur Graph

Übersicht über die Datenstruktur als Typ-Deklaration:

```
  TListe=class(Tlist)
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    function Element(Index:Integer):TElement;
    property Items[Index:Integer]:TElement read Element;
    .
    weitere Methoden
    .
  end;

  TElement=class(TListe)
  private
    Wertposition_:Integer;
    Wertliste_:TStringlist;
    procedure SetzeWertposition(P:Integer);
    function WelcheWertposition:Integer;
    procedure Wertschreiben(S:string);virtual;
    function Wertlesen:string;virtual;
  public
    constructor Create;
    procedure Free;
    property Position:Integer read WelcheWertposition write SetzeWertposition;
    function Wertlisteschreiben:Tstringlist;virtual;abstract;
    procedure Wertlistelesen;virtual;abstract;
    property Wert:string read Wertlesen write Wertschreiben;
   end;

TKantenliste = class(TListe)
    public
```

```
    constructor Create;
    procedure Free;
    procedure Freeall;
    function Kante(Index:Integer):TKante;
    property Items[Index:Integer]:TKante read Kante;
    .
    weitere Methoden
    .
end;

  TKante =  class(TKantenliste)
  private
    Anfangsknoten_:TKnoten;
    Endknoten_:TKnoten;
    Pfadrichtung_:TKnoten;
    gerichtet_:Boolean;
    Wertposition_:Integer;
    Wertliste_:TStringlist;
    Besucht_:Boolean;
    Erreicht_:Boolean;
    function WelcherAnfangsknoten:TKnoten;
    procedure SetzeAnfangsknoten(Kno:TKnoten);
    function WelcherEndknoten:TKnoten;
    procedure SetzeEndknoten(Kno:TKnoten);
    function WelchePfadrichtung:TKnoten;
    procedure SetzePfadrichtung(Kno:TKnoten);
    function Istgerichtet: Boolean;
    procedure Setzegerichtet(Gerichtet:Boolean);
    procedure SetzeWertposition(P:Integer);
    function WelcheWertposition:Integer;
    function WelcheWertliste:TStringlist;
    procedure SetzeWertliste(W:TStringlist);
    function Istbesucht:Boolean;
    procedure Setzebesucht(B:Boolean);
    function Isterreicht:Boolean;
    procedure Setzeerreicht(E:Boolean);
    procedure Wertschreiben(S:string);
    function Wertlesen:string;
  public
    constructor Create;virtual;
    procedure Free;
    procedure Freeall;
    property Anfangsknoten:TKnoten read WelcherAnfangsknoten write SetzeAnfangsknoten;
    property Endknoten:TKnoten read WelcherEndknoten write SetzeEndknoten;
    property Pfadrichtung:TKnoten read WelchePfadrichtung write SetzePfadrichtung;
    property Gerichtet:Boolean read Istgerichtet write Setzegerichtet;
    property Position:Integer read WelcheWertposition write SetzeWertposition;
    property Wertliste:TStringlist read WelcheWertliste write SetzeWertliste;
    property Besucht:Boolean read Istbesucht write Setzebesucht;
    property Erreicht:Boolean read Isterreicht write Setzeerreicht;
    property Wert:string read Wertlesen write Wertschreiben;
    function Wertlisteschreiben:Tstringlist;virtual;abstract;
    procedure Wertlistelesen;virtual;abstract;
    .
   weitere Methoden
   .
  end;

  TPfadliste  = class(TListe)
    public
    constructor Create;
    procedure Free;
    procedure Freeall;
    function Pfad(Index:Integer):TPfad;
    property Items[Index:Integer]:TPfad read Pfad;
    .
    weitere Methoden
   .
  end;

  TPfad   = class(TPfadliste)
    public
    constructor Create;
    procedure Free;
    .
    weitere Methoden
    .
  end;

  TKnotenliste = class(TListe)
  public
    constructor Create;
    procedure Free;
```

```
  procedure Freeall;
  function Knoten(Index:Integer):TKnoten;
  property Items[Index:Integer]:TKnoten read Knoten;
  .
  weitere Methoden
  .
end;

TKnoten = class(TKnotenliste)
private
  Graph_:TGraph;
  EingehendeKantenliste_:TKantenliste;
  AusgehendeKantenliste_:TKantenliste;
  Pfadliste_:TPfadliste;
  Wertposition_:Integer;
  Wertliste_:TStringlist;
  Besucht_:Boolean;
  Erreicht_:Boolean;
  function WelcherGraph:TGraph;
  procedure SetzeGraph(G:TGraph);
  function WelcheEingehendeKantenliste:TKantenliste;
  procedure SetzeEingehendeKantenliste(L:TKantenliste);
  function WelcheAusgehendeKantenliste:TKantenliste;
  procedure SetzeAusgehendeKantenliste(L:TKantenliste);
  function WelchePfadliste:TPfadliste;
  procedure SetzePfadliste(P:TPfadliste);
  procedure SetzeWertposition(P:Integer);
  function WelcheWertposition:Integer;
  function WelcheWertliste:TStringlist;
  procedure SetzeWertliste(W:TStringlist);
  function Istbesucht:Boolean;
  procedure Setzebesucht(B:Boolean);
  function Isterreicht:Boolean;
  procedure Setzeerreicht(E:Boolean);
  procedure Wertschreiben(S:string);
  function Wertlesen:string;
public
  constructor Create;virtual;
  procedure Free;
  procedure Freeall;
  property Graph:TGraph read WelcherGraph write SetzeGraph;
  property EingehendeKantenliste:TKantenliste read WelcheEingehendeKantenliste
  write SetzeEingehendeKantenliste;
  property AusgehendeKantenliste:TKantenliste read WelcheAusgehendeKantenliste
  write SetzeAusgehendeKantenliste;
  property Pfadliste:TPfadliste read WelchePfadliste write SetzePfadliste;
  property Position:Integer read WelcheWertposition write SetzeWertposition;
  property Wertliste:TStringlist read WelcheWertliste write SetzeWertliste;
  property Besucht:Boolean read Istbesucht write Setzebesucht;
  property Erreicht:Boolean read Isterreicht write Setzeerreicht;
  property Wert:string read Wertlesen write Wertschreiben;
  function Wertlisteschreiben:Tstringlist;virtual;abstract;
  procedure Wertlistelesen;virtual;abstract;
  .
  weitere Methoden
  .
end;

TGraph = class(TObject)
private
  Knotenliste_:TKnotenliste;
  Kantenliste_:TKantenliste;
  Wertposition_:Integer;
  Wertliste_:TStringlist;
  function WelcheKnotenliste:TKnotenliste;
  procedure SetzeKnotenliste(K:TKnotenliste);
  function WelcheKantenliste:TKantenliste;
  procedure SetzeKantenliste(K:TKantenliste);
  procedure SetzeWertposition(P:Integer);
  function WelcheWertposition:Integer;
  function WelcheWertliste:TStringlist;
  procedure SetzeWertliste(W:TStringlist);
  function Wertlesen:string;
  procedure Wertschreiben(S:string);
public
  constructor Create;virtual;
  procedure Free;
  procedure Freeall;
  property Knotenliste:TKnotenliste read WelcheKnotenliste write SetzeKnotenliste;
  property Kantenliste:TKantenliste read WelcheKantenliste write SetzeKantenliste;
  property Position:Integer read WelcheWertposition write SetzeWertposition;
  property Wertliste:TStringlist read WelcheWertliste write SetzeWertliste;
  property Wert:string read Wertlesen write Wertschreiben;
```

```
    function Wertlisteschreiben:TStringlist;virtual;abstract;
    procedure Wertlistelesen;virtual;abstract;
    .
    weitere Methoden
    .
  end;
```

Wie schon oben erwähnt,leiten sich von den Objekten TKnoten,TKante und TGraph durch Vererbung die Objektstrukturen TInhaltskante,TInhaltsknoten und TInhaltsgraph ab.Diese werden in der Unit UInhgrph mit den dazugehörigen Methoden definiert.Die Erweiterung stellt hauptsächlich die benötigten Datenfelder und Methoden bereit,um die genannten Objekte zeichnerisch auf der Objekt-Zeichenfläche darzustellen.Nach den obigen Ausführungen enthalten die Objekttypen TInhaltsknoten und TInhaltskante zudem jeweils ein Datenfeld vom Typ string (s.o.),um einen Knoten-bzw. Kanteninhalt aufnehmen zu können (Feld Inhalt_).Diese Felder erweisen sich als nützlich,weil bei vielen Algorithmen der Graphentheorie den Knoten oder Kanten ein Inhalt in Form einer Bezeichnung oder einer Bewertungszahl zugewiesen wird.Die Beschränkung auf den Datentyp string ist nicht besonders einschränkend,da unter Delphi die Ein- und Ausgaben sowieso nur als strings vorgenommen werden können und anderseits von der Unit UList geeignete Proceduren zur Typkonversion (in Realzahlen bzw. den Datentyp Extended oder Integerzahlen und wieder zurück) bereitgestellt werden.

TInhaltsknoten enthält zur Darstellung von Knoten dieses Typs auf der Zeichenoberfläche die Felder X_ und Y_ zur Aufnahme von Koordinaten,das Feld Radius_ für den Radius des Kreises,der den Knotenrand darstellt,das Feld Farbe_ für die Farbe,mit der der Knotenrand gezeichnet wird,Stil_ für die Linienart des Knotenrandes und Typ_ für die Art und Weise,wie der als string in Inhalt_ gespeicherte Wert aufgefaßt werden soll (z.B. als Real-Zahl,Integer-Zahl oder als Zeichenkette).Auf die Felder kann mittels Methoden durch die entsprechenden Propertys zugegriffen werden.

TInhaltskante enthält mit dem Feld Farbe_ die Farbe,mit der die Kante gezeichnet wird,mit dem Feld Stil_ die Linienart der Kante,mit dem Feld Weite_ die Weite,um die die Kantenmitte von der direkten Verbindungsstrecke der Knoten entfernt ist,und mit dem Feld Typ_ eine Möglichkeit,um den Typ des Inhalts einer Kante festzulegen (z.B. als Real-Zahl,Integer-Zahl oder als Zeichenkette).Auf die Felder kann ebenfalls mittels Methoden durch die entsprechenden Propertys zugegriffen werden.

TInhaltsgraph enthält die Felder MomentaneKanenliste_ und MomentaneKnotenliste,die die Möglichkeit bieten mittels ihrer Propertys dort eine Knoten- bzw. Kantenliste des Graphen global zwischenzuspeichern.Außerdem wird der Objekttyp der im Graph enthaltenden Knoten und Kanten in den beiden Feldern InhaltsKnotenclass_ und InhaltsKantenclass_ festgehalten.Durch diese Felder bzw. ihre Propertys ist es möglich,Knoten-und Kantenobjekte mittels virtueller Constructoren zu erzeugen.Die übrigen Private-Felder,auf die wiederum nur mit Methoden bzw. als Property zugegriffen werden kann,dienen wieder überwiegend der zeichnerischen Darstellung oder zum Speichern und Laden des Graphen:

Knotengenauigkeit_ und Kantengenauigkeit_ (Genauigkeit mit der die Knoten- und Kanteninhalte z.B. als Zahlen dargestellt werden),Knotenwertposition_ und Kantenwertposition_ (Position für alle Knoten bzw. Kanten des Graphen des in der Wertliste befindlichen strings,der als Wert (s.o.) ausgegeben wird),Demo_ und Pausenzeit_ (Eigenschaften der Darstellung des Graphen im Demonstrationsmodus),Zustand_ und Stop_ (Flags,um die Auswahl von Elementen im Graph durch ereignisgesteuerte Methoden mit der Maus zu gewährleisten) sowie Dateiname_ und Graphistgespeichert_ (Unterstützung des Speichern und Laden des Graphen).

(Die Knoten K1_ bis K4_ dienen,um beim Ablauf einiger (interner) Methoden Zwischenergebnisse aufnehmen zu können.Auf sie soll und kann (durch die private-Deklaration) durch einen späteren Benutzer nicht zugegriffen werden.)

```
TInhaltsknoten = class(TKnoten)
  private
    X_,Y_:Integer;
    Farbe_:TColor;
    Stil_:TPenstyle;
    Typ_:char;
    Radius_:Integer;
    Inhalt_:string;
    function Lesex:Integer;
    procedure Schreibex(X:Integer);
    function Lesey:Integer;
    procedure Schreibey(Y:Integer);
    function Welcherradius:Integer;
    procedure Setzeradius(R:Integer);
    function WelcheFarbe:TColor;
    procedure SetzeFarbe(F:TColor);
    function WelcherStil:TPenstyle;
    procedure SetzeStil(T:TPenstyle);
    function WelcherTyp:Char;
    procedure SetzeTyp(Typ:Char);
  public
    constructor Create;override;
    procedure Free;
    procedure Freeall;
    function Wertlisteschreiben:TStringList;override;
    procedure Wertlistelesen;override;
    property X:Integer read Lesex write Schreibex;
    property Y:Integer read Lesey write Schreibey;
    property Radius:Integer read Welcherradius write Setzeradius;
    property Farbe:TColor read WelcheFarbe write SetzeFarbe;
    property Stil:TPenstyle read WelcherStil write SetzeStil;
    property Typ:char read WelcherTyp write SetzeTyp;
    .
    weitere Methoden
    .
  end;

  TInhaltsknotenclass = class of TInhaltsknoten;

  TInhaltskante = class(TKante)
  private
    Farbe_:TColor;
    Stil_:TPenstyle;
    Weite_:Integer;
    Typ_:Char;
    Inhalt_:String;
    function Welchertyp:char;
    procedure Setzetyp(Typ:char);
    function Welcheweite:Integer;
    procedure SetzeWeite(Weite:Integer);
    function WelcheFarbe:TColor;
    procedure SetzeFarbe(F:TColor);
    function WelcherStil:TPenstyle;
    procedure SetzeStil(T:TPenstyle);
  public
    constructor Create;override;
    procedure Free;
    procedure Freeall;
    function Wertlisteschreiben:TStringList;override;
    procedure Wertlistelesen;override;
    property Typ:char read WelcherTyp write SetzeTyp;
    property Weite:Integer read WelcheWeite write SetzeWeite;
    property Farbe:TColor read WelcheFarbe write SetzeFarbe;
    property Stil:TPenstyle read WelcherStil write SetzeStil;
    .
    weitere Methoden
    .
  end;

  TInhaltskanteclass = class of TInhaltskante;

  TInhaltsgraph   =  class;

  TInhaltsgraphclass =class of TInhaltsgraph;

  TInhaltsgraph      =     class(TGraph)
  private
    Knotenwertposition_:Integer;
    Kantenwertposition_:Integer;
    Demo_:Boolean;
    Pausenzeit_:Integer;
    Zustand_:Boolean;
    Stop_:Boolean;
```

```
    Knotengenauigkeit_:Integer;
    Kantengenauigkeit_:Integer;
    Radius_:Integer;
    Liniendicke_:Integer;
    Graphistgespeichert_:Boolean;
    Inhaltsknotenclass_:TInhaltsknotenclass;
    Inhaltskanteclass_:TInhaltskanteclass;
    MomentaneKnotenliste_:TKnotenliste;
    MomentaneKantenliste_:TKantenliste;
    Dateiname_:string;
    K1_,K2_,K3_,K4_:TInhaltsknoten;
    function WelcheKnotenwertposition:Integer;
    procedure SetzeKnotenwertposition(P:Integer);
    function WelcheKantenwertposition:Integer;
    procedure SetzeKantenwertposition(P:Integer);
    function WelcheWartezeit:Integer;
    procedure SetzeWartezeit(Wz:Integer);
    function WelcherDemomodus:Boolean;
    procedure SetzeDemomodus(D:Boolean);
    function WelcherEingabezustand:Boolean;
    procedure SetzeEingabezustand(Ezsd:Boolean);
    function WelcherStopzustand:Boolean;
    procedure SetzeStopzustand(Stop:Boolean);
    function WelcheKnotengenauigkeit:Integer;
    procedure SetzeKnotengenauigkeit(G:Integer);
    function WelcheKantengenauigkeit:Integer;
    procedure SetzeKantengenauigkeit(G:Integer);
    function WelcherKnotenradius:Integer;
    procedure SetzeKnotenradius(R:Integer);
    function WelcherKnotenradius:Integer;
    procedure SetzeKnotenradius(R:Integer);
    function IstGraphgespeichert:Boolean;
    procedure SetzeGraphgespeichert(Gesp:Boolean);
    function WelcherDateiname:string;
    procedure SetzeDateiname(N:string);
    procedure SetzeletztenMausklickKnoten(Kno:TInhaltsknoten);
    function WelcherletzteMausklickKnoten:TInhaltsknoten;
    function WelcheKnotenclass:TInhaltsknotenclass;
    procedure SetzeKnotenclass(Inhaltsclass:TInhaltsknotenclass);
    function WelcheKantenclass:TInhaltskanteclass;
    procedure SetzeKantenclass(Inhaltsclass:TInhaltskanteclass);
    function WelcheKnotenliste:TKnotenliste;
    procedure SetzeKnotenliste(L:TKnotenliste);
    function WelcheKantenliste:TKantenliste;
    procedure SetzeKantenliste(L:TKantenliste);
    function WelcherK1:TInhaltsknoten;
    procedure SetzeK1(Kno:TInhaltsknoten);
    property K1:TInhaltsknoten read WelcherK1 write SetzeK1;
    function WelcherK2:TInhaltsknoten;
    procedure SetzeK2(Kno:TInhaltsknoten);
    property K2:TInhaltsknoten read WelcherK2 write SetzeK2;
    function WelcherK3:TInhaltsknoten;
    procedure SetzeK3(Kno:TInhaltsknoten);
    property K3:TInhaltsknoten read WelcherK3 write SetzeK3;
    function WelcherK4:TInhaltsknoten;
    procedure SetzeK4(Kno:TInhaltsknoten);
    property K4:TInhaltsknoten read WelcherK4 write SetzeK4;
  public
    constructor Create;override;
    procedure Free;
    procedure Freeall;
    property MomentaneKnotenliste:TKnotenliste
      read WelcheKnotenliste write SetzeKnotenliste;
    property MomentaneKantenliste:TKantenliste
     read WelcheKantenliste write SetzeKantenliste;
    property Inhaltsknotenclass:TInhaltsknotenclass read WelcheKnotenclass
      write SetzeKnotenclass;
    property Inhaltskanteclass:TInhaltskanteclass read WelcheKantenclass
      write SetzeKantenclass;
    property Knotenwertposition:Integer read WelcheKnotenwertposition
      write SetzeKnotenwertposition;
    property Kantenwertposition:Integer read WelcheKantenwertposition
      write SetzeKantenwertposition;
    property Pausenzeit:Integer read WelcheWartezeit write SetzeWartezeit;
    property Demo:Boolean read WelcherDemomodus write SetzeDemomodus;
    property Zustand:Boolean read WelcherEingabezustand
    write SetzeEingabezustand;
    property Stop:Boolean read WelcherStopzustand write SetzeStopzustand;
    property Knotengenauigkeit:Integer
      read WelcheKnotengenauigkeit write SetzeKnotengenauigkeit;
    property Kantengenauigkeit:Integer
      read WelcheKantengenauigkeit write SetzeKantengenauigkeit;
    property Radius:Integer read WelcherKnotenradius write SetzeKnotenradius;
```

```
    property Liniendicke:Integer read WelcheLiniendicke write SetzeLiniendicke;
    property Graphistgespeichert:Boolean read IstGraphgespeichert
      write SetzeGraphgespeichert;
    property Dateiname:string read WelcherDateiname write SetzeDateiname;
    property LetzterMausklickknoten:TInhaltsknoten
      read WelcherLetzteMausklickknoten
      write SetzeletztenMausklickknoten;
    function Wertlisteschreiben:TStringList;override;
    procedure Wertlistelesen;override;
  .
  weitere Methoden
  .
  end;
```

Von den Strukturen TInhaltsknoten,TInhaltsKante und TInhaltsgraph leiten sich nun durch Vererbung die Datenstrukturen der bei den einzelnen mathematischen Anwendungen benutzten Graphen ab.Auf sie wird jeweils bei der Beschreibung der Anwendungen (siehe Kapitel C) eingegangen.

Durch die Definition jeweils einer neuen Datenstruktur für jede dieser Anwendungen,wird deutlich gemacht,welche Felder und Methoden jede dieser Anwendungen zusätzlich benötigt,und es wird ermöglicht,die einzelnen Anwendungen unter didaktischen Gesichtspunkten unabhängig voneinander darzustellen.

Dazu wird wesentlich auf die Methode InhaltskopiedesGraphen von TInhaltsgraph zurückgegriffen,die es erlaubt die Kopie eines Graphen zu erstellen,wobei durch die in dieser Methode übergebenden Parameter gewählt werden kann,von welchem Objektdatentyp die neuen Knoten und Kanten sowie der neue Graph selber sein sollen.Der Objektdatentyp der Knoten-,Kanten- und der Graphenkopie muß dazu jeweils in einer Vererbungsbeziehung (Nachfolger oder Vorgänger) zu dem Kanten-, Knoten- oder Graphentyp des vorgegebenen Graphen stehen.

z.B.(Graph ist vom Typ TInhaltsgraph.):

```
F:=TMarkovgraph(Graph.InhaltskopiedesGraphen(TMarkovgraph,TMarkovknoten,TMarkovkante,false));
```

(Der letzte Parameter bestimmt,ob die Kanten des Graphen in ungerichtete Kanten umgewandelt werden (true) oder nicht (false).)

Es ist jedoch natürlich auch möglich ohne Benutzung der Methode InhaltskopiedesGraphen direkt Nachfolger von TInhaltsgraph,TInhaltsknoten und TInhaltskante durch Vererbung abzuleiten und mit deren Hilfe entsprechende Instanzen zu definieren.Die Technik dazu beschreibt der folgende Abschnitt.

Erzeugen neuer Graphentypen durch Vererben:

Um einen neuen Graphentypen mit neuen Kanten und Knotendatentypen zu erzeugen,sind zunächst die einzelnen Datentypen durch Vererbung von TInhaltsgraph,TInhaltsknoten und TInhaltskante zu definieren:

Das folgende Beispiel definiert die Datentypen TXknoten,TXKante und TXGraph,die sich jeweils von TInhaltsknoten,TInhaltskante und TInhaltsgraph ableiten:

```
TXKnoten =  class(TInhaltsknoten)
            KnotenFeld_:string;
            weitere Felder
            constructor create;override;
            function Wertlisteschreiben:TStringlist;override;
            procedure Wertlistelesen;override;
           weitere Methoden
           end;

TXKante = class(TInhaltskante)
          KanteFeld_:string;
          weitere Felder
          constructor create;override;
          function Wertlisteschreiben:TStringlist;override;
          procedure Wertlistelesen;override;
          weitere Methoden
          end;
```

```
TXGraph = class(TInhaltsgraph)
           GraphFeld_:string;
           weitere Felder
           constructor create;override;
           function Wertlisteschreiben:Tstringlist;override;
           procedure Wertlistelesen;override;
           weitere Methoden
           end;
```

Die neuen Datentypen enthalten die neuen Datenfelder KnotenFeld_,KanteFeld_ und GraphFeld_ (beispielsweise) vom Datentyp string (und eventuell weitere Felder).

Jeder der Datentypen beinhaltet die obligatorischen Methoden Wertlisteschreiben und Wertlistelesen.Der prinzipielle Aufbau erfolgt wiederum nach dem schon weiter oben angegebenen Schema:

```
function TXKnoten.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;
  Wertliste.add(KnotenFeld_);
  Wertlisteschreiben:=Wertliste;
 end;

procedure TXKnoten.Wertlistelesen;
begin
  inherited Wertlistelesen;
   KnotenFeld_:=Wertliste.Strings[Wertliste.Count-1];
end;

function TXKante.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;
  Wertliste.add(KanteFeld_);
  Wertlisteschreiben:=Wertliste;
 end;

procedure TXKante.Wertlistelesen;
begin
  inherited Wertlistelesen;
   KanteFeld_:=Wertliste.Strings[Wertliste.Count-1];
end;

function TXGraph.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;
    Wertliste.add(GraphFeld_);
    Wertlisteschreiben:=Wertliste;
 end;

procedure TXGraph.Wertlistelesen;
begin
  inherited Wertlistelesen;
   GraphFeld_:=Wertliste.Strings[Wertliste.Count-1];
end;
```

Die Constructoren der drei Objekte sind im folgenden dargestellt.Sie besetzen das Datenfeld mit der Wert 'x' und rufen jeweils die Methode Wertlisteschreiben auf.

```
constructor TXKnoten.Create;
 begin
   inherited Create;
   KnotenFeld_:='x';
   Wertlisteschreiben;
 end;

constructor TXKante.Create;
 begin
   inherited Create;
   KanteFeld_:='x';
   Wertlisteschreiben;
 end;

constructor TXgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TXKnoten;
  InhaltsKanteclass:=TXKante;
```

```
  GraphFeld_:='x';
  Wertlisteschreiben;
  Knotenformular.Graph:=self;
  end;
```

Der Constructor zu TXGraph weist mittels der Propertys InhaltsKnotenclass und InhaltsKantenclass den Feldern InhaltsKnotenclass_ und InhaltsKanteclass_ die im Graph vorhanden Knoten- und Kantendatentypen zu,so dass Knoten und Kanten dieses Objekttyps durch den Aufruf virtueller Constructoren erzeugt werden können.

Ab der Version Delphi 2.0 ist eine visuelle Vererbung der Benutzeroberfläche TKnotenform möglich.Um diese Möglichkeit zu nutzen,muß der Constructor von TXGraph außerdem die Zeile Knotenform.Graph:=self enthalten,um der in der Benutzeroberfläche benutzten Graphinstanz (Feld von TKnotengraph) Graph_ bzw. der Property Graph die neue Graphenstruktur mitzuteilen.

Für den Fall der Vererbung der Benutzeroberfläche wird durch die Entwicklungsumgebung von Knotengraph in der Unit UForm die folgende Typdeklaration TKnotenformular=class(TKnotenform) vorgegeben:

(Die Entwicklungsumgebung von Delphi erzeugt schon automatisch einen Eintrag TKnotenform1=class(TKnotenform),der dann entsprechend umbenannt werden kann.)

```
TKnotenformular = class(TKnotenform)

    zusätzliche Menüeinträge:
    z.B.:
    NeueAnwendung: TMenuItem;
    Knotenumspeichern: TMenuItem;
    Knoteneingeben: TMenuItem;

    neue Methoden:
    z.B.:
    procedure FormActivate(Sender: TObject);
    sowie weitere Methoden

  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;
```

Mit der Instanzdeklaration

```
var
  Knotenformular:TKnotenformular;
  XGraph:TXGrapH;
```

hat FormActivate (Aktivierung der Form) dann folgenden Aufbau:

```
procedure TKnotenformular.FormActivate(Sender: TObject);
begin
  inherited;
  XGraph:=TXGraph.Create;
end;
```

Für den Fall,dass die Methode InhaltskopiedesGraphen zur Erzeugen eines neuen Graphen benutzt werden soll,ist der Constructor-Aufruf gemäß der eben beschriebenen Methode TKnotenformular.FormActivate nicht erforderlich.Mit der obigen Instanzdeklaration und den obigen Typdeklarationen ist lediglich vor Verwendung von XGraph folgender Aufruf notwendig,der die Graphenstruktur des vorhandenen Graphen vom Typ TInhaltsgraph in die neue Struktur konvertiert:

```
XGraph:=Graph.InhaltskopiedesGraphen(TXGraph,TXKnoten,TXKante,false)
```

(Graph:Ausgangsgraph vom Typ TInhaltsgraph)

Der letzte übergebende Boolesche Parameter gibt an,ob bei einem gerichteten Graph die Richtung aller Kanten beibehalten werden sollen (false) oder ob alle gerichteten Kanten in ungerichtete umgewandelt werden sollen (true).

Diese Erzeugung einer Inhaltskopie hat gegenüber der direkten Benutzung eines Graphen XGraph vom Typ TXGraph die Eigenschaft (Vorteil bzw. Nachteil?),dass nicht der neue Graph XGraph sondern der alte Graph Graph (nur mit dessen Datenfeldern) bei den Verwendung der Menüs Graph speichern und Graph lesen (als auch der Menüs Graph speichern unter und Graph hinzufügen) von TKnotenformular (die von TKnotenform an TKnotenformular vererbt werden) auf dem Datenträger gespeichert oder vom Datenträger eingelesen wird,weil die Menüalgorithmen dann nur mit Graph statt XGraph arbeiten.(Dies gilt dann auch für alle anderen durch Vererbung von TKnotenform an TKnotenformular übergebenden Menüs,so dass sich deren Algorithmen jeweils stets auf Graph statt XGraph beziehen:z.B. Knoten/Kanten löschen,Knoten/Kanten editieren usw.)

Bei der direkten Verwendung von XGraph vom Typ TXGraph (ohne eine Inhaltskopie zu benutzen,aber durch Erzeugung von XGraph mittels der Methode TXGraph.Create in der (oben beschriebenen) Methode TKnotenformular.FormActivate) werden dagegen auch alle Datenfelder des neuen Graphen XGraph bei Verwendung der beiden genannten Menüs Graph speichern und Graph lesen (als auch der Menüs Graph speichern unter und Graph hinzufügen) berücksichtigt.Auch die übrigen von TKnotenform an TKnotenformular visuell vererbten Menüs wie z.B. Knoten/Kanten löschen,Knoten/Kanten editieren usw. arbeiten dann automatisch mit den Neuen Objekten vom Typ TXKnoten,TXKante bzw. TXGraph.

Allgemein steht damit mit TXGraph ein Graph-Objekttyp zur Verfügung,der als Knoten Knoten vom Typ TXKnoten und als Kanten Kanten vom Typ TXKante enthält,so dass die neuen Knoten,Kanten und der neue Graph alle Eigenschaften von TInhaltsknoten,TInhaltskante und TInhaltsgraph erben und zusätzlich über ihre neuen Datenfelder und Methoden verfügen.

Die von TKnotenform an TKnotenformular vererbten Menü-Methoden arbeiten dann automatisch mit der neuen Graphenstruktur.Neu in Knotenformular definierte Menüs können dann darüberhinaus die neuen Methoden (und dadurch indirekt auch die neuen Datenfelder) des Graphen XGraph bzw. dessen neue Knoten und Kanten in ihrem Quellcode zur Programmierung der für den neuen Graphen spezifischen Algorithmen verwenden.

Die Verwendung des beschriebenen Graph-Objekttyps wird im Programm Knotengraph bei der Anwendung Endlicher Automat demonstriert.Dazu wird zunächst TAutomatenneugraph als Nachfolgerobjekt von TAutomatengraph in der Unit UForm definiert:

```
TAutomatenneugraph=class(TAutomatengraph)
   constructor Create;override;
 end;
```

Mit der Constructor-Methode:

```
Constructor TAutomatenneugraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TAutomatenknoten;
  Inhaltskanteclass:=TInhaltskante;
  Knotenformular.Graph:=self;
end;
```

Dabei wird außer auf TInhaltskante auf folgende Objekte zurückgriffen:

```
TKnotenart = 0..3;

TAutomatenknoten = class(TInhaltsknoten)
private
  Knotenart_:TKnotenart;
  procedure SetzeKnotenart(Ka:TKnotenart);
  function WelcheKnotenart:TKnotenart;
public
  constructor Create;override;
  property KnotenArt:TKnotenart read WelcheKnotenart write SetzeKnotenart;
  function Wertlisteschreiben:TStringlist;override;
  procedure Wertlistelesen;override;
end;
```

```
TAutomatengraph = class(TInhaltsgraph)
private
  MomentanerKnoten_:TAutomatenknoten;
  procedure SetzeMomentanenKnoten(Kno:TAutomatenknoten);
  function WelcherMomentaneKnoten:TAutomatenknoten;
public
  constructor Create;override;
  property MomentanerKnoten:TAutomatenknoten read WelchermomentaneKnoten
  write SetzeMomentanenKnoten;
  function Wertlisteschreiben:TStringlist;override;{entbehrlich}
  procedure Wertlistelesen;override;{entbehrlich}
  procedure SetzeAnfangsWertKnotenart;
  function BestimmeAnfangsknoten:TAutomatenknoten;
end;
```

Es werden in der Unit folgende globale Variablen definiert:

```
var
  Knotenformular: TKnotenformular;
  Automatenneugraph:TAutomatenneugraph;
  Automatenneugraphaktiv:Boolean;
```

TKnotenformular stellt dabei das Nachfolgerobjekt von TKnotenform mittels visueller Vererbung dar:

```
TKnotenformular = class(TKnotenform)
  .
  .
  .
end;
```

Durch den Constructor-Aufruf

Automatenneugraph:=TAutomatenneugraph.Create wird jetzt die Instanz eines Graphen erzeugt,auf den die Menü-Ereignismethoden der Form Knotenform,die an die Hauptform Knotenformular visuell vererbt werden,anwendbar sind.

Dadurch ist es jetzt beispielsweise möglich,Zwischenzustände des Graphen bzw. Automaten,die sich bei der Ausführung des Algorithmus ergeben,abzuspeichern und später wieder einzulesen,um den Ablauf an dieser Stelle wieder fortzusetzen.

Der Quellcode ist in der Procedure AutomatmitTautomatenneugraph in der Menü-Methode TKnotenformular.EndlicherAutomatClick(Sender: TObject) enthalten:

```
Procedure AutomatmitTAutomatenneugraph;
label Endproc;
var NeuerGraph:Boolean;
begin
  if not Automatenneugraphaktiv then
  begin
    Menuenabled(false);
    Graphspeichern.enabled:=true;
    Graphspeichernunter.enabled:=true;
    Automatenneugraphaktiv:=true;
    if MessageDlg('Soll ein neuer Graph erzeugt werden?',
      mtConfirmation,[mbYes,mbNo],0)=mrYes then
    begin
      if not Graph.Graphistgespeichert
      then
        if MessageDlg('Momentaner Graph ist nicht gespeichet! Speichern?',
          mtConfirmation, [mbYes, mbNo], 0) = mrYes
        then
          GraphspeichernunterClick(Sender);
      NeuerGraph:=true;
      Automatenneugraph:=TAutomatenneugraph.Create;                                      1)
      Bildloeschen;
      Showmessage('Neuen Graph erzeugen.Danach Button Start anwählen!');
      Button.visible:=true;
      Button.Caption:='Start';
      Menuenabled(true);
      Mainmenu.Items[5].enabled:=false;
      Mainmenu.Items[6].enabled:=false;
      exit;
    end
    else  if MessageDlg('Soll ein Graph geladen werden?',
      mtConfirmation,[mbYes,mbNo],0)=mrYes
```

```
  then
  begin
    if not Graph.Graphistgespeichert
    then
      if MessageDlg('Momentaner Graph ist nicht gespeichet! Speichern?',
        mtConfirmation, [mbYes, mbNo], 0) = mrYes
        then
          GraphspeichernunterClick(Sender);
      Automatenneugraph:=TAutomatenneugraph.Create;                               2a)
      Knotenformular.GraphladenClick(Sender);                                     2b)
      NeuerGraph:=false;
      Bildloeschen;
      AutomatenNeuGraph.zeichneGraph(Paintbox.Canvas);
  end
  else
  begin
    NeuerGraph:=true;
    Bildloeschen;
    Automatenneugraph:=TAutomatenneugraph(Graph.InhaltskopiedesGraphen
    (TAutomatenneugraph,TAutomatenknoten,TInhaltskante,false));                   3)
    Showmessage('Der Graph kann noch verändert werden.Danach Button Start anwählen!');
    Button.visible:=true;
    Button.Caption:='Start';
    Menuenabled(true);
    Mainmenu.Items[5].enabled:=false;
    Mainmenu.Items[6].enabled:=false;
    exit;
  end;
end;
if  (Automatenneugraphaktiv) then
if ((not NeuerGraph)and(MessageDlg('Fortsetzung der Zustandsfolge?               4)
    (sonst          Neubeginn!)',mtConfirmation,[mbYes,mbNo],0)=mrno))
  or (NeuerGraph)
then
begin                                                                             5)
  Button.Caption:='Abbruch';
  Automatenneugraph.FaerbeGraph(clblack,pssolid);
  if Automatenneugraph.Leer then exit;
  if Automatenneugraph.AnzahlKomponenten>1 then
  begin
    ShowMessage('Mehrere Komponenten!');
    exit;
  end;
  Menuenabled(false);
  Graphspeichern.enabled:=true;
  Graphspeichernunter.enabled:=true;
  Bildloeschen;
  Automatenneugraph.ZeichneGraph(Paintbox.Canvas);
  if Automatenneugraph.AnzahlungerichteteKanten>0 then
  begin
    ShowMessage('Der Graph hat ungerichtete Kanten!');
    Menuenabled(true);
    exit;
  end;
  if Automatenneugraph.AnzahlKnoten<2 then
  begin
    ShowMessage('Graph hat nur einen Knoten!');
    Menuenabled(true);
    exit;
  end;
  Automatenneugraph.SetzeAnfangswertknotenart;
  GraphH:=Automatenneugraph;
  GraphH.Zustand:=false;
  ShowMessage('Anfangszustand mit Mausklick bestimmen');
  Paintbox.ONmousemove:=nil;
  Paintbox.OnDblclick:=niL;
  Paintbox.OnMouseDown:=Automaten1MouseDown;
  repeat
    Application.Processmessages;
    if GraphH.Abbruch then goto Endproc;
  until GraphH.Zustand;
  repeat
    Ende:=false;
    GraphH.Zustand:=false;
    ShowMessage('Endzustand mit Mausklick bestimmen');
    Paintbox.OnMouseDown:=Automaten2MouseDown;
    repeat
      Application.Processmessages;
      if GraphH.Abbruch then goto Endproc;
    until GraphH.Zustand;
    if MessageDlg('Alle Endzustände markiert',
      mtConfirmation, [mbYes, mbNo], 0) = mrYes then Ende:=true
  until Ende;
```

```
    Automatenneugraph.MomentanerKnoten:=Automatenneugraph.BestimmeAnfangsknoten;
    Automatenneugraph.ZeichneGraph(Paintbox.Canvas);
  end;
  Button.Caption:='Abbruch';                                                    6)
  Eingabe.Visible:=true;
  Button.Visible:=true;
  Ausgabe1.Caption:='Kante eingeben:';
  Automatenneugraph.MomentaneKantenliste:=TKantenliste.Create;;
  Eingabe.Setfocus;
  Endproc:
  GraphH:=Automatenneugraph;
  Aktiv:=false;
  if GraphH.Abbruch then
  begin
    Bildloeschen;
    Aktiv:=true;
    Automatenneugraph.ZeichneGraph(Paintbox.Canvas);
    GraphH:=nil;
    ShowMessage('Abbruch!');
  end;
end;
```

Entweder wird bei 1) ein neuer Graph erzeugt,bei 2) ein Graph geladen oder bei 3) der vorhandene Graph in einen Graph des Typs TAutomatenneugraph umgewandelt.

Die Erzeugung des neuen Graphen erfolgt bei 1) durch direkten Constructor-Aufruf,bei 2) wird noch zusätzlich zum Constructor-Aufruf (2a) bei 2b) die Menü-Ereignismethode Graphladen des Formulars Knotenformulars,die von Knotenform vererbt wurde,aufgerufen,die nach dem Constructor-Aufruf einen Graphen vom Typ TAutomatenneugraph lädt.Die Umwandlung des vorhandenen Graphs vom Typ TInhaltsgraph erfolgt bei 3) mittels der Methode InhaltskopiedesGraphen,die mittels virtueller Constructor-Aufrufe ebenfalls einen Graph vom Typ TAutomatengraph erzeugt.In allen Fällen sind die im Graph vorhandenen oder zu erzeugenden Knoten vom Typ TAutomatenknoten und die Kanten vom Typ TInhaltskante.

Alle von TKnotenform an TKnotenformular visuell vererbten Methoden und Datenfelder in Form der Komponenten sind mit dem neuen Graphtyp verwendbar,d.h. die gesamte bereitgestellte Benutzeroberfläche ist zu benutzen und wird jetzt um eine Menü-Methode EndlicherAutomat erweitert.

Bei 4) kann gewählt werden,ob einem Graph das Durchlaufen einer Zustandfolge bis zum Endzustand vom abgespeicherten Knoten aus fortgesetzt werden soll,der vorher in einem bestimmten Zustand abgespeichert wurde.Der Algorithmus wird dann bei 6) mit der Übergabe des Programmablaufs an die Eingabe-Komponente fortgeführt.Bei 5) wird dagegen zuerst das Bestimmen eines neuen Anfangszustandknotens und neuer Endzustandsknoten durchgeführt.

```
procedure TKnotenformular.ButtonClick(Sender: TObject);
begin
  try
    if Automatenneugraphaktiv
    then
    begin
      if Button.Caption='Abbruch' then                                          7)
      begin
        Automatenneugraphaktiv:=false;
        Knotenformular.Graph:=Automatenneugraph.InhaltskopiedesGraphen(
        TInhaltsgraph,TInhaltsknoten,TInhaltskante,false);
        Knotenformular.GraphH:=Knotenformular.Graph;
        Knotenformular.Graph.FaerbeGraph(Clblack,pssolid);
        Mainmenu.Items[5].Enabled:=true;
        Mainmenu.Items[6].Enabled:=true;
        Automatenneugraph.Freeall;
      end;
      if Button.Caption='Start' then                                            8)
      begin
        Button.Caption:='Abbruch';
        EndlicherAutomatClick(Sender);
        exit;
      end
    end;
    Eingabe.Visible:=false;
    Button.Visible:=false;
    Ausgabe1.Caption:='';
```

```
    Ausgabe1.Refresh;
    Ausgabe2.Caption:='';
    Ausgabe2.Refresh;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Aktiv:=true;
    Menuenabled(true);
  except
    BildzeichnenClick(Sender);
    AbbruchClick(Sender);
    ShowMessage('Fehler');
  end;
end;
```

In der Button-Methode Button-Click wird,falls der Automatenneugraph verwendet wird,was durch die globale Boolesche Variable Autoamtenneugraphaktiv angezeigt wird,durch Mausklick auf diesen Button mit der Bezeichnung (Caption) Start,der Algorithmus des EndlicheAutomat-Menüs erneut gestartet(7).Zuvor kann ein Graph (vom Typ TAutomatenneugraph) entweder neu erzeugt,geladen oder verändert werden.Während des Durchlaufens einer Zustandsfolge kann der Graph dann jederzeit im momentanen Zustand gespeichert werden.Durch Mausklick auf den Button,dessen Bezeichnung jetzt Abbruch ist (8),wird der Algorithmus gestoppt,der Graph vom Typ TAutomatenneugraph in einen Graph vom Typ TInhaltsgraph zurückverwandelt und anschliessend aus dem Speicher gelöscht.Dasselbe gilt,wenn der Endzustand erreicht ist,und der Algorithmus nicht weiter fortgesetzt wird (9).(Methode EingabeKeypress:hier heißt der Graph Automatenneugraph der Methode EndlicherAutomat wieder Automatengraph und umfaßt so auch den in Kapitel C VII dargestellten Algorithmus.)

```
procedure TKnotenformular.EingabeKeyPress(Sender: TObject; var Key: Char);
.
.
begin
 .
 .
  Automatengraph:=TAutomatengraph(GraphH);
   .
   .
  if Automatenneugraphaktiv                                                        9)
  then
  begin
    Automatenneugraphaktiv:=false;
    Bildloeschen;
    Knotenformular.Graph:=Automatenneugraph.InhaltskopiedesGraphen(TInhaltsgraph,
      TInhaltsknoten,TInhaltskante,false);
    Knotenformular.Graph.FaerbeGraph(clblack,pssolid);
    Knotenformular.Graph.ZeichneGraph(Paintbox.Canvas);
    Mainmenu.Items[5].Enabled:=true;
    Mainmenu.Items[6].Enabled:=true;
  end;
.
.
  GraphH:=Automatengraph.InhaltskopiedesGraphen(TInhaltsgraph,TAutomatenknoten,
    TInhalt  sKante,false);
   Automatengraph.Freeall;
.
.
end;
```

Im übrigen läuft der Algorithmus gemäß dem im Kapitel C VII „Graph als endlicher Automat“ beschriebenen Verfahren ab.

Das dargestellte Verfahren lässt sich natürlich nicht nur bei der Anwendung Endlicher Automat sondern bei jeder der im Menü Anwendungen enthaltenden Algorithmen einsetzen.(Das Menü Pfade dagegen benötigt keinen neuen Graph sondern arbeitet stets mit TInhaltsgraph.)

Der Vorteil liegt darin,dass der so definierte und erzeugte Graph alle Möglichkeiten der vererbten Oberfläche benutzen kann.Der Nachteil liegt darin,dass für jede Anwendung ein neuer Graphtyp definiert und mit einer eigenen globalen Instanzvariablen versehen werden muß.Diese Variablen könnten natürlich auch wieder Datenfelder von TKnotenformular (wie auch die Boolesche Variable Automatenneugraphaktiv) sein.Hierauf wurde in der Unit UForm verzichtet,damit der neue Graphtyp sofort auffällt.)

Die abgespeicherten Graphtypen sind dann aber alle inkompatibel zueinander,da sie unterschiedliche Datenfelder besitzen,und man muß beim Laden die Art des Graphen genau beachten.

Daher wurde in dem Programm Knotengraph bei allen anderen Anwendungen stets die Methode benutzt,dass der neue Graphtyp nur innerhalb der zu ihm gehörenden Methoden des Formulars als lokale Variable (Instanz) definiert wird,und die visuell vererbte Benutzeroberfläche,die durch TKnotenform definiert wird,ansonsten mit dem Grundtyp TInhaltsgraph,mit dem Knotentyp TInhaltsknoten und dem Kantentyp TInhaltskante arbeitet.Nur in der entsprechenden Anwendung wird der Graph dann mittels der Methode InhaltskopiedesGraphen in den entsprechenden Graphtypen mit den entsprechenden Knoten-und Kantentypen umgewandelt.So kann dann stets ein einheitlicher Graphtyp,nämlich TInhaltsgraph mit den Knoten und Kanten vom Typ TInhaltsknoten und TInhaltskante geladen und gespeichert werden.

Sollen dagegen nur Anwendungen mit nur **einem** neuen Graphtypen erstellt werden,ist das globale Erzeugen einer Instanz des neuen Graphtypen z.B. als (private) Datenfeld von TKnotenformular (unter Zugriff durch entsprechende Propertys) der zu bevorzugende Weg.

Der im Kapitel B III beschrieben Unterrichtsgang wurde von mir mehrfach in Grundkursen der Jahrgangsstufen 11,12 und 13 durchgeführt.Die dabei bisher in diesem Kapitel beschrieben Datenstruktur des Graphen stand dabei nicht von vornherein fest,sondern ist das Ergebnis immer weiterer Verbesserungen,die sich auf Grund der Unterrichtserfahrungen ergab.

Ausgangspunkt dabei war zunächst eine Datenstruktur eines Graphen,wie sie im Buch Listen,Bäume und Graphen als Objekte (Lit 51) von D. Schaerer beschrieben ist.

Im Gegensatz zu der dem Programm Knotengraph zu Grunde liegenden Struktur eines Graphen als zusammenfassendes Objekt einer Knoten-und Kantenliste

```
type

TGraph=class(TObject)
 .
 Knotenliste_:TKnotenliste;
 Kantenliste_:TKantenliste;
 .
end;

mit

TKnotenliste=class(TList)
             .
             end;

TKantenliste=class(TList)
             .
             end;
```

ist dort der Graph gleichbedeutend mit (nur) einer Knotenliste:

```
type Graph=NodeList;

mit

type Nodelist=object(List)
     .
     end;
```

(object ist hier gleichbedeutend mit class)

Eine im direkten Zugriff des Graphen befindliche Kantenliste existiert nicht,stattdessen wird jedem Knoten eine Liste der zu ihm gehörenden Kanten zugeordnet,ähnlich der Liste der ein-und auslaufenden Kanten beim Programm Knotengraph.Allerdings werden hier ein-und ausgehende Kanten in nur einer Liste gemeinsam verwaltet.

Statt der ein-und ausgehende Kantenliste im Programm Knotengraph

```
type TKnoten=class(TKnotenliste)
        .
        EingehendeKantenliste_:TKantenliste;
        AusgehendeKantenliste_:TKantenliste;
        .
      end;
```

wird definiert:

```
type EdgeList=object(TList)
       .
      end;

Node=EdgeList;

EdgeEnd=object(Edgelist)
           .
           Outgoing:Boolean;
           there:Node;
           .
         end;
```

Das heißt,dass ein Knoten eine andere Bezeichnung für die zu ihm gehörende Kantenliste ist,wobei die Elemente dieser Kantenliste allerdings nicht die Kanten selber sondern nur die Kantenenden (Edgeend) bedeuten.

Die Unterscheidung zwischen ein-und auslaufenden Kantenenden,die sich gemeinsam in der Liste des Objekts Node befinden,wird über die Boolesche Variable Outgoing (true/false) vorgenommen.

Eine Kante besteht nun aus zwei Kantenenden (ein-und auslaufend,die sich jeweils in den Listen der jeweiligen Anfangs-und Endknoten befinden und enthalten zusätzlich einen Kantenkörper Edgebody zur Aufnahme eines Kanteninhalts

mit der Definition:

```
type Edgebody=Thing;
```

(Thing ist gleichbedeutend mit TObject.)

Die beiden Kantenenden Edgeend und der Kantenkörper Edgebody werden durch wechselseitige Zeiger aufeinander miteinander verknüpft (2 Zeiger von den Kantenenden auf den Kantenkörper und 2 Zeiger zur jeweiligen Verkettung der beiden Kantenenden miteinander).Außerdem werden zusätzlich von den beiden Kantenenden Zeiger auf den Knoten Node (mittels there) gesetzt,um von der Kante aus auf den Anfangs-und Endknoten zugreifen zu können.

Insgesamt sind also 6 Zeiger zu jeder Kante zu verwalten.

Ein direkter Zugriff von der Graphenstruktur auf die Kanten ist nicht möglich,sondern der Zugriff erfolgt stets über die Anfangs-und Endknoten.

Im Gegensatz dazu bietet die Kantendeklaration des Programms Knotengraph

```
type TKante =  class(TKantenliste)
        .
        Anfangsknoten_:TKnoten;
        Endknoten_:TKnoten;
        .
      end;
```

mittels der in TKantenliste vorhandenen Deklaration

```
type TKantenliste=class(TListe)
     .
     function Kante(Index:Integer):TKante;
     property Items[Index:Integer]:TKante read Kante;
      .
     end;
```

jederzeit vom Graph aus direkten indizierten Zugriff auf jede Kante:

Graph.Kantenliste.Kante(Index)

Die Verknüpfung mit dem Anfangs-und Endknoten erfolgt hier einerseits durch Einfügen in die ein-und ausgehenden Kantenlisten dieser Knoten und andererseits direkt durch die Felder Anfangsknoten_ und Endknoten_ des Objekts TKante.

Der Unterschied zu der von Schaerer vorgenommen Deklaration besteht also darin,dass im Programm Knotengraph eine (gerichtete) Kante ein einziges Objekt ist,das sich gleichzeitig in drei verschiedenen Kantenlisten,nämlich der Kantenliste des Graphen und den ein-bzw. ausgehenden Kantenlisten des Anfangs-und Endknoten dieser Kante befindet,während nach der anderen Konzeption eine Kante ein dreigeteiltes Objekt bestehend aus zwei Kantenenden und einem Kantenkörper ist,dessen Teile durch Zeiger miteinander verknüpft sind,wobei die Kantenenden in einer einzigen Liste,die auch als Knoten bezeichnet wird,verwaltet werden.

Die Verwaltung der Kanten gemäß dieser Konzeption der Dreiteilung erwies sich als didaktisch-methodisch schwer vermittelbar.Vielmehr ist gemäß der zeichnerischen Darstellung eines Graphen die Vorstellung der Kante als ein einziges Objekt viel näherliegend,weil sich auch hieraus viel leichter die Vorstellung ableitet,dass ein Graph aus den beiden Objekten Knoten und Kanten zusammensetzt.

Außerdem werden die Verwaltungsalgorithemen wie z.B. Erzeugen bzw. Löschen einer Kante auf Grund der Konzeption der Dreiteilung wegen der Zeigerverwaltung bedeutend aufwendiger und sind deshalb wenig motivierend.

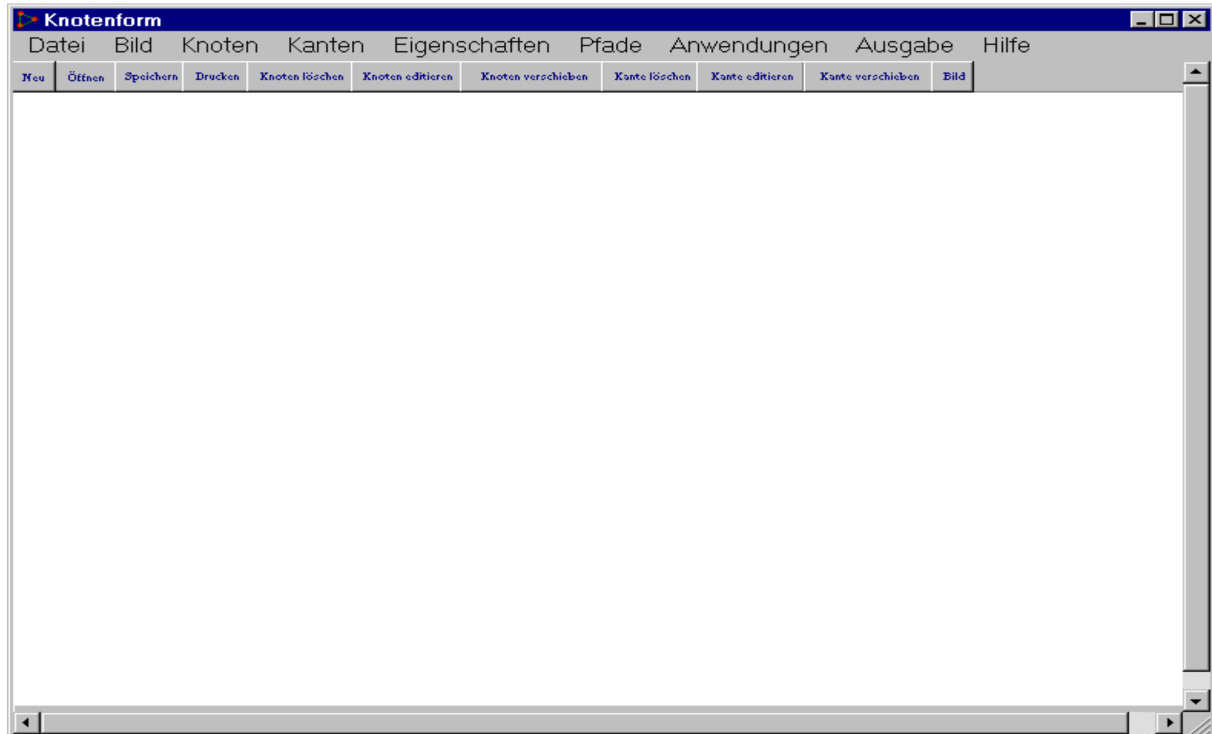
Die Algorithmen werden darüberhinaus dadurch erschwert,dass das Verwalten der ein-und ausgehenden Kantenenden in nur einer Liste erfolgt,so dass entsprechende Suchvorgänge durchgeführt werden müssen.

Die im Programm Knotengraph verwendete Objektstruktur ergab sich deshalb nach und nach im Unterricht auf Grund der negativen Erfahrungen mit der ursprünglichen Konzeption und dem Streben nach Vereinfachung der Algorithmen sowie einer möglichst wirklichkeitsgetreuen Modellierung und Nachbildung einer zeichnerisch Graphenstruktur.

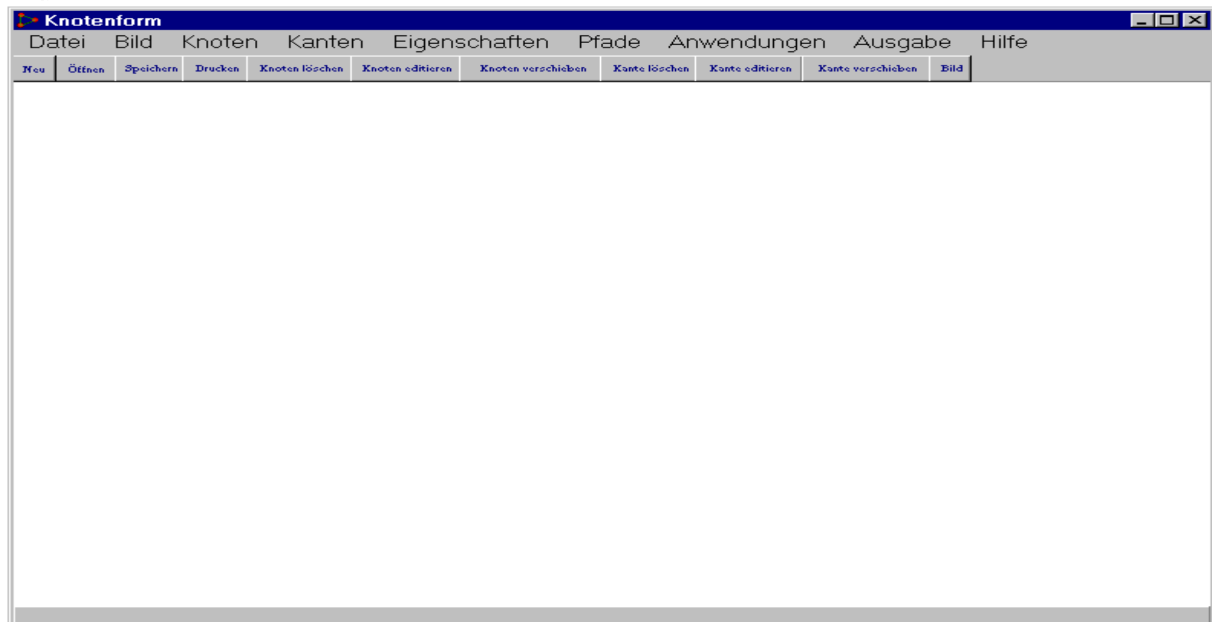
F II Die Bedienung des Programms Knotengraph

Das Programm wird unter Windows durch Starten des Files KProjekt.exe gestartet.Danach erscheint die folgende **Benutzeroberfläche:**

Als Fenster mit Scrollleisten:



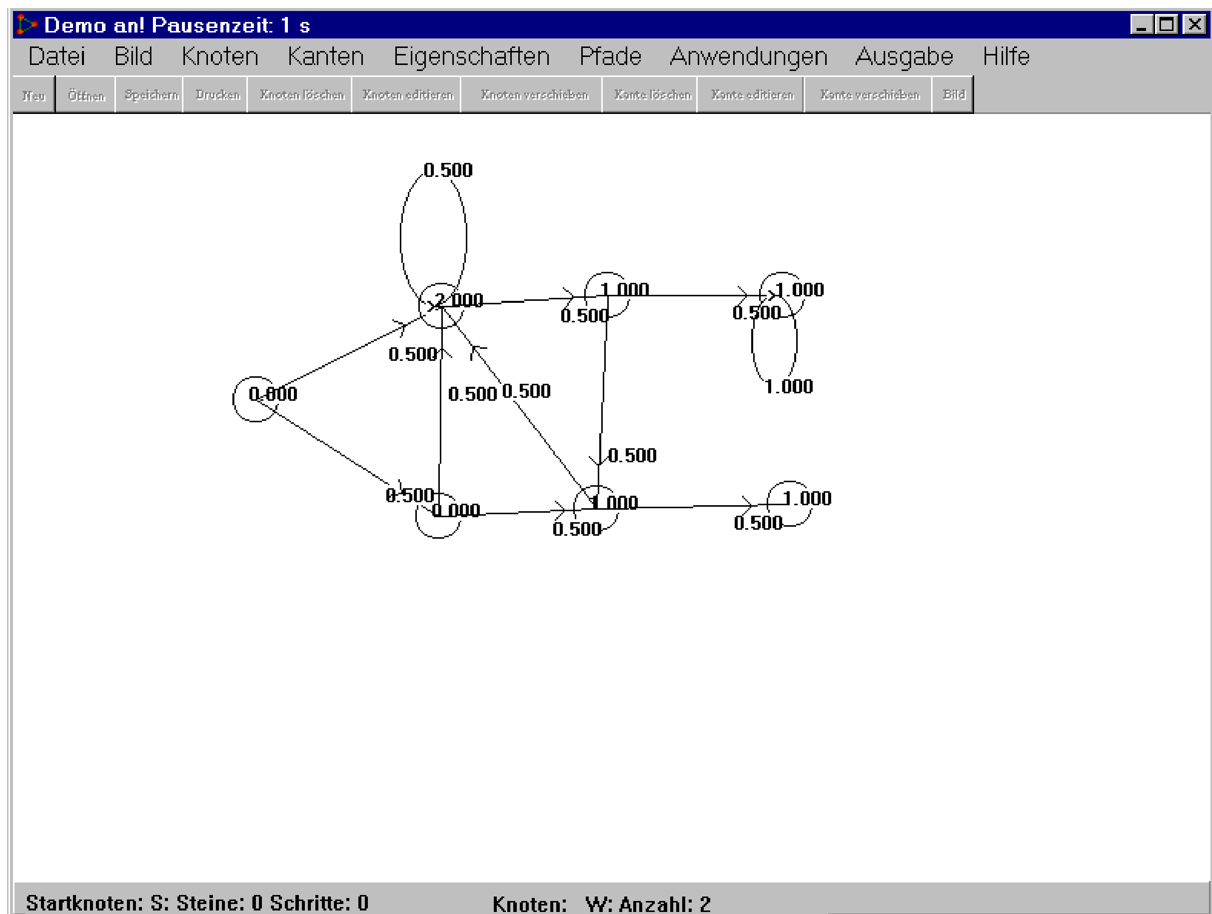
Als Vollbild:



Das Hauptanwendungsfenster weist eine Dreiteilung auf.

Der große Mittelbereich ist die Zeichenfläche (Paintbox),auf der ein Graph mit Knoten und Kanten sowie deren Inhalten dargestellt wird.Am oberen Rand befindet sich die Menü-und Menü-Button-Leiste und am unteren Rand eine Panelfläche,die zwei Ausgabelabel enthält,in denen Ergebnisse von Algorithmen dargestellt werden können:

Hauptanwendungsfenster (Form:Knotenform) mit Menüleiste,Button-Menü-Leiste (deaktiviert),Zeichenfläche (Paintbox) und Panelfläche mit den zwei Ausgabelabeln:



Die Knoten des Graphen werden durch Kreise dargestellt,in deren Fläche ein möglicher Knoteninhalt erscheint.Die Kanten des Graphen sind Verbindungslinien zwischen den Mittelpunkten zweier mit ihnen inzidenter Knoten,nämlich dem Anfangs-und Endknoten.Bei gerichteten Kanten sind die Linien mit einem Pfeil (>) versehen,der die Kantenrichtung beschreibt.Die Kantenlinien können entweder eine gerade Strecke zwischen den Mittelpunkten zweier Knotenkreise sein.

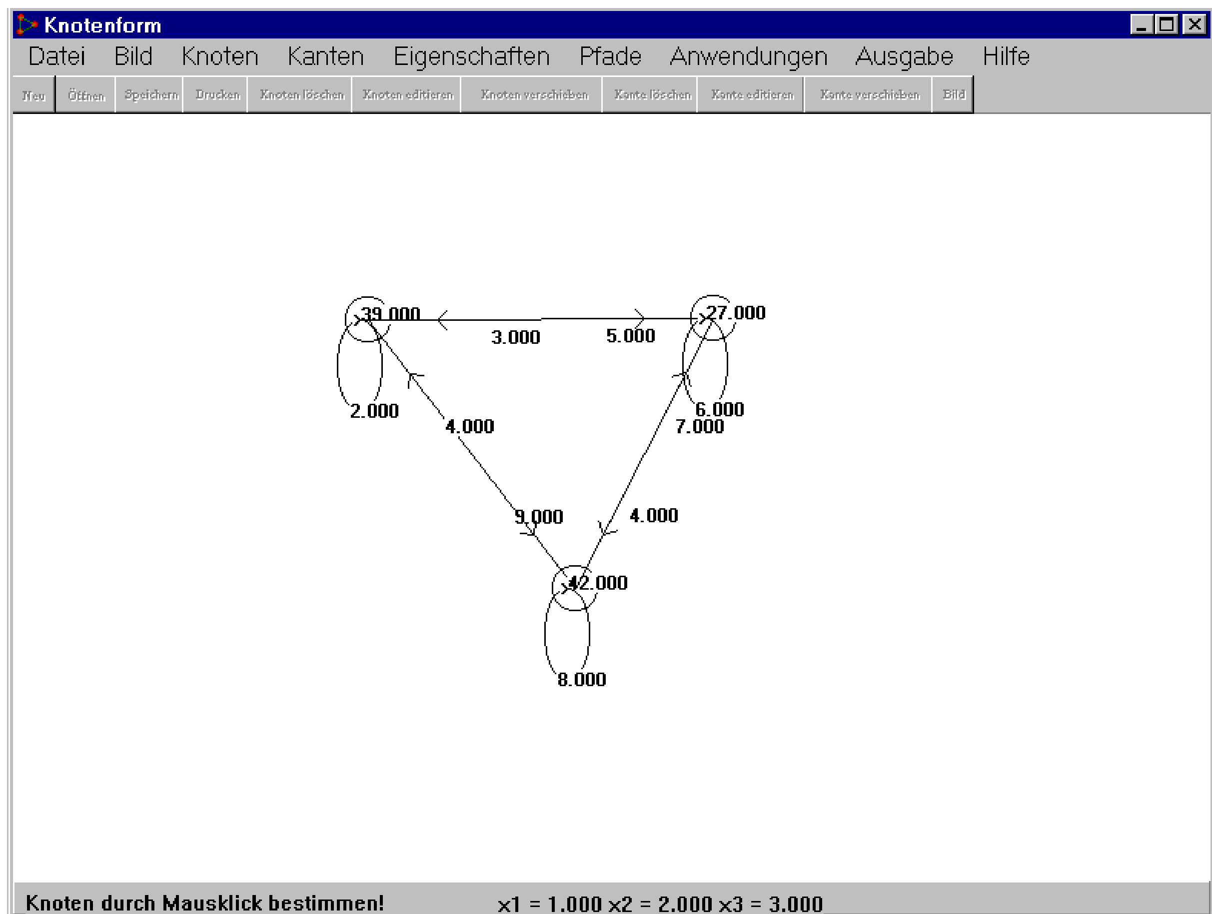
Andererseits ist es möglich als Kante einen Polygonzug,bestehend aus zwei Strecken zu erzeugen,deren Verbindungspunkt aus der Mitte der Strecke zwischen dem Start-und Endknoten senkrecht zu dieser Strecke beliebig weit hinausgeschoben werden kann (Weite der Kante ungleich Null).Auf diese Weise können beliebig viele Kanten zwischen zwei Knoten erzeugt werden,die auf der Zeichenoberfläche auch als verschiedene Kanten und nicht übereinanderliegend dargestellt werden.In der Nähe des Polygonzuges bzw. der Strecke der Kante wird ein möglicher Kanteninhalt gezeigt.Durch einen Klick mit der linken Maustaste auf einen Knoten werden dessen Inhalt (Wert) und dessen Koordinaten angezeigt.In gleicher Weise erscheint bei einem Klick mit der linken Maustaste auf eine Kante ein Fenster,in dem der Kanteninhalt (Wert) sowie der Anfangs-und Endknoteninhalt angegeben werden.

(Die Lage der Pfeile der gerichteten Schlingen befinden sich auf den Bildern stets im Knotenmittelpunkt,bei der Darstellung im Programm Knotengraph dagegen an den linken oder rechten Seite des Kreises der Schlinge.)

Anzeigefenster der Knoten-und Kanteninformationen durch Klick mit der linken Maustaste:

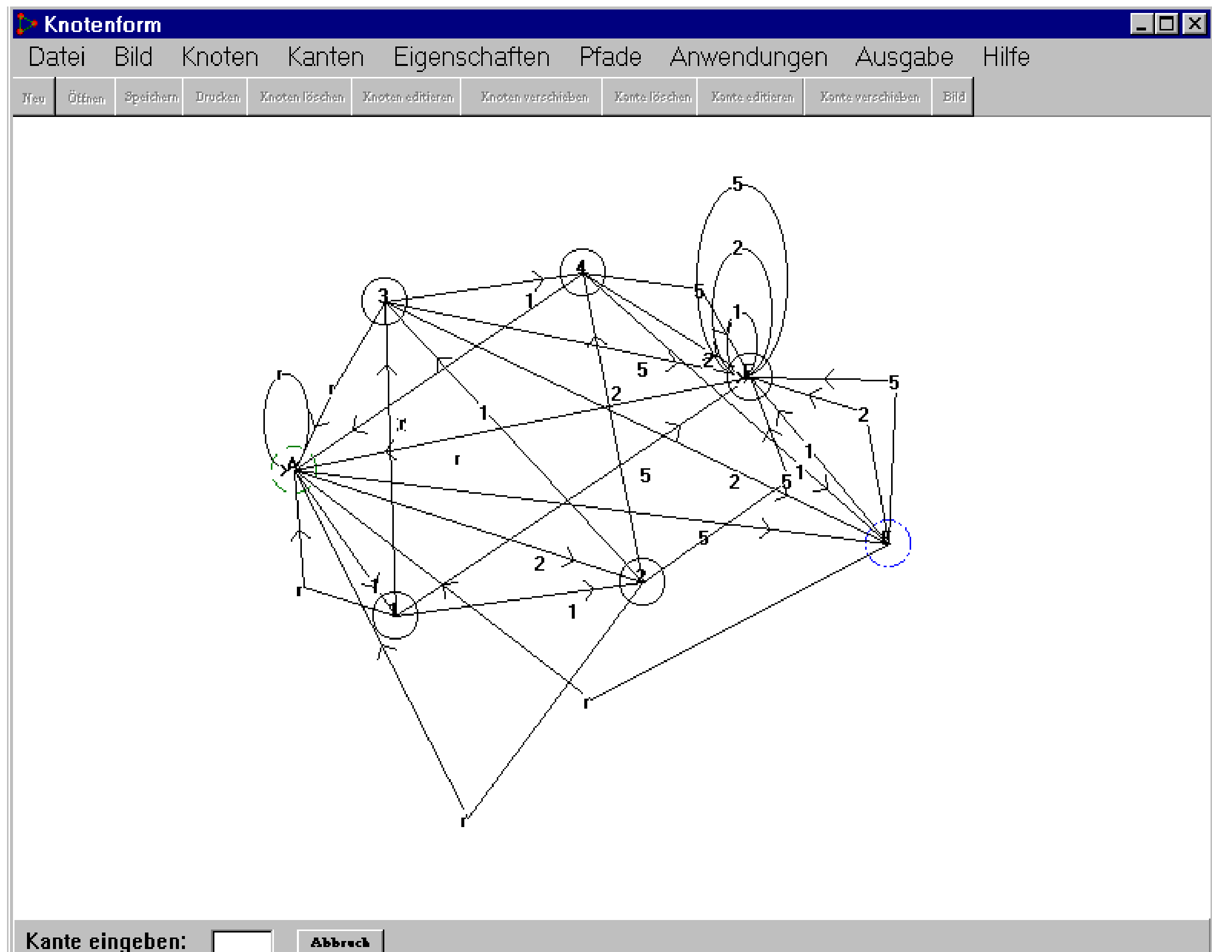
Als weitere Anzeigemöglichkeit sind,wie schon oben erwähnt,in der unteren Panelfläche jeweils links und rechts zwei Label Ausgabe 1 und 2 untergebracht,in denen Texte (Erläuterungen und Informationen) ausgegeben werden können.

Die beiden Ausgabelabel in der Panelfläche mit Textanzeige:



In dem Bereich des linken Ausgabelabels befindet sich zusätzlich (normalerweise unsichtbar) ein Editierfensterelement (,das bei der Anwendung Endlicher Automat sichtbar gemacht und benötigt wird).In dem Element können Werte für Kanteninhalte (Zeichen des Übergangsalphabets) eingegeben werden,um den Zustand zu wechseln.Außerdem erscheint dort zusätzlich ein Abbruch-Button.

Editierelement und Abbruch-Button:



Ergebnisse lassen sich mit Hilfe eines Anzeigefensters am unteren Rand der Paintbox in Form einer Liste anzeigen.

Bei Einfachklick auf die Panelfläche erscheint von dem Anzeigefenster nur eine Zeile mit der Möglichkeit (falls erforderlich) mittels eines Rollbalkens zu scrollen,so dass man alle Ergebnisse nacheinander anzeigen kann,ohne dass der Graph verdeckt wird.

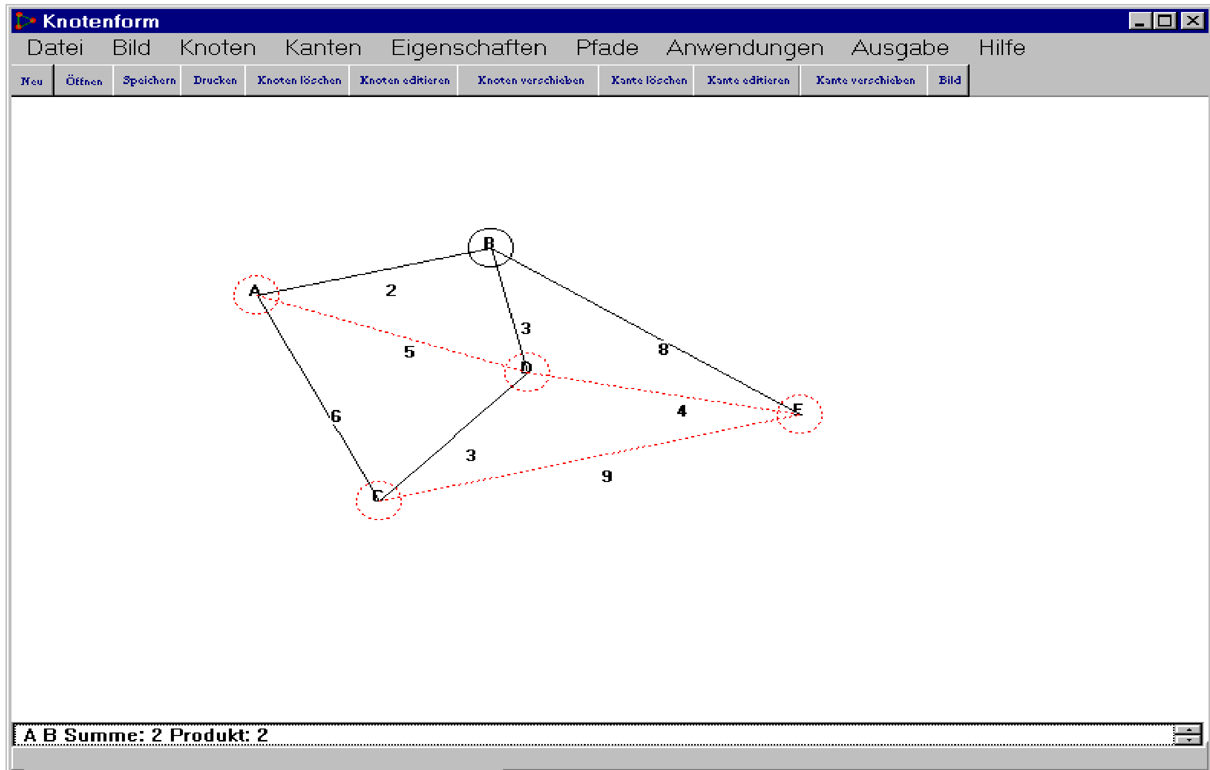
Beim Doppelklick auf die Panelfläche überdeckt das Anzeigefenster (falls erforderlich mit Scrollbar) teilweise oder ganz die Zeichenfläche.

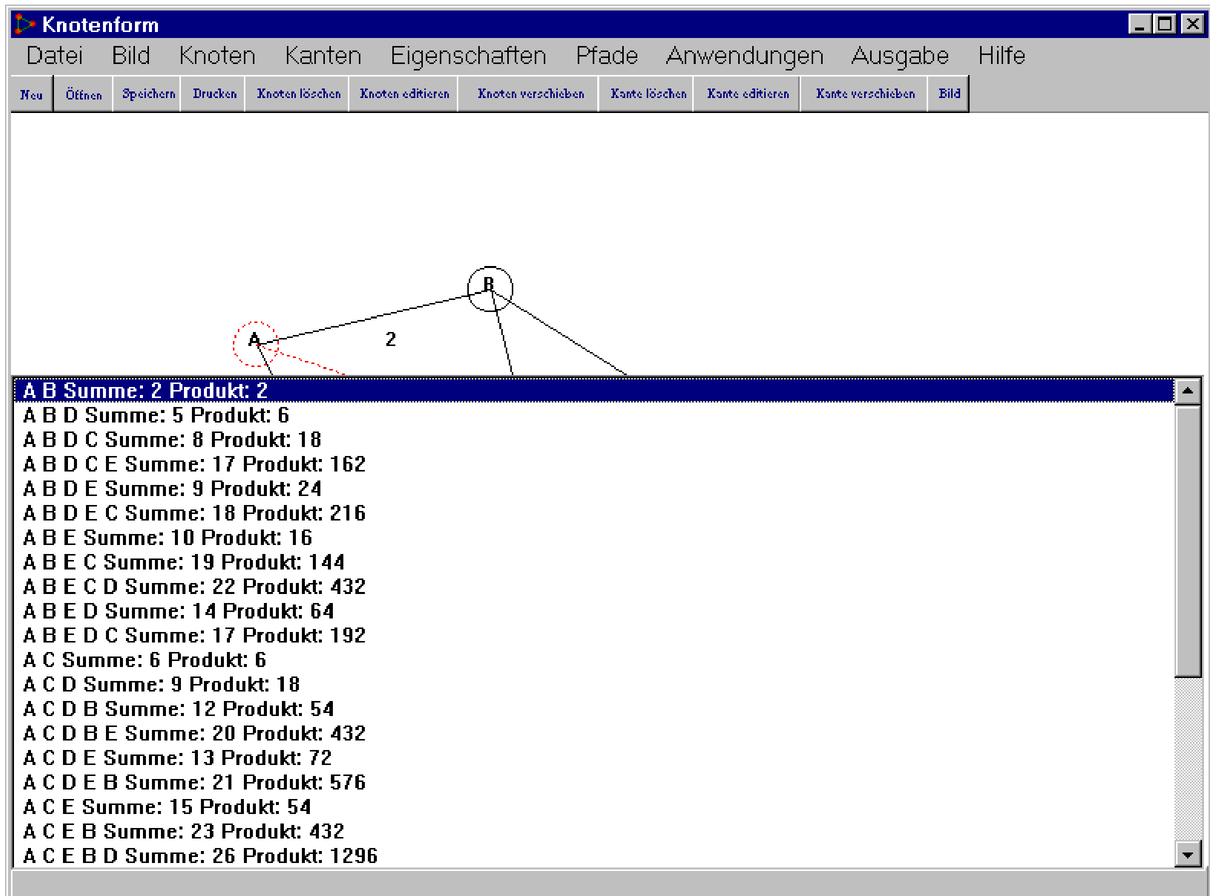
Wenn man auf die rechte Hälfte der Panelfläche klickt,vergrößert sich das Anzeigefenster auf die Hälfte der Zeichenfläche,doppelklickt man auf die linke Hälfte,wird die gesamte Zeichenfläche eingenommen.

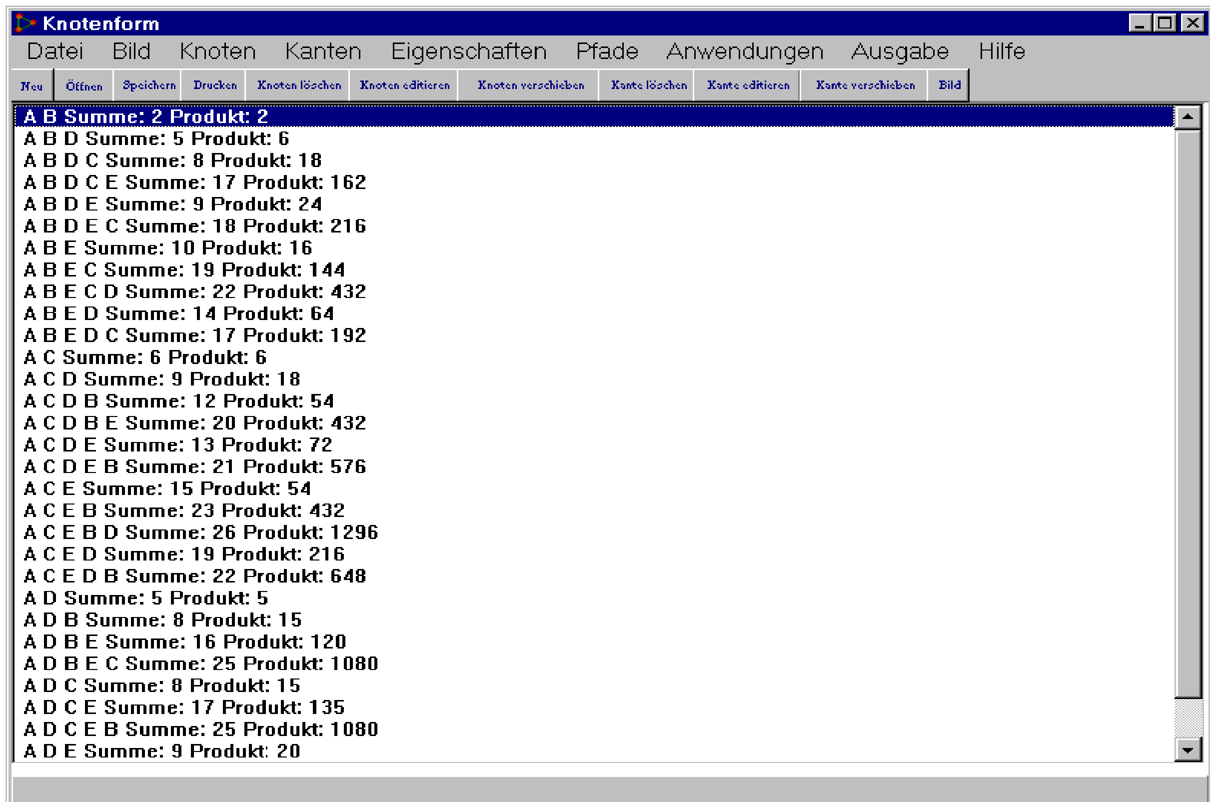
Durch einen Einfachklick mit der rechten Maustaste auf die Panelfläche wird das Anzeigefenster geschlossen.

Der Vorteil des Anzeigefensters ist,dass sich auf diese Weise gleichzeitig mit dem Graphen die Ergebnisse von Algorithmen in einer Zeile am unteren Rand,die gescrollt werden kann,sichtbar machen lassen.

Das Anzeigefenster:

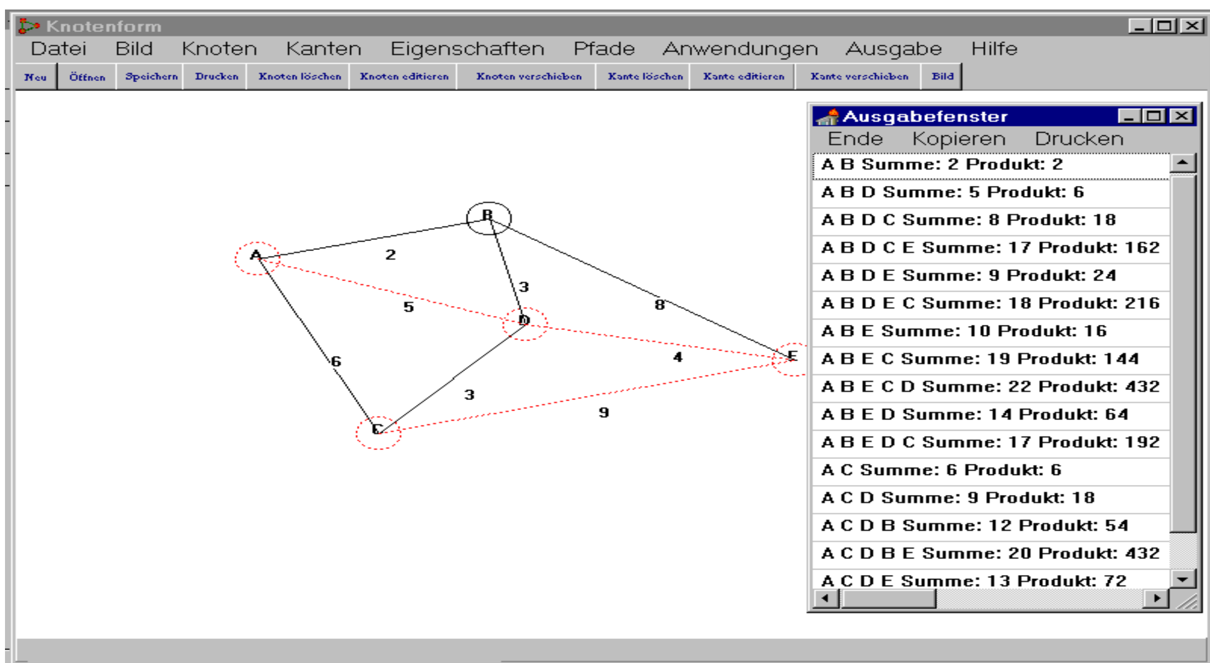






Dieselben Informationen des Anzeigefensters erscheinen auch im Ausgabefenster,das sich durch das Menü Ausgabefenster im Menü Ausgabe aktivieren lässt.

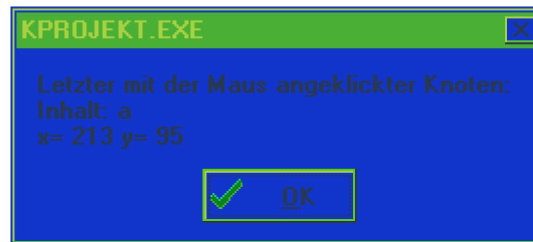
Das Ausgabefenster:



Wird die Shift-Taste gedrückt,während auf die rechte Hälfte der Panelfläche mit der linken Maustaste geklickt wird,erscheint ein Informationsfenster,in dem der Inhalt und die Koordinaten des letzten mit der linken Maustaste durch Klick ausgewählten Knotens erscheint.(Wenn kein Knoten zuvor ausgewählt wurde,erscheint der erste in den Graph eingefügte Knoten mit den entsprechenden Informationen.)

Der letzte mit der linken Maustaste angewählte Knoten ist z.B. bei den Pfad-Algorithmen der Ausgangsknoten für die zu suchenden Pfade.

Informationsfenster für den letzten mit der linken Maustaste durch Klick ausgewählten Knoten:



An der Oberseite des Hauptfensters (Knotenform) befindet sich die Menüleiste,bei der durch Mausklick die verschiedenen Menüs mit ihren Untermenüs angewählt werden können.Die Untermenüs können jeweils ebenfalls durch Mausklick gestartet werden.Unterhalb der Menüleiste liegt die Button-Menüleiste.In diese Leiste sind die die wichtigsten Untermenüs noch einmal ausgelagert.Sie können durch Mausklick auf den entsprechenden Button ausgeführt werden.

Die Menüleiste:

Die Button-Menüleiste:

Das Menü Datei:

Das Menü Datei besteht aus den Untermenüs:

Neue Knoten,Graph laden,Graph hinzufügen,Graph speichern,Graph speichern unter,Graph drucken,Quit

Neue Knoten:

Bei Anwahl dieses Menüs wird eine leere Zeichenfläche hergestellt,auf der ein neuer Graph erzeugt werden kann.Ein vorher vorhandener Graph wird gelöscht.Dieses Menü kann auch über die Button-Menü-Leiste mittels des Buttons Neu gestartet werden.

Graph laden:

Ein Dateiauswahlfenster erscheint,in dem ein auf einem Datenträger gespeicherter Graph (Standardextention: gra) ausgewählt werden kann und danach geladen wird.Der Graph wird auf der Zeichenfläche gezeichet.Dieses Menü kann auch über die Button-Menü-Leiste mittels des Buttons Öffnen gestartet werden.

Graph hinzufügen:

Ein Dateiauswahlfenster erscheint,in dem ein auf einem Datenträger gespeicherter Graph (Standardextension: gra) ausgewählt werden kann und danach geladen und zu dem auf der Zeichenfläche befindlichen Graph hinzugefügt wird.Der Gesamtgraph wird auf der Zeichenfläche dargestellt.

Graph speichern:

Ist dem auf der Zeichenfläche vorhandenen Graph schon eine Dateiname zugeordnet,wird der (eventuell veränderte) Graph unter diesem Namen gespei-

chert.Ist dem Graph noch kein Dateiname zugeordnet,erscheint ein Dateiauswahlfenster,in dem ein Dateiname sowie ein Verzeichnis auf einem Datenträger festgelegt werden kann,unter bzw. in dem der Graph gespeichert wird.(Standardextension: gra)

Graph speichern unter:

Ein Dateiauswahlfenster erscheint,in dem ein Dateiname sowie ein Verzeichnis auf einem Datenträger festgelegt werden kann,unter bzw. in dem der Graph der Zeichenfläche gespeichert wird.(Standardextension: gra)
Dieses Menü kann auch über die Button-Menü-Leiste mittels des Buttons Speichern gestartet werden.

Graph drucken:

Nach Anwahl dieses Menüs erscheint ein Druckerauswahlfenster,in dem ein neuer Drucker gewählt oder der bisher eingestellte Drucker beibehalten werden kann.Außerdem können Eigenschaften des Druckers eingestellt werden.Danach wird das Bild des Graphen der Zeichenfläche auf dem eingestellten Drucker gedruckt.Dieses Menü kann auch über die Button-Menü-Leiste mittels des Buttons Drucken gestartet werden.

Quit:

Das Programm Knotengraph wird beendet.Wenn die Veränderungen eines Graphen noch nicht gespeichert wurden,erscheint vorher eine Abfrage,ob der Graph gespeichert werden soll.Falls Speichern gewählt wurde,wird dann automatisch zu dem Ablauf eines der Menüs Graph speichern oder Graph speichern unter verzweigt.Außerdem wird vor dem Beenden abgefragt,ob die Beendigung des Programms wirklich gewünscht wird.

Das Menü Bild:

Das Menü Bild besteht aus den Untermenüs:

Ergebnis zeichnen, Bild zeichnen, Bild kopieren, Bild wiederherstellen, Ungerichtete Kanten, Knotenradius ändern, Genauigkeit Knoten, Genauigkeit Kanten

Ergebnis zeichnen:

Wenn ein Algorithmus der Menüs Pfade oder Anwendungen gestartet wird,wird der ursprünglich vorhandene Graph so verändert,dass auf ihm ein Ergebnis z.B. durch Markierung von Knoten oder Kanten oder durch Anzeige von neuen Knoten-oder Kanteninhalten nach Beenden des Algorithmus angezeigt wird.Durch Anwahl des Menüs Bild zeichnen, wird der ursprüngliche Graph auf der Zeichenfläche wieder angezeigt.Durch Aufruf des Menüs Ergebnis zeichnen kann nochmal der letzte Ergebnisgraph angezeigt werden.

Bild zeichnen:

Wenn ein Algorithmus der Menüs Pfade oder Anwendungen gestartet wird,wird der ursprünglich vorhandene Graph so verändert,dass auf ihm ein Ergebnis z.B. durch Markierung von Knoten oder Kanten oder durch Anzeige von neuen Knoten-oder Kanteninhalten nach Beenden des Algorithmus angezeigt wird.Durch Anwahl des Menüs Bild zeichnen,wird der ursprüngliche Graph auf der Zeichenfläche wieder angezeigt.
(Dies geschieht auch,wenn ein anderes Untermenü,das nicht zu dem Menü Ausgabe gehört,aufgerufen wird.)

Dieses Menü kann auch über die Button-Menü-Leiste mittels des Buttons Bild gestartet werden.Durch Anwahl dieses Button-Menüs wird auch das Erzeugen von Knoten und Kanten durch (Doppel-/Einfach-)Mausklick mit der linken bzw. rechten Maustaste nach Ablauf eines Algorithmus erneut aktiviert.Dieser Button ist also immer dann zu betätigen,wenn beim Erscheinen eines Ergebnisbildes des Graphen nach Anwahl der Menüs Anwendungen oder Pfade der ursprüngliche Graph editiert oder verändert werden soll.

Bild kopieren:

Das Bild des Graphen auf der Zeichenfläche wird in die Zwischenablage kopiert und steht dann anderen Windows-Anwendungen als Bitmap zur Verfügung.Es kann in einer Eingabeaufforderung gewählt werden,ob eventuell das Bild des Ergebnisgraphen eines Algorithmus statt des Bildes des ursprünglichen Graphen in die Zwischenablage kopiert werden soll.(Wenn ein Algorithmus der Menüs Pfade oder Anwendungen gestartet wird,wird der ursprünglich vorhandene Graph so verändert,dass auf ihm ein Ergebnis z.B. durch Markierung von Knoten oder Kanten oder durch Anzeige von veränderten Knoten-oder Kanteninhalten nach Beenden des Algorithmus angezeigt wird.)

Das Bild des Ergebnisgraphen kann auch während des Ablaufs eines Algorithmus im Demomodus in die Zwischenablage gebracht werden,so dass Schnappschüsse von Zwischenergebnisssen möglich sind.

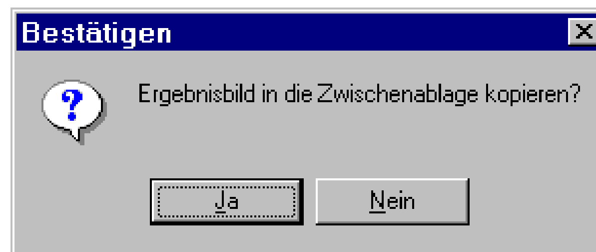


Bild wiederherstellen:

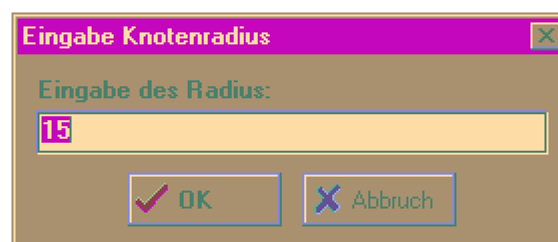
Wird der vorhandene Graph durch eine der Operationen der Menüs Knoten bzw. Kanten oder durch Anwählen des Untermenüs Neue Knoten des Menüs Datei bzw. des Untermenüs Ungerichtete Kanten des Menüs Bild verändert,kann dieser Schritt durch Anwählen des Menüs Bild wiederherstellen rückgängig gemacht werden.

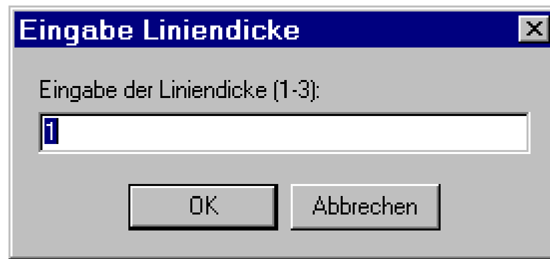
Ungerichtete Kanten:

Nach Anwahl dieses Menüs mit der linken Maustaste werden alle gerichteten Kanten des vorgegebenen Graphen in ungerichtete Kanten umgewandelt.Dieser Schritt kann durch Anwählen des Menüs Bild wiederherstellen wieder rückgängig gemacht werden.

Knotenradius ändern:

Nach Anwahl dieses Menüs durch Klick mit der linken Maustaste erscheint ein Eingabefenster,in das ein neuer Radius für das Zeichnen der Knotenkreise auf der Zeichenfläche vorgegeben werden kann.Der Kreis schon gezeichneter bzw. vorhandener Knoten wird ebenfalls an die neue Größe angepaßt.Vorgabewert für den Radius ist 15.Kleinster Wert ist 10 und größter Wert 100.

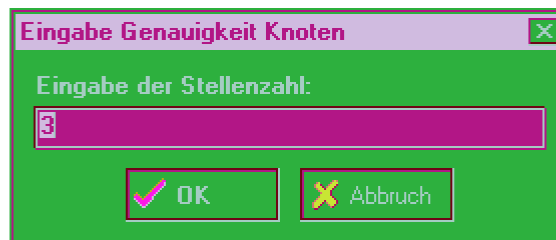




Nach Auswahl des Radius erscheint ein weiteres Eingabefenster,in dem die Liniendicke,mit der Kanten und Knoten gezeichnet werden,in drei Stufen von 1 bis 3 gewählt werden kann.Vorgabewert ist 1.Schon gezeichnete Elemente werden wiederum angepaßt.

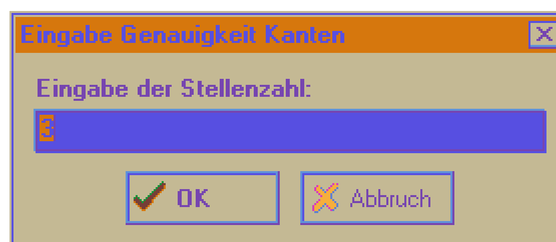
Genauigkeit Knoten:

In einem Eingabefenster kann die Art bzw. Genauigkeit,mit der der Inhalt (Wert) der Knoten des Graphen auf der Zeichenfläche dargestellt wird,gewählt werden.
Bei Knoteninhalten,die als Zahlenwert interpretiert werden können,bedeutet eine positive Zahl (Bereich von 0 bis 6) die Anzahl der Nachkommastellen,die dargestellt werden.Andere Knoteninhalte werden nicht verändert.Bei Eingabe einer negativen Zahl werden die Knoteninhalte mit der Zeichenzahl angezeigt,die dem Absolutwert der negativen Zahl entspricht (unabhängig davon,ob der Inhalt als Zahl interpretiert werden kann oder nicht).Bei Eingabe von x werden stets alle (eingegebenen) Zeichen des Knoteninhalts angezeigt.
Vorgabewert ist ein Wert von 3 für die Knotengenauigkeit.



Genauigkeit Kanten:

In einem Eingabefenster kann die Art bzw. Genauigkeit,mit der der Inhalt (Wert) der Kanten des Graphen auf der Zeichenfläche dargestellt wird,gewählt werden.
Bei Kanteninhalten,die als Zahlenwert interpretiert werden können,bedeutet eine positive Zahl (Bereich von 0 bis 6) die Anzahl der Nachkommastellen,die dargestellt werden.Andere Kanteninhalte werden nicht verändert.Bei Eingabe einer negativen Zahl werden die Kanteninhalte mit der Zeichenzahl angezeigt,die dem Absolutwert der negativen Zahl entspricht (unabhängig davon,ob der Inhalt als Zahl interpretiert werden kann oder nicht).Bei Eingabe von x werden stets alle (eingegebenen) Zeichen des Kanteninhalts angezeigt.
Vorgabewert ist ein Wert von 3 für die Kantengenauigkeit.



Menü Knoten:

Das Menü Knoten besteht aus den Untermenüs:

Knoten erzeugen,Knoten löschen,Knoten editieren,Knoten verschieben

Knoten erzeugen:

Nach Anwahl diese Menüs ist mit der linken Maustaste auf eine Stelle der Zeichenfläche zu klicken.Danach erscheint ein Eingabefenster,in das ein Knoteninhalt (Wert) (als String) eingegeben werden kann.Nach Auswahl von OK (bzw. Entertaste),wird an der Stelle des Mausklicks auf der Zeichenfläche ein Knoten mit Anzeige des Inhalts gezeichnet.

Statt der Anwahl dieses Menüs kann ein Knoten auch durch einen Doppelklick mit der linken Maustaste auf eine Stelle der Zeichenfläche erzeugt werden.

Knotendaten-Eingabefenster:

Knoten löschen:

Nach Anwahl dieses Menüs ist mit der linken Maustaste auf einen zu löschenden Knoten zu klicken,der dann inklusive der mit ihm inzidenten Kanten aus dem Graph gelöscht wird.Dieses Menü kann auch über die Button-Menü-Leiste mittels des Buttons Knoten löschen gestartet werden.

Knoten editieren:

Nach Anwahl dieses Menüs ist mit der linken Maustaste auf den zu editierenden Knoten zu klicken.Darauf erscheint das obige Knotendaten-Eingabefenster,das den bisherigen Knoteninhalt (Wert) anzeigt.Stattdessen kann jetzt ein anderer Inhalt eingegeben werden.Durch Auswahl von OK (oder Enter-Taste) wird der Inhalt des Knotens verändert.Andernfalls kann Abbrechen gewählt werden.Dieses Menü kann auch über die Button-Menü-Leiste mittels des Buttons Knoten editieren gestartet werden.

Knoten verschieben:

Nach Anwahl dieses Menüs ist mit der linken Maustaste ein zu verschiebender Knoten des Graphen auszuwählen und die Maustaste ist dabei gedrückt zu halten.Während die Maustaste gedrückt gehalten wird,kann der Knoten durch Bewegen der Maus verschoben werden.Dabei andern sich auch die mit dem Knoten inzidenten Kanten in ihrer Länge und Position.Sobald die linke Maustaste wieder losgelassen wird,nimmt der verschobene Knoten seine neue Position ein.Dieses Menü kann auch über die Button-Menü-Leiste mittels des Buttons Knoten verschieben gestartet werden.

Menü Kanten:

Das Menü Kanten besteht aus den Untermenüs:

Kanten erzeugen, Kanten löschen, Kante editieren, Kante verschieben

Kanten erzeugen:

Nach Auswahl dieses Menüs sind nacheinander die zwei Knoten mit der linken Maustaste anzuklicken,zwischen denen eine neue Kante eingefügt werden soll.Nach Anwahl des zweiten Knotens erscheint das Kantendaten-Eingabefenster:

Kantendaten-Eingabefenster:

Durch Mausklick auf die entsprechenden Checkboxen kann gewählt werde,ob es sich um eine vom ersten angewählten Knoten betrachtet ausgehende oder eingehende (gerichtete) Kante handeln soll.Bei Auswahl beider Optionen oder keiner Option wird eine ungerichtete Kante erzeugt.Der Kante kann durch Klick auf die entsprechenden Checkboxen bezüglich des einzugebenden Inhalts (Wert) der Datentyp Real (Extended) oder Integer zugeordnet werden.Wird keine dieser Optionen gewählt,ist der Datentyp des Inhalts String.In das Feld Weite kann eine positive oder negative Integerzahl eingegeben werden,wodurch die Weite der Kante,d.h. die Größe der Verschiebung ihrer Mitte (der Verbindungspunkt der beiden Strecken des Polygonzuges) vom Mittelpunkt der Strecke zwischen den Knotenmittelpunkten des Anfangs-und Endknotens der Kante vorgegeben wird.(Dadurch können verschiedene Kanten zwischen gleichen Knoten gezogen werden,so dass die Kanten auch optisch verschieden erscheinen und nicht aufeinander liegen.Die Verschiebung erfolgt senkrecht zur Verbindungsstrecke der Mittelpunkte der Kreise des Anfangs-und Endknotens.)Die Kantenweite kann auch durch den Schiebregler eingestellt werden.In das Feld Inhalt wird schließlich der Inhalt (Wert) der Kante eingegeben.Dabei ist der vorgegebene Datentyp zu berücksichtigen.Bei der Vorgabe Real werden alle Eingaben zurückgewiesen,die keine Real-Zahlen sind oder aber nicht im zulässigen Wertebereich liegen.Dasselbe gilt für den Typ Integer.Wenn keine der Optionen Real oder Integer gewählt wurde und der Datentyp automatisch String ist (s.o.),können beliebige Zeichen eingegeben werden.
Durch Anwahl von OK wird die Kante erzeugt (auch mittels Enter-Taste),durch Anwahl von Abbrechen wird keine Kante erzeugt.

Statt das Menü anzuwählen kann auch zur Erzeugung einer Kante ein Doppelklick mit der rechte Maustaste auf eine beliebige Stelle der Zeichenfläche vorgenommen werden.Danach sind die Knoten der Kante mit der linken Maustaste auszuwählen,und es ist dann wie weiter oben beschrieben zu verfahren.

Kante löschen:

Nach Anwahl dieses Menüs ist die zu löschende Kante durch Klick mit der linken Maustaste anzuwählen.Daraufhin wird die Kante aus dem Graph gelöscht.Dieses Menü kann auch über die Button-Menü-Leiste mittels des Buttons Kante löschen gestartet werden.

Kante editieren:

Nach Anwahl dieses Menüs ist die zu editierende Kante durch Klick mit der linken Maustaste anzuwählen.Danach erscheint das unter Kante erzeugen beschriebene Kantendaten-Eingabefenster,das die aktuellen Daten der Kante zeigt.Diese Daten können nun entsprechend abgeändert werden.Durch Klick auf OK werden die neuen Daten der Kante zugeordnet und im Graph gezeichnet bzw. angezeigt.Bei Wahl von Abbrechen ändern sich die Daten der Kante nicht.Dieses Menü kann auch über die Button-Menü-Leiste mittels des Buttons Kante editieren gestartet werden.

Kante verschieben:

Nach Anwahl dieses Menüs kann durch Klick mit der linken Maustaste die zu verschiebende Kante angewählt werden.Danach ist die Maustaste gedrückt zu halten,und die Kantenmitte kann senkrecht zur Verbindungstrecke des Anfangs-und Endknoten (Mitte der Kreise) der Kante verschoben werden,d.h. die Weite der Kante wird geändert.Dieses Menü kann auch über die Button-Menü-Leiste mittels des Buttons Kante verschieben gestartet werden.

Menü Eigenschaften:

Das Menü Eigenschaften besteht aus folgenden Untermenüs:

Anzahl Kanten und Knoten, Parallelkanten und Schlingen, Eulerlinie, Kreise, Anzahl Brücken, Knoten anzeigen, Kanten anzeigen

Anzahl Kanten und Knoten:

Ein Ausgabefenster zeigt die Zahl der Knoten und Kanten des Graphen,die Anzahl der Gebiete für planare Graphen (nach der Euler-Formel,wobei zwischen parallelen bzw. antiparallelen Kanten jeweils Gebiete vorhanden sind,auch falls die Kanten optisch zusammenfallen sollten),die Anzahl der Komponenten und die Anzahl der Kanten,die den verschiedenen Datentypen Real,Integer und String zugeordnet sind,an.

Parallelkanten und Schlingen:

Angegeben wird die Zahl der Kanten zwischen ungleichen Knoten (d.h. keine Schlingen),die Zahl der Kanten zwischen gleichen Knoten (Schlingen),die Zahl der parallelen Kanten,wenn der Graph als gerichtet aufgefaßt wird,die Zahl der antiparallelen Kanten,wenn der Graph als gerichtet aufgefaßt wird, und die Zahl der parallelen Kanten,wenn der Graph als ungerichtet aufgefaßt wird.

Eulerlinie:

Angegeben wird,ob der Graph eine geschlossene oder offene Eulerlinie hat für die Fälle,dass der Graph jeweils als gerichtet oder ungerichtet aufgefaßt werden soll.Im Falle der offenen Eulerlinie wird auch der Anfangs-und Endknoten angegeben.Der Algorithmus beruht auf den bekannten Eulerkriterien bezüglich der Knotengrade des Graphen.

Kreise:

Zunächst wird abgefragt,ob der Graph als gerichteter oder ungerichteter Graph untersucht werden soll.Im ersteren Fall wird der Graph wie vorgegeben untersucht,andernfalls werden alle gerichteten Kanten in ungerichtetete umgewandelt.

Es wird untersucht,ob der vorliegende Graph bzw. der ungerichtete Graph,einen Kreis hat.(Als kleinster Kreis zählt dabei auch schon ein Pfad aus zwei Kanten.)
Danach wird die Kantenzahl eines Kreises mit kleinster Kantenzahl (aber mit mehr als zwei Kanten) und die Kantenzahl eines Kreises mit größter Kantenzahl ermittelt (aber mit mehr als zwei Kanten).
Für den Fall eines schlichten Graphs wird angegeben,ob er vollständig ist.

Außerdem wird auf Grund eines notwendigen Kriterium für die Planarität von Graphen geprüft,ob der Graph nichtplanar ist.
(Näheres siehe im Kapitel IV bei Planarität und Eulerformel.)

Schließlich werden alle Kreise mit vorzugebender fester Pfadlänge gesucht.Ergebnisse werden im Ausgabefenster ausgegeben und im Demomodus als Pfade markiert angezeigt.

Der zu Grunde liegende Algorithmus beruht zunächst auf dem Prinzip der Tiefensuche,wobei von jedem Knoten nacheinander aus durch jeweils sukzessive Auswahl aller Kanten von einem erreichten Knoten alle geschlossene Kreise zum Startknoten zurück gesucht werden.Als Abbruchkriterium dient einerseits,dass die vorgegebene Kantenzahl im momentanen Pfad schon überschritten wurde.Ein anderes Beschränkungskriterium ergibt sich aus folgender Überlegung:Jeder Kreis von vorgegebener Länge von Knoten A,der auch durch Knoten B verläuft,braucht bei dem Suchen von Knoten B aus nicht mehr berücksichtigt zu werden,da er schon gefunden wurde.Daher werden die ausgehenden Kanten von Knoten A markiert.Trifft der Algorithmus von Knoten B aus dann auf eine schon markierte Kante,kann der Algorithmus beendet werden,da dieser Kreis schon gefunden wurde oder aber kein Kreis der vorgegebenen Länge durch diese Kante und durch Knoten B existiert.Durch das immer weitere Markieren der Kanten,terminiert das Verfahren relativ rasch.In ähnlicher Weise funktionieren auch die Algorithmen zur Suche des kleinsten und größten Kreises.Beim Suchen des kleinsten Kreises kann dabei beim Finden eines momentanen Kreises kleinerer Länge als die Kantenzahl des Abbruchkriteriums die Kantenzahl des Abbruchkriteriums auf die aktuelle Länge des momentanen Kreises herabgesetzt werden.Zu Beginn des Verfahrens ist die Zahl des Abbruckkriteriums gleich der Zahl aller Kanten des Graphen vergrößert um Eins.

Anzahl Brücken:

Angegeben wird die Anzahl der Brücken im vorliegenden Graph.Außerdem werden die Brücken mittels der Bezeichnung durch Anfangs-und Endknoten im Ausgabefenster (Menü Ausgabe) und im Anzeigefenster (erreichbar durch Mausklick auf die Panelfläche) der Zeichenfläche angegeben.
Definition Brücke:
Wenn man eine Brückenkante aus einem Graph löscht,erhöht sich die Anzahl der Komponenten um 1.

Der Algorithmus beruht darauf,dass für jede Kante des Graphen ein Pfad gesucht wird,der vom Anfangsknoten zum Zielknoten aber nicht über die Kante selber führt.Als Suchverfahren dient dabei ein der Tiefensuche ähnliches Verfahren,das aber beim ersten Finden des Zielknotens terminiert.

Knoten anzeigen:

Nach Anwahl dieses Menüs wird die Anzahl der Knoten des Graphen in einem Ausgabefenster angezeigt.Im Anzeigefenster Anzeigefenster (erreichbar durch Mausklick auf die Panelfläche) und im Ausgabefenster (Menü Ausgabe) werden alle Knoten mit Knoteninhalt, Zeichenflächenkoordinaten X und Y des Mittelpunkts (des Kreises) des Knotens,sowie dem gerichteten und ungerichtetem Knotengrad angezeigt.Der gerichtete Knotengrad ist die Anzahl der ausgehenden Kanten vermindert um die Anzahl der eingehenden Kanten.Der ungerichtete Knotengrad ist die Anzahl der mit diesem Knoten inzidenten Kanten.
Außerdem ist vorgesehen,den Typ des Knotens anzuzeigen.Die Anzeige bleibt leer,da diese Eigenschaft beim Programm Knotengraph nicht verwendet wird.(aber möglicherweise von Benutzern der Entwicklungsumgebung von Knotengraph.Dann erscheint hier der entsprechende Eintrag.)

Kanten anzeigen:

Nach Anwahl dieses Menüpunktes werden die Anzahl der Kanten sowie die Anzahl der gerichteten und ungerichteten Kanten und die Kantensumme (Real- und Integer-Bewertung) aller Kanten des Graphen (Kanten vom Typ String zählen dabei mit dem Wert 1) ausgegeben.

Im Anzeigefenster und im Ausgabefenster werden alle Kanten mit den Anfangs- und Endknoten(-Inhalten),dem Kanteninhalt,dem Typ der Kante (r Real,i Integer und s string) sowie der Information gerichtete Kante oder ungerichtete Kante angezeigt.

Menü Pfade (Wege,Bäume,Kreise):

Das Menü Pfade besteht aus folgenden Untermenüs:

Alle Pfade, Alle Kreise, Minimale Pfade, Anzahl Zielknoten, Tiefer Baum, Weiter Baum, Abstand von zwei Knoten, Alle Pfade zwischen zwei Knoten, Minimales Gerüst des Graphen

In diesem Kapitel wird nur die Bedienung der Menüalgorithmen besprochen.Für deren Beschreibung und für die Erläuterung des mathematischen Umfelds,für didaktisch-methodische Bemerkungen,für Unterrichtsskizzen sowie für die Angabe von Beispielgraphen und Anwendungsaufgaben zu den Algorithmen siehe die Kapitel C II und C III Suchverfahren in Graphen,Baum-und Pfadalgorithmen,Gerüste,Alle Pfade und Minimale Pfade zwischen zwei Knoten.

Der letzte Mausklickknoten ist der letzte mit der linken Maustaste angewählte Knoten.Wenn dieser nicht existiert,ist der letzte Mausklickknoten der erste (vom Benutzer) in den Graph eingefügte Knoten.

Alle Pfade:

Nach Anwahl dieses Menüs werden ausgehend vom letzten Mausklickknoten als Ausgangsknoten alle Pfade zu anderen Knoten des Graphen bestimmt und nacheinander rot-gestrichelt-markiert auf der Zeichenfläche angezeigt sowie als Folge von Knoteninhalten (Werten) der Pfade mit Kantensumme und Kantenprodukt im Label Ausgabe1 ausgegeben als auch im Anzeige-und Ausgabefenster angezeigt.Im Demomodus kann die Anzeige und Ausgabe der Pfade schrittweise ablaufen.
Falls der Graph als Binärbaum aufgefaßt werden kann,kann die Option Inorder-Durchlauf vom letzten Mausklickknoten aus gewählt werden,und es wird ein Inorder-Durchlauf gestartet,wobei die als erste eingefügte ausgehende Kante von einem Knoten jeweils auf den linken Teilbaum und die als zweite eingefügte Kante jeweils auf den rechten Teilbaum zeigt.Die Pfade werden im Demomodus markiert auf der Zeichenoberfläche und auch als Knotenfolge des Pfades mit Kantensumme und Kantenprodukt im Anzeige-und Ausgabefenster sowie im Label Ausgabe1 angezeigt.Vom Algorithmus wird **nicht** überprüft,ob ein **geordneter** Binärbaum vorliegt.

Alle Kreise:

Nach Anwahl dieses Menüs werden ausgehend vom letzten Mausklickknoten als Ausgangsknoten alle Kreise zurück zum Ausgangsknoten bestimmt und nacheinander rot-gestrichelt-markiert auf der Zeichenfläche angezeigt,sowie als Folge von Knoteninhalten der Pfade mit Kantensumme und Kantenprodukt im Label Ausgabe1 ausgegeben,als auch im Anzeige-und Ausgabefenster angezeigt.Im Demomodus kann die Anzeige und Ausgabe der Pfade verlangsamt ablaufen.

Minimale Pfade:

Nach Anwahl dieses Menüs werden ausgehend vom letzten Mausklickknoten als Ausgangsknoten alle Pfade kleinster Kantensumme (Kanten des Datentyps string zählen bei der Kantensumme als 1) zu anderen Knoten des Graphen bestimmt und nacheinander rot-gestrichelt-markiert auf der Zeichenfläche angezeigt,sowie als Folge von Knoteninhalten der Pfade mit Kantensumme und Kantenprodukt im Label Ausgabe1 ausgegeben,als auch im Anzeige-und Ausgabefenster angezeigt.Im Demomodus kann die Anzeige und Ausgabe der Pfade verlangsamt ablaufen.

Anzahl Zielknoten:

Nach Anwahl dieses Menüs werden ausgehend vom letzten Mausklickknoten als Ausgangsknoten die Anzahl der vom Ausgangsknoten aus auf Pfaden erreichbaren Zielknoten des Graphen ermittelt und in einem Ausgabefenster angezeigt.

Tiefer Baum:

Nach Anwahl dieses Menüs werden ausgehend vom letzten Mausklickknoten als Ausgangsknoten alle Pfade (in der Reihenfolge) eines tiefen Baumdurchlaufs mit dem Ausgangsknoten als Wurzel zu anderen Knoten des Graphen bestimmt, nacheinander rot-gestrichelt-markiert auf der Zeichenfläche angezeigt sowie als Folge von Knoteninhalten der Pfade mit Kantensumme und Kantenprodukt im Label Ausgabe1 ausgegeben,als auch im Anzeige-und Ausgabefenster angezeigt.(Kanten vom Typ string zählen dabei mit dem Wert 1.)Im Demomodus kann die Anzeige und Ausgabe der Pfade schrittweise ablaufen.Zu Beginn kann in einem Eingabefenster durch Ja oder Nein gewählt werden,ob der Durchlauf in der Reihenfolge Postorder oder Preorder erfolgen soll.

Weiter Baum:

Nach Anwahl dieses Menüs werden ausgehend vom letzten Mausklickknoten als Ausgangsknoten alle Pfade (in der Reihenfolge) eines weiten Baumdurchlaufs mit dem Ausgangsknoten als Wurzel zu anderen Knoten des Graphen bestimmt, nacheinander rot-gestrichelt-markiert auf der Zeichenfläche angezeigt sowie als Folge von Knoteninhalten der Pfade mit Kantensumme und Kantenprodukt im Label Ausgabe1 ausgegeben,als auch im Anzeige-und Ausgabefenster angezeigt.(Kanten vom Typ string zählen dabei mit dem Wert 1.)Im Demomodus kann die Anzeige und Ausgabe der Pfade schrittweise ablaufen.

Abstand von zwei Knoten:

Nach Anwahl dieses Menüs wird der Benutzer aufgefordert,nacheinander zwei Knoten durch Klick mit der linken Maustaste auszuwählen.Danach wird der Pfad kleinster Kantensumme (Kanten des Datentyps string zählen dabei mit dem Wert Eins) bestimmt und rot-gestrichelt-markiert auf der Zeichenfläche angezeigt,sowie als Folge von Knoteninhalten des Pfades einschließlich der Kantensumme im Label Ausgabe1 ausgegeben als auch im Anzeige-und Ausgabefenster angezeigt.Algorithmen:Dijkstra und Ford (negative Kantenbewertung)

Wenn der Graph als Binärbaum aufgefaßt werden kann,wird durch eine Eingabeaufforderung abgefragt,ob der Algorithmus Binäres Suchen vom ersten durch Mausklick gewählten Knoten aus aufgerufen werden soll,der nach einem Knotenwert,der gleich dem Inhalt (Wert) des zweiten durch Mausklick gewählten Knoten ist,im Baum binär sucht.Falls ein entsprechender Knoten existiert,wird der Pfad dann rot-gestrichelt-markiert auf der Zeichenoberfläche und als Knotenfolge des Pfades mit Kantensumme und Kantenprodukt im Anzeige-und Ausgabefenster sowie im Label Ausgabe1 angezeigt.Das Vorliegen eines **geordneten** Binärbaums bezüglich der Knoteninhalte (Werte) wird vom Algorithmus **nicht** überprüft und führt bei Nichtvorliegen zu falschen Ergebnissen.

Alle Pfade zwischen zwei Knoten:

Nach Anwahl dieses Menüs wird der Benutzer aufgefordert,nacheinander zwei Knoten durch Klick mit der linken Maustaste auszuwählen.Danach werden alle Pfade mit dem ersten Knoten als Ausgangsknoten und dem zweiten Knoten als Zielknoten bestimmt und nacheinander rot-gestrichelt-markiert auf der Zeichenfläche (verlangsamt) angezeigt,sowie als Folge von Knoteninhalten der Pfade einschließlich der Kantensumme und des Kantenprodukts im Label Ausgabe1 ausgegeben (Kanten des Datentyps string zählen dabei mit dem Wert 1) als auch im Anzeige-und Ausgabefenster angezeigt.

Minimales Gerüst des Graphen:

Nach Anwahl dieses Menüs wird auf dem Graphen ein minimal spannender Baum nach dem Algorithmus von Kruskal bestimmt.Die Kanten des Baumes werden rot-

gestrichelt-markiert angezeigt,der Baum als Folge von Kanteninhalten im Label Ausgabe1 mit seiner Kantensumme (Kanten des Datentyps string zählen dabei mit dem Wert 1) angezeigt,sowie im Anzeigefenster und im Ausgabefenster ausgegeben.Im Demomodus läßt sich die Funktionsweise des Algorithmus schrittweise (und verlangsamt bei Wahl einer Pausenzeit) verfolgen.Es wird die Reihenfolge der Auswahl der Gerüstkanten im Label Ausgabe1 angezeigt.

Menü Anwendungen:

Das Menü Anwendungen besteht aus folgenden Untermenüs:

Netzwerkzeitplan, Hamiltonkreise, Eulerlinie (geschlossen),Färbbarkeit, Eulerlinie (offen),Endlicher Automat, Graph als Relation, Maximaler Netzfluss,Maximales Matching, Gleichungssystem, Markovkette (abs),Markovkette Graph redu- (stat),Graph reduzieren, Minimale Kosten, Transportproblem,Optimales Matching und chinesischer Briefträger.

In diesem Kapitel wird nur die Bedienung der Menüalgorithmen besprochen.Für deren Beschreibung und für Erläuterungen des mathematischen Umfeldes,für didaktisch-methodische Bemerkungen,für Unterrichtsskizzen sowie für Angabe von Beispielgraphen und Anwendungsaufgaben zu den Algorithmen siehe die Kapitel C I bis C XIV,die entsprechende weiterführende Darstellungen enthalten.

(Zur Definition des Begriffs letzter Mausklickknoten siehe die Definition weiter oben.)

Netzwerkzeitplan:

Der Graph auf der Zeichenfläche sollte bei Aufruf dieses Menüs nur gerichtete Kanten ohne Schlingen aufweisen,kreisfrei und zusammenhängend sein und genau einen Quell-und Zielknoten aufweisen,von dem nur Kanten ausgehen bzw. in den nur Kanten einlaufen.

Der Graph wird als Netzwerk aufgefaßt,zu dem ein Zeitplan nach der CPM-Methode erstellt werden soll.Die Knoten bedeuten dabei Ereignisse und die Kanten Vorgänge.Die Kanteninhalte (Werte) werden als Zeiten der Vorgänge aufgefaßt.(Kanten des Datentyps string ist der Wert 1 zugeordnet.)
Nach Start des Algorithmus wird der Quell-und Zielknoten (das Start-und Zielereignis) automatisch bestimmt,und es erscheinen nacheinander zwei Eingabefenster,in denen jeweils den beiden (Knoten-)Ereignissen eine Zeit zugeordnet werden soll.Diese Zeiten bedeuten die (frühest mögliche) Anfangszeit und die (spätestens mögliche) Endzeit des Projekts.Wird der vorgegebene Wert Null bei der Eingabe der Endzeit übernommen,wird automatisch die frühestens mögliche Endzeit,die sich aus der Anfangszeit und dem kritischen Pfad des Graphen (des Projekts) ergibt,als spätest mögliche Endzeit angenommen.Dies gilt auch,wenn eine Zeit eingegeben wird,die kleiner als die frühest mögliche Endzeit ist.

Der Algorithmus berechnet danach für jedes (Knoten-)Ereignis die (frühest mögliche) Anfangszeit,die (spätest mögliche) Endzeit und die Pufferzeit, sowie für jeden (Kanten-)Vorgang die frühstmögliche Anfangszeit,die frühstmögliche Endzeit,die spätmöglichste Anfangszeit und die spätmöglichste Endzeit sowie die Pufferzeit.Die Ergebnisse können für jedes (Knoten-)Ereignis und für jeden (Kanten-)Vorgang durch Anklicken mit der linken Maustaste auf die Knoten bzw. Kanten in einem Fenster angezeigt werden.Sie werden außerdem im Anzeige-und Ausgabefenster dargestellt.Die minimale Projektzeit wird in einem Fenster nach Ablauf des Algorithmus ausgegeben.Der kritische Weg wird rot-gestrichelt-markiert dargestellt.

Im Demomodus kann die Arbeitsweise des Algorithmus schrittweise und verlangsamt verfolgt werden.

Hamiltonkreise:

Der Graph auf der Zeichenfläche sollte bei Aufruf dieses Menüs aus einer Komponente bestehen.Er kann gerichtete und ungerichtete Kanten enthalten.Zu Beginn erscheint ein Abfragefenster,in dem bestimmt werden kann,ob die gerichteten Kanten ebenfalls als ungerichtet aufgefaßt werden sollen oder nicht.(Ungerichteten Graph untersuchen)

Danach werden ausgehend vom letzten Mausklickknoten alle Hamiltonkreise des Graphen bestimmt und rot-gestrichelt-markiert auf der Zeichenfläche als Pfade dargestellt.Die Lösung des Traveling-Salesman-Problems wird in einem Fenster als Folge der Knoteninhalte (Werte) mit Summe sowie Produkt der Kantenbewertungen ausgegeben.(Kanten des Datentyps string ist der Wert 1 zugeordnet.)

Falls keine Hamiltonkreise gefunden wurden,wird eine entsprechende Meldung angezeigt.

Alle Hamilton-Pfade werden auch mit Summe und Produkt der Kantenbewertungen im Anzeige-und Ausgabefenster dargestellt.Im Demomodus kann die Arbeitsweise des Algorithmus schrittweise und verlangsamt verfolgt werden.

Eulerlinien (geschlossen):

Der Graph auf der Zeichenfläche sollte bei Aufruf dieses Menüs aus einer Komponente bestehen.Er kann gerichtete und ungerichtete Kanten enthalten.Die Knoten des Graphen sollten alle einen geraden Knotengrad (ungerichteter Graph) oder alle einen Knotengrad Null (gerichteter Graph) besitzen.Zu Beginn erscheint ein Abfragefenster,in dem bestimmt werden kann,ob die gerichteten Kanten ebenfalls als ungerichtet aufgefaßt werden sollen oder nicht.(Ungerichteten Graph untersuchen).Es kann außerdem gewählt werden,ob nur ein geschlossener oder alle Eulerlinien (Unterschiede in der Reihenfolge der Kanten) gesucht werden sollen.Bei Auswahl einer Linie kann ein weiterer schneller Euler-Algorithmus gegenüber dem Backtracking-Verfahren ausgeführt werden.Danach werden ausgehend vom letzten Mausklickknoten alle geschlossenen Eulerlinien des Graphen bestimmt und rot-gestrichelt-markiert auf der Zeichenfläche als Pfade unter Ausgabe der Folge der Knoteninhalte (Werte) im Label Ausgabe1 (mit Summe und Produkt der Kantenbewertungen) angezeigt. (Kanten des Datentyps string ist dabei der Wert 1 zugeordnet.)Alle geschlossenen Eulerlinien werden außerdem mit Summe und Produkt der Kantenbewertungen im Anzeige-und Ausgabefenster dargestellt.
Falls keine Eulerlinien existieren wird eine entsprechende Mitteilung ausgegeben.Die Existenz von Eulerlinien wird vor Ablauf des Hauptalgorithmus geprüft.

Im Demo-Modus kann die Arbeitsweise des Algorithmus schrittweise und verlangsamt verfolgt werden.

Färbbarkeit:

Nach Aufruf dieses Menüs wird zunächst nach der Anzahl der vorzugebenden Farben gefragt.Vorgabewert ist die Zahl 4,die auch bei der Auswahl der Option Abbrechen gewählt wird.

Es wird dann untersucht,ob die Knoten des Graphen der Zeichenfläche mit der vorgegebenen Anzahl der Farben so gefärbt werden können,dass je zwei durch eine Kante direkt miteinander verbundene Knoten verschiedene Farben erhalten.Vor Ablauf des Algorithmus kann gewählt werden,ob alle möglichen Farbverteilungen der Knoten oder nur eine gesucht werden soll.

Die Farbverteilungen werden durch Färben der Kreislinien der Knoten sowie durch Anzeige einer Farbzahl (natürliche Zahl) als Inhalt (Wert) der Knoten dargestellt.

Die Ergebnisse werden außerdem im Anzeigefenster und Ausgabefenster dargestellt.

Im Demomodus kann der Ablauf des Algorithmus schrittweise und verlangsamt verfolgt werden.

Die kleinste Farbzahl,mit der die Knoten eines Graphen noch zulässig gefärbt werden können,heißt chromatische Zahl des Graphen.Sie kann durch Variation der vorzugebenden Farbzahl bestimmt werden.

Eulerlinie(offen):

Der Graph auf der Zeichenfläche sollte bei Aufruf dieses Menüs aus einer Komponente bestehen.Er kann gerichtete und ungerichtete Kanten enthalten.Er sollte genau zwei Knoten mit ungeradem Knotengrad (ungerichteter Graph) bzw. mit dem Knotengrad ungleich Null (positiv/negativ) (gerichteter Graph) besitzen.Zu Beginn erscheint ein Abfragefenster,in dem bestimmt werden kann,ob die gerichteten Kanten ebenfalls als ungerichtete Kanten aufgefaßt werden sollen oder nicht.(Ungerichteten Graph untersuchen).Es kann außerdem gewählt werden,ob nur eine Eulerlinie oder alle gesucht (Unterschiede in der Reihenfolge der Kanten) werden sollen. .Bei Auswahl einer Linie kann ein weiterer schneller Euler-Algorithmus gegenüber dem Backtracking-Verfahren augeführt werden.Danach werden zwischen den zwei Knoten des Graphen mit ungeradem Knotengrad bzw. Knotengrad ungleich Null (die vom Algorithmus selber bestimmt werden) alle offenen Eulerlinien des Graphen bestimmt und rot-gestrichelt-markiert auf der Zeichenfläche als Pfade unter Ausgabe der Folge der Knoteninhalte im Label Ausgabe1 (mit Summe und Produkt der Kantenbewertungen) angezeigt.(Kanten des Datentyp string ist dabei der Wert 1 zugeordnet.)

Alle offenen Eulerlinien werden außerdem mit Summe und Produkt der Kantenbewertungen im Anzeige-und Ausgabefenster dargestellt.

Falls keine offenen Eulerlinien existieren,wird eine entsprechende Mitteilung ausgegeben.Die Existenz von offenen Eulerlinien wird vor Ablauf des Hauptalgorithmus geprüft.

Im Demomodus kann die Arbeitsweise des Algorithmus schrittweise und verlangsamt verfolgt werden.

Endlicher Automat:

Der Graph der Zeichenfläche sollte bei Anwahl dieses Menüs gerichtet und zusammenhängend sein.Der Algorithmus simuliert einen endlichen,determinierten Automaten,wobei die Knoten des Graphen mittels ihres Inhalts (Werts) als Zustände und die Kanten mittels ihres Inhalts (Werts) als Eingabealphabet aufgefaßt werden.

Nach Start kann zunächst gewählt werden,ob die Zwischenzustände des Automaten zwischengespeichert (und wieder eingelesen) werden sollen oder nicht.Falls nein muß zunächst ein Knoten durch Mausklick mit der linken Taste als Anfangszustand ausgewählt werden.Danach erfolgt ebenfalls durch Mausklick mit der linken Taste die Auswahl der Endzustände,bis im Auswahlfenster angewählt wird,dass alle Endzustände als Knoten markiert wurden.Die Knoten der Anfangs-und Endzustände werden dann farbig markiert dargestellt (grün und blau).Der Anfangszustand wird zum momentanen Zustand gemacht.

Gleichzeitig erscheint das Editierfensterelement,in das der Inhalt (Wert) einer vom Anfangszustand ausgehenden Kante eingegeben werden sollte.Nach Drücken der Enter-Taste,wird der Zielknoten der Kante zum momentanen Zustandsknoten.Er wird rot markiert dargestellt.Durch weitere Eingabe von geeigneten Kanteninhalten werden jeweils immer deren Zielknoten zum momentanen Zustand gemacht und schließlich sollte einer der Endzustände als momentaner Zustandsknoten erreicht werden.Dann erscheint die Meldung,dass ein Endzustand erreicht ist,und es wird gefragt,ob der Benutzer noch weitere Eingaben vornehmen möchte,um z.B. noch in einen anderen oder nochmals in denselben Endzustand zu gelangen.

Sobald ein Ende des Algorithmus gewählt wird,wird die Folge der Inhalte (Werte) der durchlaufenden Knoten (Zustände) sowie der eingegebenen

Kanteninhalten (ausgewählte Zeichen des Eingabealphabets) im Anzeigefenster und im Ausgabefenster dargestellt.Die Folge der Inhalte der Knotenzustände wird auch synchron zum Wechsel des momentanen Zustands im Label Ausgabe2 angezeigt.Außerdem wird der durchlaufende Pfad rot-gestrichelt-markiert gezeichnet.

Durch Anwahl des Abbruch-Buttons,der sich rechts neben dem Editierfensterelement befindet,kann der Algorithmus jederzeit abgebrochen werden.

Falls das Zwischenspeichern (und Wiedereinlesen) der Zustände möglich sein soll,kann gewählt werden,ob ein neuer Graph erzeugt werden soll,ein vorhandener Graph geladen oder der vorhandene Graph übernommen und für das Verfahren benutzt werden soll.In allen drei Fällen wird ein neuer (zu den in den anderen Anwendungen inkompatibler) Graphtyp benutzt.Danach kann entweder der Graph neu erzeugt oder abgeändert werden.Durch Mausklick auf den Button Start wird dann der Algorithmusablauf wie oben beschrieben gestartet.Beim Laden eines Graphen kann gewählt werden,ob der eventuell abgebrochene und zwischengespeicherte Algorithmus fortgesetzt werden oder das Verfahren neu gestartet werden soll.

Bei der Fortsetzung des Algorithmus ist dann ein Eingabezeichen einzugeben,um in den nächsten Zustand zu gelangen.Die Graphen,die bei diesem neuen Graphtyp erzeugt werden,sind inkompatibel zu den bei allen anderen Anwendungen verwendeten Graphtypen,worauf beim Einlesen zu achten ist.(D.h. in allen anderen Anwendungen kann dieser Graph nicht verwendet werden.Der Grund liegt darin,dass hiermit eine bestimmte Art,objektorientiert durch Vererbung Graphen zu erzeugen,abzuspeichern und einzulesen demonstriert werden soll:siehe Anhang F I Beschreibung der Datenstruktur)Im übrigen läuft der Algorithmus wie im ersten Teil weiter oben beschrieben ab.

Graph als Relation:

Vor Anwahl dieses Menüs sollte der Graph der Zeichenfläche gerichtet sein.Es wird dann nacheinander zunächst abgefragt,ob die Reflexivität und/oder die Transitivität untersucht und die reflexive und/oder die transitive Hülle bestimmt werden sollen.Außerdem kann bestimmt werden,ob der ursprüngliche Graph wiederhergestellt werden sollte oder ein veränderter Graph zum Schluss entsteht.
Der Graph wird bei dieser Anwendung als Relation aufgefaßt,wobei die gerichteten Kanten die Relation zwischen den Knoten (d.h. den Knoteninhalten bzw. Werten) beschreiben.Im Falle der Existenz einer gerichteten Kante vom Knoten mit dem Inhalt x zum Knoten mit dem Inhalt y gilt also,dass $(x,y)\in R$ ist.

Je nach obiger Auswahl wird dann der Graph (die Relation) auf Reflexivität und/oder Transitivität untersucht.Wenn der Graph reflexiv ist,muß eine gerichtete Schlinge zu jedem Knoten existieren.Fehlende Schlingen werden dann,wo sie fehlen,zur Erzeugung der reflexiven Hülle grün markiert hinzugefügt.

In gleicher Weise wird die transitive Hülle erzeugt,indem rot-markierte Kanten überall dort zwischen zwei Knoten hinzugefügt werden,wo sie zwecks Transitivität notwendig sind.

Außerdem wird,falls möglich,eine globale Ordnung und Rangfolge auf der Menge der Knoten bestimmt,indem jedem Knoten eine Rangfolgezahl von 1 bis zur Anzahl der Knoten zugeordnet wird (und als Inhalt/Wert angezeigt wird).Dies ist immer dann möglich,wenn die durch den Graphen dargestellte Relation antisymmetrisch ist.(Der Anfangsknoten einer Kante liegt dabei in der Rangfolge stets vor dem Endknoten einer Kante.)

Es wird geprüft,ob die durch den Graph dargestellte Relation symmetrisch bzw. antisymmetrisch,reflexiv und/oder transitiv ist und die Resultate sowie
die eventuell bestimmte Rangfolge der Knoten als Ordnungsrelation werden im Anzeige-und Ausgabefenster dargestellt.

Falls gewählt wurde,dass der ursprüngliche Graph wiederhergestellt wird,werden die Änderungen durch Erzeugen der reflexiven und/oder transitiven Hülle wieder rückgängig gemacht.Ansonsten bleiben sie nach Beendigung des Algorithmus erhalten.

Im Demomodus kann das Verfahren zur Bestimmung der Rangfolge der Knoten bezüglich der Ordnungsrelation schrittweise und verlangsamt verfolgt werden.

Maximaler Netzfluss:

Der Graph auf der Zeichenfläche sollte bei Aufruf dieses Menüs nur gerichtete Kanten aufweisen,kreisfrei und zusammenhängend sein und genau einen Quellen-und Zielknoten aufweisen,von dem nur Kanten ausgehen bzw. in den nur Kanten einlaufen.

Vom Algorithmus wird der Quellen-und Zielknoten selbstständig erkannt.Danach wird der maximale Netzfluss vom Quellen- zum Zielknoten nach dem Verfahren von Ford-Fulkerson bestimmt.Die vorgegebenen (positiven) Kanteninhalte werden dabei als obere Schranken des Flusses längs dieser Kante aufgefaßt.(Die unteren Schranken sind Null.)

Der maximale Netzfluss durch jede Kante wird auf der Zeichenfläche in der Form Schranke:Fluss als Inhalt/Wert jeder Kante bei Beendigung des Algorithmus angezeigt,als auch werden die Ergebnisse im Anzeigefenster und Ausgabefenster dargestellt.Der maximale Gesamtfluss wird in einem Fenster ausgegeben.

Im Demomodus kann die Arbeitsweise des Algorithmus von Ford-Fulkerson verlangsamt und schrittweise verfolgt werden.

Maximales Matching:

Nach Aufruf dieses Menüs kann zunächst gewählt werden,ob vor Beginn des Hauptalgorithmus ein Anfangsmatching erzeugt werden soll oder nicht.Dieses Matching wird so erzeugt,dass bei allen Kanten des Graphen nacheinander geprüft wird,ob ihre Anfangs-und Endknoten noch isolierte Knoten sind,und wenn dies der Fall ist,wird die Kante dem Matching hinzugefügt.Wenn ein Anfangsmatching gewählt wird,benötigt der anschließende Hauptalgorithmus nicht mehr soviel Zeit,zur Bestimmung eines maximalen Matchings.

Nach dieser Abfrage wird der Hauptalgorithmus gestartet,der auf dem Algorithmus des Suchen eines erweiterten Weges beruht.

Im Demomodus kann die Arbeitsweise des Verfahrens schrittweise und verlangsamt verfolgt werden.Dabei werden (Anwärter-)Kanten,die beim Umfärben eine neue Kante des Matchings ergeben mit Bezeichnung A und blau markiert dargestellt,Kanten,die dem bisherigen Matching angehörten und beim Umfärben aus dem Matching entfernt werden,werden mit Bezeichnung L und grün markiert dargestellt und Kanten die zum (alten oder neuem erweiterten) Matching gehören,werden mit Bezeichnung M und rot markiert dargestellt.Das Suchen und Umfärben des erweiterten Weges kann im Demomodus anschaulich beobachtet werden.

Nach Beendigung des Algorithmus wird die Zahl der Kanten des Matchings in einem Fenster angezeigt,und es wird eventuell angegeben,ob das Matching perfekt ist (keine isolierten Knoten mehr).Die Kanten des Matchings werden auf der Zeichenfläche rot markiert und mit der Buchstabenbezeichnung M versehen angezeigt.Im Anzeige-und Ausgabefenster werden alle Kanten des Matchings mit Anfangs-und Endknoten(-Inhalt/Wert) angegeben.

Gleichungssystem:

Beim Start dieser Anwendung sollte der Graph gerichtet sein und keine Parallelkanten enthalten.Die Kanteninhalte sollten vom Typ Real (Extended) sein.

Der Algorithmus löst ein Gleichungssystem der Form:
$a_{11}x_1+....+a_{1n}=b_1$

.

.

.

$amn+.....+amn=b_m$

Dabei werden die Werte der b_i als Knoteninhalte (Werte) und die Werte der a_{ij} als Inhalt (Wert) der Kante vom i.tem zum j.ten Knoten dargestellt.Die Schlingenkanten besitzen also die Werte der Variablen a_{ii}.Nicht vorhandene Kanten zwischen verschiedenen Knoten,die nach dem obigen Schema bei einem schon vorgegebenen Graph benötigt werden,werden selbstständig erzeugt und haben als Verbindung von verschiedenen Knoten den Inhaltswert 0 (vom Typ Real) und die Schlingen den Wert 1 (vom Typ Real).(Kanten vom Typ string besitzen automatisch der Wert Eins.)Den einzelnen Knoten werden die Lösungsvariablen $x_1...x_n$ zugeordnet.

Das Verfahren arbeitet nach dem Algorithmus von Mason.Nach Anwahl des Menüs sind die Knoten des Graphen nacheinander durch Anklicken mit der linken Maustaste in beliebiger Reihenfolge zu löschen,wobei nach jedem Löschen ein neuer Graph entsteht,in dem eine Lösungsvariable eliminiert ist,der jedoch bezüglich der anderen Lösungsvariablen dieselbe Lösungsmenge hat.Die Kanten-und Knoteninhalte (Werte) werden dabei äquivalent bei jedem Löschen eines Knotens (und der damit inzidenten Kanten) angepaßt,so dass ein Gleichungssystem mit derselben Lösungsmenge bezüglich der verbleibenden Knoten entsteht.

Bei der Löschung des letzten Knotens wird der Wert für dessen Lösungsvariable x_i ermittelt und in einem Fenster angezeigt.Danach wird der ursprüngliche Graph wieder hergestellt.Durch mehrfache Wiederholung des Verfahrens lassen sich so alle Lösungen des Gleichungssystems ermitteln,sofern das System eindeutig lösbar ist.Ansonsten bricht das Verfahren beim Löschen eines der Knoten des Graphen mit dem Hinweis auf keine oder keine eindeutige Lösung ab.Sobald die gewünschten Lösungsvariablen berechnet sind,wählt man im Abfragefenster,das nach Anzeige einer Lösungsvariablen erscheint,auf die Frage Weitere Lösung bestimmen die Option Nein.

Die ermittelten Lösungen werden im Label Ausgabe2 sowie nach Beendigung des Algorithmus im Anzeige-und Ausgabefenster angezeigt.

Im Demomodus kann die Berechnung der neuen Knoten-und Kanteninhalte nach dem Algorithmus von Mason schrittweise und verlangsamt verfolgt werden.

Zum Schluß des Algorithmus wird der ursprüngliche Graph wiederhergestellt.

Markovkette (abs):

Beim Start dieses Menüs sollte der Graph eine absorbierende Markovkette darstellen,bei der die Übergangswahrscheinlichkeiten und die mittleren Schrittzahlen von jedem beliebigen Zustand (Knoten) in ausgewählte Zustände (Knoten) des Randes ermittelt werden sollen.

Dies bedeutet,dass die Summe der Übergangswahrscheinlichkeiten aller von einem Knoten (Zustand) ausgehenden Kanten gleich Eins sein muß,und die Inhalte (Werte) aller Kanten,die die Übergangswahrscheinlichkeiten zwischen zwei Zuständen (Knoten) darstellen,vom Datentyp Real sein sollten und zwischen 0 und 1 liegen müssen.Bei Nichtvorliegen dieser Voraussetzungen werden entsprechende Fehlermeldungen ausgegeben.Der Graph sollte außerdem gerichtet sein.

Die Zustände (Knoten) des Randes (die nicht Auswahlzustände sind) werden dadurch gekennzeichnet,dass von Ihnen keine weiteren Kanten ausgehen.Die Auswahlzustände des Randes,in die die Übergangswahrscheinlichkeiten von den übrigen Zuständen (Knoten) aus ermittelt werden sollen,werden durch eine Schlingenkante mit dem Wert 1 markiert.

Der Algorithmus beruht auf einem Algorithmus von A.Engel (Lit 15),bei dem der Graph als Spielbrett,auf dem Steine gezogen werden,aufgefaßt wird (Wahrscheinlichkeitsabakus).

Nach Start des Algorithmus wird gefragt,ob lediglich die Übergangswahrscheinlichkeit und die mittlere Schrittzahl für einen oder für alle Knoten bestimmt werden sollen.

Im ersten Fall werden nur die Wahrscheinlichkeit und die mittlere Schrittzahl für den letzten Mausklickknoten bestimmt,sonst für alle Knoten (Zustände).

Außerdem wird die Genauigkeit als Stellenzahl hinter dem Komma,mit der intern gerechnet werden soll (zu unterscheiden von der Anzeigegenauigkeit), abgefragt (1 bis 5 Stellen mit einem Vorgabewert von 3 Stellen).

Wird mit zu großer Genauigkeit gerechnet,dauert die Bestimmung von Ergebnissen insbesondere bei umfangreichen Graphen und,wenn die Ergebnisse für alle Knoten berechnet werden sollen,länger.

Nach Abschluß des Algorithmus werden die Übergangswahrscheinlichkeiten in die Auswahlzustände vom aktuellen Knoten aus in der Form Knoteninhalt:Wahrscheinlichkeit auf der Zeichenoberfläche als Knoteninhalt (Wert) angezeigt (bei der Auswahl „ein Knoten“ nur bei einem Knoten).Durch Anklikken der Knoten mit der linken Maustaste werden die Ergebnisse darüberhinaus in einem sich öffnenden Fenster ausgegeben und noch zusätzlich die mittlere Schrittzahl dargestellt,um vom aktuellen Knoten (Zustand) in die Auswahlzustände zu gelangen.Außerdem erscheint ein (Message-)Fenster,in dem die Übergangswahrscheinlichkeit und die mittlere Schrittzahl für den letzten Mausklickknoten angezeigt wird.
Darüberhinaus werden die Ergebnisse im Anzeige-und Ausgabefenster angezeigt.

Im Demomodus kann der Ablauf des Algorithmus,d.h. die Spielzüge bzw. das Ziehen der Steine durch Anzeige der gezogenen Steinezahlen in den Labeln Ausgabe1 und Ausgabe2 verlangsamt und schrittweise verfolgt werden.

<u>Markovkette (stat):</u>

Beim Start dieses Menüs sollte der Graph eine stationäre Markovkette darstellen,bei der die stationären Grenz-Wahrscheinlichkeiten und die mittleren Grenz-Schrittzahlen von jedem beliebigen Zustand (Knoten) in dieselben Zustände (Knoten) wieder zurückzukehren (nach unendlich vielen Durchführungen der zu Grunde liegenden Zufallsprozesse),ermittelt werden sollen.

Dies bedeutet,dass die Summe der Übergangswahrscheinlichkeiten aller von einem Knoten (Zustand) ausgehenden Kanten gleich 1 sein muß,und die Inhalte aller Kanten,die die Übergangswahrscheinlichkeiten zwischen zwei Zuständen (Knoten) darstellen,vom Datentyp Real sein sollten und zwischen 0 und 1 liegen müssen.Bei Nichtvorliegen dieser Voraussetzungen werden entsprechende Fehlermeldungen ausgegeben.Der Graph sollte außerdem gerichtet sein.

Randzustände oder Auswahlzustände wie bei den absorbierenden Markovketten gibt es hierbei nicht.

Der Algorithmus beruht auf einem (vom Algorithmus Markovkette (abs) verschiedenen) Algorithmus von A.Engel (Lit 15),bei dem der Graph als Spielbrett,auf dem Steine gezogen werden, aufgefaßt wird (Wahrscheinlichkeitsabakus).Nach Start des Algorithmus wird gefragt,ob lediglich die Übergangswahrscheinlichkeit und die mittlere Schrittzahl für einen oder für alle Knoten bestimmt werden sollen.

Im ersten Fall werden nur die Wahrscheinlichkeit und die mittlere Schrittzahl für den letzten Mausklickknoten bestimmt,sonst für alle Knoten (Zustände).

Außerdem wird die Genauigkeit als Stellenzahl hinter dem Komma,mit der intern gerechnet werden soll (zu unterscheiden von der Anzeigegenauigkeit), abgefragt (1 bis 5 Stellen mit einem Vorgabewert von 3 Stellen).

Wird mit zu großer Genauigkeit gerechnet,dauert die Bestimmung von Ergebnissen insbesondere bei umfangreichen Graphen und wenn die Ergebnisse für alle Knoten berechnet werden sollen,länger.

Nach Abschluß des Algorithmus werden die Grenzwahrscheinlichkeiten vom aktuellen Knoten aus in diesen Zustand zurückzugelangen in der Form Knoteninhalt:Wahrscheinlichkeit auf der Zeichenoberfläche als Knoteninhalt (Wert) angezeigt (bei der Auswahl „ein Knoten" nur bei einem Knoten).

Durch Anklicken der Knoten mit der linken Maustaste werden die Ergebnisse darüberhinaus in einem sich öffnenden Fenster angezeigt und noch zusätzlich die mittlere Schrittzahl dargestellt,um vom aktuellen Knoten (Zustand) in diesen Zustand zurück zu gelangen.Außerdem erscheint ein Fenster,in dem die Grenzwahrscheinlichkeit und die mittlere Schrittzahl für den letzten Mausklickknoten angezeigt werden.

Darüber hinaus werden die Ergebnisse im Anzeige-und Ausgabefenster dargestellt.

Im Demomodus kann der Ablauf des Algorithmus,d.h. die Spielzüge bzw. das Ziehen der Steine mittels Anzeige der gezogenen Steinezahlen in den Labeln Ausgabe1 und Ausgabe2 schrittweise und verlangsamt verfolgt werden.

Graph reduzieren:

Bei Aufruf dieses Menüs sollte der Graph gerichtet sein.Mittels des Algorithmus können die Übergangswahrscheinlichkeiten von Zuständen (Knoten) einer absorbierenden Markovkette in die Auswahlzustände des Randes oder aber der Signalfluss von Zuständen (Knoten) in die (vorzugebenden) Auswahlzustände (Knoten) eines Signalflussgraphen bestimmt werden,indem die Knoten nacheinander samt den mit ihnen inzidenten Kanten aus dem Graph gelöscht werden, und dabei jeweils ein äquivalenter reduzierter Graph entsteht, dessen Knoten (Zustände) dieselben Übergangswahrscheinlichkeiten bzw. Signalflüsse in die Auswahlzustände aufweisen,wie es beim ursprünglichen Graph der Fall ist.

Im Fall einer absorbierenden Markovkette muß die Kantensumme der ausgehenden Kanten jedes Knotens den Wert 1 haben,und der Inhalt (Wert) jeder Kante des Graphen zwischen 0 und 1 liegen.Der Typ der Kanten sollte Real (Extended) sein.

Die Auswahlknoten werden durch eine Schlingenkante mit dem Wert Eins oder aber beim Signalflussgraphen durch das Zeichen ‘q‘ als Inhalt (Wert) gekennzeichnet(Quelle).

Zu Beginn des Algorithmus wird abgefragt,ob der Graph als Markovkette (oder als Signalflussgraph) aufgefaßt werden sollte.

Nach dem Start wird zunächst eine Signalflussgraphreduzierung von Parallelkanten und Schlingen (nach Mason) vorgenommen.

Danach sollte wie beim Algorithmus Gleichungssystem ein Knoten nach dem anderen gelöscht werden,wobei jeweils bezüglich der Übergangswahrscheinlichkeiten äquivalente reduzierte Graphen entstehen.

Die Auswahlknoten und Randknoten können dabei erst dann gelöscht werden,wenn nur noch ein letzter innerer Knoten vorhanden ist.Dann können an Hand der angezeigten Kantenwahrscheinlichkeiten schon die Übergangswahrscheinlichkeiten in die Auswahl-und Nichtauswahlzustände abgelesen werden.Beim Löschen des letzten Knoten wird die Übergangswahrscheinlichkeit bzw. der Signalfluss des Knotens in die Auswahlzustände in einem Fenster angezeigt.

Danach wird,wenn einer entsprechenden Abfrage auf Wiederholung des Verfahrens zugestimmt wurde,der ursprüngliche Graph wiederhergestellt,und mittels desselben Verfahrens kann die Übergangswahrscheinlichkeit bzw. der Signalfluss eines anderen Knotens (Zustands) bestimmt werden.

Die Ergebnisse werden im Label Ausgabe 2 und nach Ende des Algorithmus im Anzeige-und Ausgabefenster angezeigt.Außerdem werden sie in der Form Knoteninhalt:Wahrscheinlichkeit bzw. Knoteninhalt:Signalfluss als Inhalte (Werte) der Knoten auf der Zeichenfläche angezeigt.

Im Demomodus kann der Ablauf des Algorithmus Graph reduzieren,der u.a. den Algorithmus Gleichungssystem benutzt,schrittweise und verlangsamt verfolgt werden:inbesondere die Transformation des Graphen in einen für das Lösen eines Gleichungssystems geeigneten Graphen und die entsprechende Rücktransformation,das Lösen des Gleichungssytems sowie das Reduzieren von Parallelkanten und Schlingen.

Zum Schluss des Algorithmus wird der ursprüngliche Graph wiederhergestellt. Dieser Algorithmus gestattet also mittels eines anderen Verfahrens gegenüber dem Menü Markovkette (abs) die Bestimmung der Übergangswahrscheinlichkeiten von Markovketten in die Randzustände bzw. Auswahlzustände von einem bestimmten Zustand (Knoten) aus.(u.a. werden die Mittelwertregeln benutzt.)

Minimale Kosten:

Dieser Algorithmus von Busacker und Gowen bestimmt zum vorgegebenen Graph einen Kosten-minimalen oder Kosten-maximalen zulässigen Fluss,wobei die Kanteninhalte als obere Schranken aufgefaßt werden und die unteren Schranken 0 sind.Die Kosten müssen als Kosten pro Flusseinheit für jede Kante eingegeben werden.Der Fluss fließt aus einem Quellenknoten zu einem Senkenknoten des Graphen.

Der Graph sollte gerichtet sein,und es sollten genau ein Quellenknoten,der nur ausgehende Kanten und ein Senkenknoten,der nur eingehende Kanten besitzt,vorhanden sein.Die beiden Knoten werden beim Start des Algorithmus automatisch erkannt,und ihr Knoteninhalt (Wert) wird angezeigt.Danach kann gewählt werden,ob ein maximales Kostenproblem statt eines minmalen Kostenproblems gelöst werden soll,und ob nur ganzzahlig gerechnet werden soll oder nicht.

Anschließend sind die Kosten für jede Kante pro Flusseinheit einzugeben.Dies geschieht durch Anklicken der Kanten mit der linken Maustaste.Danach öffnet sich ein Eingabefenster,in das die Kosten der Kante pro Flusseinheit eingegeben werden können.Nicht angewählte Kanten haben automatisch die Kosten 0.Wird auf eine Stelle der Zeichenoberfläche geklickt,an der sich keine Kante befindet,erscheint ein Auswahlfenster,in dem bestimmt werden kann,ob noch weitere Kantenkosten eingegeben werden sollen oder ob die Eingabe der Kosten beendet werden soll.

Nach Eingabe aller gewünschten Kosten wird nach dem Fluss durch den Quellenknoten gefragt,der in einem Fenster einzugeben ist.Danach berechnet der Algorithmus alle notwendigen Werte,und es werden zunächst die minimalen bzw. maximalen Gesamtkosten des Flusses durch alle Kanten des Graphen als Inhalte (Werte) der Kanten angezeigt.

Danach wird noch einmal der vorgegebene Fluss (der eventuell nicht realisiert werden kann) ausgegeben,und der maximal zu erreichende Fluss angezeigt.Dieser Fluss ist dann gleichzeitig die Lösung des Maximalflussproblems.Falls der vorgegebene Fluss nicht erreicht werden kann,beziehen sich die Lösungen des Problems auf den maximalen Fluss.

Gleichzeitig mit der Ausgabe von Kosten und Fluss in einem Fenster werden auf der Zeichenfläche als Kanteninhalte (Werte) die Kosten für jede Kante und der Fluss durch jede Kante angezeigt.

Zum Schluss kann gewählt werden,ob der Algorithmus mit denselben Kosten der Kanten pro Flusseinheit (aber evtl. anderem Fluss durch den Quellenknoten) nochmals wiederholt werden soll oder nicht.

Nach Ende des Algorithmus werden im Anzeige-und Ausgabefenster alle Kanten mit Anfangs-und Endknoteninhalt (Wert),der Fluss durch diese Kanten,die (Fluss-)Kosten dieser Kanten,die obere Schranke dieser Kanten und die Kosten pro Flusseinheit angezeigt.Außerdem wird der vorgegebene Fluss,der erreichte Fluss und die Gesamtkosten des Flusses durch alle Kanten angegeben.Im Demomodus kann der Ablauf des Algorithmus nach dem Verfahren von Busacker und Gowen in Einzelheiten schrittweise und verlangsamt verfolgt werden.

Transportproblem:

Dieser Algorithmus löst auf dem vorgegebenen Graph,der gerichtet sein soll, das Transport-oder das Hitchcockproblem.Beim Transportproblem und Hitchcockproblem sind n Quellknoten,von denen nur Kanten ausgehen und m Senkenknoten,in die nur Kanten einlaufen,vorgegeben.In jedem der Quellenknoten i befinden sich n_i Flusseinheiten eines Transportguts.

Von jedem Quellenknoten sind diese n_i Flusseinheiten insgesamt zu den Senkenknoten j zu transportieren,bei denen jeweils ein Bedarf von m_j Flusseinheiten des Transportgutes besteht,so daß Angebots- und Bedarfszahlen jeweils berücksichtigt und die Bedarfszahlen möglichst abgedeckt werden sollten.Jeder Kante werden Kosten pro Flusseinheiten zugeordnet.

Der Transport ist so auf Kantenpfade von den Quellen zu den Senkenknoten zu verteilen,dass die Gesamtkosten minimal bzw. maximal sind.Beim Transportproblem sind den Kanten jeweils noch obere Schranken für den Fluss durch die Kanteninhalte (Werte) zugeordnet.Die unteren Schranken sind gleich Null.Beim Hitchockproblem sind die oberen Schranken unendlich groß.

Nach Anwahl des Menüs wird zunächst gefragt,ob ein maximales Kostenproblem statt eines minimalen Kostenproblems gelöst werden werden,und ob nur ganzzahlig gerechnet werden soll (oder jeweils nicht).Außerdem ist zu entscheiden,ob das Transportproblem oder das Hitchcockproblem gelöst werden soll.Nach Auswahl der gewünschten Optionen sind die Kosten der Kanten pro Flusseinheit unter Anwahl der Kanten mittels Klick der linken Maustaste auf die jeweilige Kante in einem Eingabefenster einzugeben.(Kanten,die nicht angewählt werden,erhalten dabei automatisch die Kosten 0.)Durch Klick auf eine Stelle der Zeichenfläche,an der sich keine Kante befindet,erscheint ein Auswahlfenster,in dem das weitere Eingeben von Kosten der Kanten oder aber das Ende des Eingebens gewählt werden können.

Bei Wahl von Ende werden auf der Zeichenfläche die eingegebenen Kosten pro Flusseinheit aller Kanten als Kanteninhalte (Werte) angezeigt.Danach erscheinen Eingabefenster für alle Quellen-und danach für alle Senkenknoten,in die die dortigen Produktionseinheiten bzw. Abnahmeeinheiten als Fluss durch diese Knoten eingegeben werden sollen.Als Vorgabewert wird hierbei 1 angeboten.

Anschließend wird geprüft,ob der Geamtquellenfluss gleich oder ungleich dem Gesamtsenkenfluss ist und eventuell wird bei Ungleichheit eine entsprechende Warnung ausgegeben.Dann wird eine Lösung mit dem kleinsten der beiden Gesamtflüsse gesucht.

Anschließend bestimmt der Algorithmus durch Zurückführung auf das Verfahren Minimale Kosten eine Lösung.Man sieht dabei auf der Zeichenfläche,dass ein neuer Quellen-und Senkenknoten mit entsprechenden Kanten,die zu den vorgegebenen Quellen-bzw. Senkenknoten führen und deren Schranken aus den vorgegebenen Produktions-und Abnahmemengen(-Flüsssen) (mit den Kosten 0) bestehen,erzeugt werden,so dass der Algorithmus Minimale Kosten anwendbar ist.(Der neue Quellen-bzw. Senkenknoten wird nach Ende des Algorithmus Minimale Kosten jeweils wieder gelöscht.)

Anschließend werden die Gesamtkosten des Flusses durch alle Kanten des vorgegebenen Graphen und danach auch der vorgegebene Fluss,der dem Gesamtfluss durch alle Quellenknoten bzw. durch alle Senkenknoten und der damit den (Gesamt-) Produktions-bzw. Abnahmemengen entspricht,ausgegeben.

Gleichzeitig werden auf der Zeichenfläche die Kosten des Flusses in jeder Kante und der Fluss durch jede Kante als Kanteninhalte (Werte) angezeigt. Nach Ende des Algorithmus werden im Anzeige-und Ausgabefenster alle Kanten mit Anfangs-und Endknoteninhalt (Wert bzw. Bezeichner),der Fluss durch diese Kanten,die (Fluss-)Kosten dieser Kanten, die obere Schranke dieser Kanten und die Kosten pro Flusseinheit angezeigt.Außerdem wird der vorgegebene Fluss, der erreichte Fluss und die Gesamtkosten des Flusses durch alle Kanten angegeben.

Im Demomodus können der Ablauf des Algorithmus nach Busacker und Gowen (vgl. Algorithmus Minimale Kosten) sowie das Erzeugen und Löschen eines neuen Quellen-und Senkenknoten schrittweise,verlangsamt und detailiert verfolgt werden.

Optimales Matching:

Der Algorithmus dieses Menüs bestimmt ein optimales Matching auf einem bipartiten Graph.Dazu wird jeder Kante eine nichtnegative Bewertung,nämlich die Kosten zugeordnet.Ein optimales Matching ist dann ein Kostenminimales bzw. Kostenminimales Matching bezüglich der Gesamtzahl aller Kanten,die zum Matching gehören im Vergleich zu allen anderen Matchingauswahlen auf dem Graphen.

Der Graph sollte also gerichtet und bipartit sein.Andernfalls werden Fehlermeldungen ausgegeben.

Durch die Eigenschaften gerichtet und bipartit sind automatisch Quellen-und Senkenknoten vorgegeben,von denen nur Kanten ausgehen und in die nur Kanten einlaufen.Der Algorithmus führt das Problem dann zurück auf das Lösen eines Hitchcockproblems mit Produktions-und Abnahmemengen (-Flüssen) in den Quellen-und Senkenknoten von jeweils der Größe 1.

Nach Start des Algorithmus wird zunächst gefragt,ob ein Kostenminimales oder Kostenmaximales Matching gesucht werden soll.Danach sind durch Mausklick mit der linken Taste auf die Kanten jeweils die Kosten der Kanten pro Flusseinheit einzugeben.Durch Klick auf eine Stelle der Zeichenfläche,an der sich keine Kante befindet,erscheint ein Auswahlfenster,in dem das weitere Eingeben von Kosten der Kanten oder aber das Ende des Eingebens gewählt werden können.

Bei Wahl von Ende werden auf der Zeichenfläche die eingegebenen Kosten pro Flusseinheit aller Kanten als Kanteninhalte (Werte) angezeigt.

Danach werden den Quellen-und Senkenknoten des bipartiten Graphen je die Produktions-und Abnahmemengen 1 zugeordnet und unter Erzeugung eines neuen (Gesamt-)Quellen-und Senkenknoten mit entsprechenden Kantenschranken (1) und Kantenkosten (0) wird dann das Problem auf den (obigen beschriebenen) Hitchcockalgorithmus mit ganzahliger Rechnung zurückgeführt.(Die neu erzeugten Quellen-und Senkenknoten werden nach Berechnung aller Ergebnisse wieder gelöscht.)

Nach Ermittlung einer Lösung werden zunächst die Gesamtkosten des Flusses (des Matchings) ausgegeben,und anschließend werden der vorgegebene Fluss (gleich Gesamtzahl der Quellenknoten) und der erreichte Fluss (gleich Minimalzahl der Summe der Quellenknoten oder der Senkenknoten) in Fenstern angezeigt.Gleichzeitig werden auf der Zeichenfläche die Flusskosten jeder Kante sowie der Fluss durch jede Kante (entweder 0 oder 1) als Kanteninhalte (Werte) angegeben.

Danach werden alle Kanten mit dem Fluss 0 gelöscht,so daß nur die Kanten mit dem Fluss 1 übrigbleiben.Diese Kanten sind die Kanten des Optimalen Matchings.Der ursprüngliche Graph wurde zwischengespeichert und wird nach

erneuter Anwahl eines beliebigen Menüs der Menüleiste bzw. des Buttons Bild wiederhergestellt.

Nach Ende des Algorithmus werden im Anzeige-und Ausgabefenster alle Kanten mit Anfangs-und Endknoteninhalt (Wert) bzw. Bezeichner,der Fluss durch diese Kante,die (Fluss-)Kosten dieser Kanten,die obere Schranke dieser Kante (gleich 9E31,entspricht unendlich) und die Kosten pro Flusseinheit angezeigt.Außerdem wird der vorgegebene Fluss,der erreichte Fluss und die Gesamtkosten des Flusses durch alle Kanten angegeben.Die Kanten des optimalen Matching sind die Kanten mit dem Fluss 1.

Im Demomodus können der Ablauf des Algorithmus nach dem Verfahren von Busacker und Gowen (vgl. Algorithmus Minimale Kosten und Hitchcockproblem) sowie das Erzeugen und Löschen eines neuen Quellen-und Senkenknoten als auch das Setzen der Schranken der zu diesen Kanten gehörenden Kanten schrittweise und detailliert verfolgt werden.

<u>Chinesischer Briefträger</u>

Dieser Algorithmus löst das Chinesische Briefträgerproblem auf einem gerichteten Graph,dessen Kanten eine Kantenbewertung in Form der Kanteninhalte (Werte) aufweisen sollten.(Kanten des Typs string zählen dabei mit dem Wert 1.)

Auf dem Graph wird dabei ein Kantensummenlängen (Summe der Bewertungen der Kanten) minimaler Rundpfad,der zum Ausgangsknoten zurückführt,gesucht,wobei jede Kante mindestens einmal durchlaufen wird und einige Kanten eventuell mehrfach.(Der Briefträger soll eine möglichst geringe Strecke zum Ausgangspunkt zurücklegen.)Falls es möglich ist,den Pfad so zu wählen,dass keine Kante doppelt durchlaufen wird,ist der Rundpfad eine Eulerlinie.

Die Lösung des Problems kann zurückgeführt werden auf die Bestimmung des Kostenminimalen Fluss eines (ganzahligen) Hitchcockproblems,wobei die Kosten die Bewertungen der Kanten sind.Die Quellenknoten sind dabei die Knoten negativen (gerichteten) Knotengrades und die Senkenknoten die Knoten positiven (gerichteten) Knotengrades,und der Fluss durch diese Knoten (Produktions- und Abnahmemengen) ist gleich dem des (absolut genommenen) Knotengrads.

Wenn das Kostenminimale Flussproblem gelöst ist,werden zu jeder Kante soviele Parallelkanten neu erzeugt,wie der (ganzzahlige) Fluss angibt und in den Graph eingefügt.Auf diesem neuen Graphen wird dann eine Eulelinie (nach dem Algorithmus Eulerlinie s.o.) gesucht.Für den ursprünglichen Graphen bedeutet dies,dass die Kanten,die nach dem Einfügen Parallelkanten besitzen,dann so oft mehrfach durchlaufen werden,wie im neuen Graph Parallelkanten vorhanden sind.

Nach Start des Algorithmus kann insbesondere im Demomodus verfolgt werden,wie ein Hitchcockproblem unter Erzeugung eines neuen (Gesamt-) Quellen- und Senkenknoten mit entsprechenden Kanten(-Werten) mit den Kantenbewertungen (Kanteninhalte) des ursprünglichen Graphen als Kosten gelöst wird.(Die neu erzeugten Quellen-und SenkenKnoten werden anschließend wieder gelöscht.)

Anschließend wird der Gesamtfluss durch diesen Quellen-und Senkenknoten in einem Fenster angezeigt,welcher gleich der Hälfte der Anzahl der neu erzeugten Parallelkanten ist.Zuvor wurden die unproduktiven Kosten ausgegeben,welche der Summe der Kosten des Flusses auf den zusätzlichen Parallelkanten (die die gleichen Kosten pro Flusseinheiten wie ihre Orginale besitzen) als auch der zusätzlich vom Briefträger auf den mehrfach durchlaufenden Kanten zurückgelegten Pfadlänge entsprechen.

Schließlich wird auf den durch die Parallelkanten ergänzten Graph der Algorithmus Eulerlinie angewendet.Dazu wird gefragt,ob eine oder alle Lösungen gesucht werden sollen.Der/die Eulerlinie(n) werden rot-gestrichelt markiert als Pfade auf der Zeichenoberfläche angezeigt,und einer der Pfade wird als Folge von Kanteninhalten mit Kantensumme und Produkt in einem Fenster

angezeigt.Die Kantensumme ist die Gesamtpfadlänge,die der Briefträger zurücklegen muß.(Die unproduktiven Kosten sind die Länge der mehrfach durchlaufenden Kanten.)

Nach Ende des Algorithmus werden im Anzeige-und Ausgabefenster alle Kanten mit Anfangs-und Endknoteninhalt,der Fluss durch diese Kanten, die (Fluss-) Kosten dieser Kanten,die obere Schranke dieser Kante (gleich 9E31,entspricht unendlich) und die Kosten pro Flusseinheit angezeigt.Außerdem werden der vorgegebene Fluss,der erreichte Fluss und die Gesamtkosten des Flusses durch alle Kanten angegeben.

Im Demomodus können der Ablauf des Algorithmus nach dem Verfahren von Busacker und Gowen gemäß dem zu lösenden Hitchcockproblems,das Erzeugen und Löschen eines neuen Quellen-und Senkenknotens,das Setzen der Schranken der zu diesen Knoten gehörenden Kanten und der Ablauf des Eulersalgorithmus detailliert verfolgt werden.

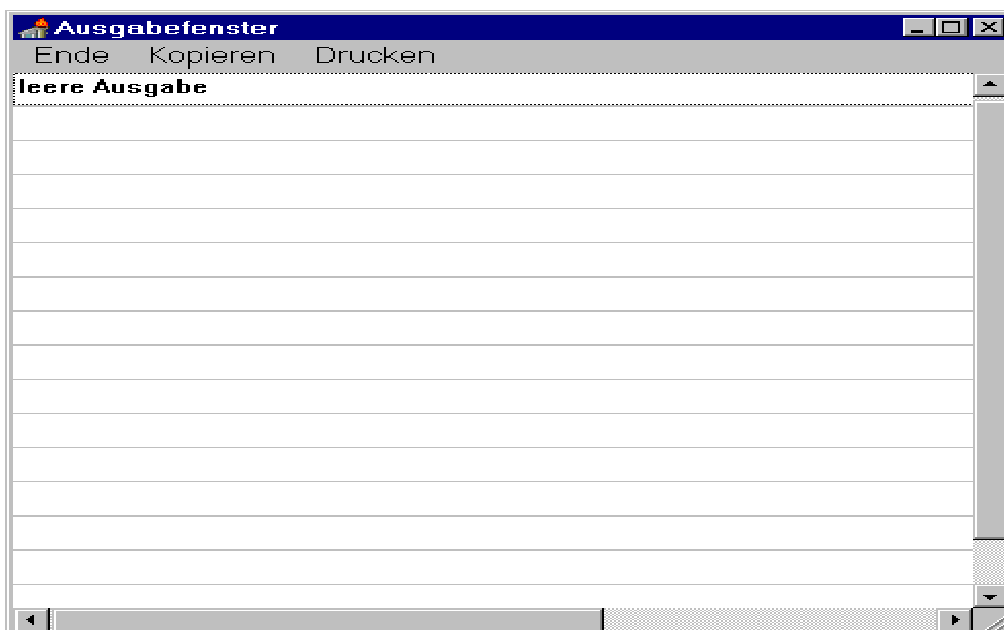
Das Briefträgerproblem kann keine Lösung haben,wenn der ermittelte Fluss des Hitchcockproblems gleich 0 ist.Dies wird durch eine entsprechende Meldung angezeigt.Dann läßt sich auch das Eulerproblem nicht lösen,es sei denn,der ursprüngliche Graph besitzt schon eine Eulerlinie.

Menü Ausgabe:

Das Menü Ausgabe besteht aus den Untermenüs Ausgabefenster, Abbruch an (aus), Demo an(aus), Pausenzeit.

Ausgabefenster:

Durch Anwahl dieses Menüs mit der linken Maustaste öffnet sich das Ausgabefenster.Es dient zur Anzeige von Ergebnissen der Algorithmen der Menüs Eigenschaften,Pfade und Anwendungen.Wenn keine Ergebnisse vorhanden sind, wird „leere Ausgabe" angezeigt.Das Ausgabefenster enthält die Menüs Ende zum Schließen des Ausgabefensters,Drucken zur Textausgabe der Ergebnisse auf einem Drucker,der in einem Druckerauswahlfenster gewählt werden kann, sowie Kopieren zum Kopieren des Textes des Ausgabefensters in die Zwischenablage von Windows.Die Anzeigeergebnisse stehen so auch anderen Anwendungen zur Verfügung.



Abbruch an (aus):

Nach Anwahl dieses Menüs mit der linken Maustaste wird eine gerade laufende Anwendung der Menüs Pfade und Anwendungen abgebrochen (Abbruch gesetzt).Es erscheint jetzt neu die Untermenübezeichnung „Abbruch aus".Durch erneuten Mausklick kann jetzt der Status Abbruch wieder zurückgesetzt werden (Anzeige dann wieder:"Abbruch an").Der Abbruch wird in gleicher Weise zurückgesetzt,wenn irgend ein beliebiges Menü der Menüleiste mit der Maus angewählt wird.Der Demomodus wird durch das Setzen des Abbruchs gelöscht,der Pausenzeit wird Null zugewiesen.

Demo an (aus):

Nach Anwahl dieses Menüs mit der linken Maustaste wird der Demomodus eingeschaltet.In diesem Modus kann die Funktions- und Arbeitsweise der Algorithmen der Menüs Pfade und Anwendungen als Demonstration schrittweise genauer verfolgt werden,und die Algorithmen laufen außerdem noch bei Wahl einer entsprechenden Pausenzeit verlangsamt ab.Die Verlangsamung kann durch Vorgabe einer Demowartezeit (Pausenzeit) (von 0 s bis 32 s wählbar) eingestellt werden.Zwischen jedem Algorithmusschritt pausiert der Algorithmus dann um die Größe dieser Wartezeit.

Nach Anwahl des Menüs erscheint ein Eingabefenster,in das die Wartezeit eingegeben werden kann (Vorgabewert ist 0).Die Anzeige des Menüs wechselt zwischen „Demo an" und „Demo aus".Im letzteren Fall kann durch erneute Anwahl des Menüs der Demomodus wieder ausgeschaltet werden (ist also momentan eingeschaltet).

Pausenzeit:

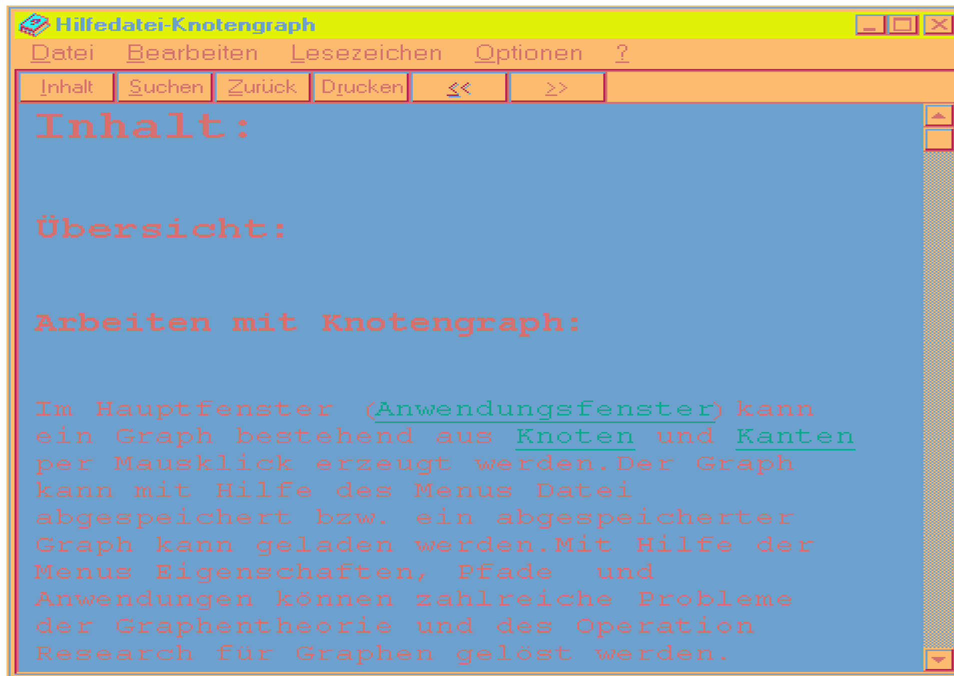
Durch Anwahl dieses Menüs mit der linken Maustaste kann,während eine Anwendung im Demo-Modus läuft,die Demowartezeit verändert werden.Hierzu erscheint ein Eingabefenster,in das die neue Pausenzeit einzugeben ist (Wert von 0 s bis 32 s).Vorgabewert ist dabei 0 s.

Menü Hilfe:

Das Menü Hilfe besteht aus den Untermenüs Hilfe und Info.

Hilfe:

Beim Klick mit der linken Maustaste auf das Menü Hilfe wird die Hilfedatei des Programms Knotengraph geladen.Ihr Aufbau entspricht den normalen Windowskonventionen.Z.B. gibt es als Übersicht ein Kapitel Inhalt.Es finden sich in der Hilfe Kurzbeschreibungen zu allen Algorithmen der Untermenüs.Diese Beschreibungen können auch kontextbezogen durch Drücken der F1-Taste aufgerufen werden,wenn sich der Mauszeiger gerade auf einem der Untermenüeinträgen befindet.

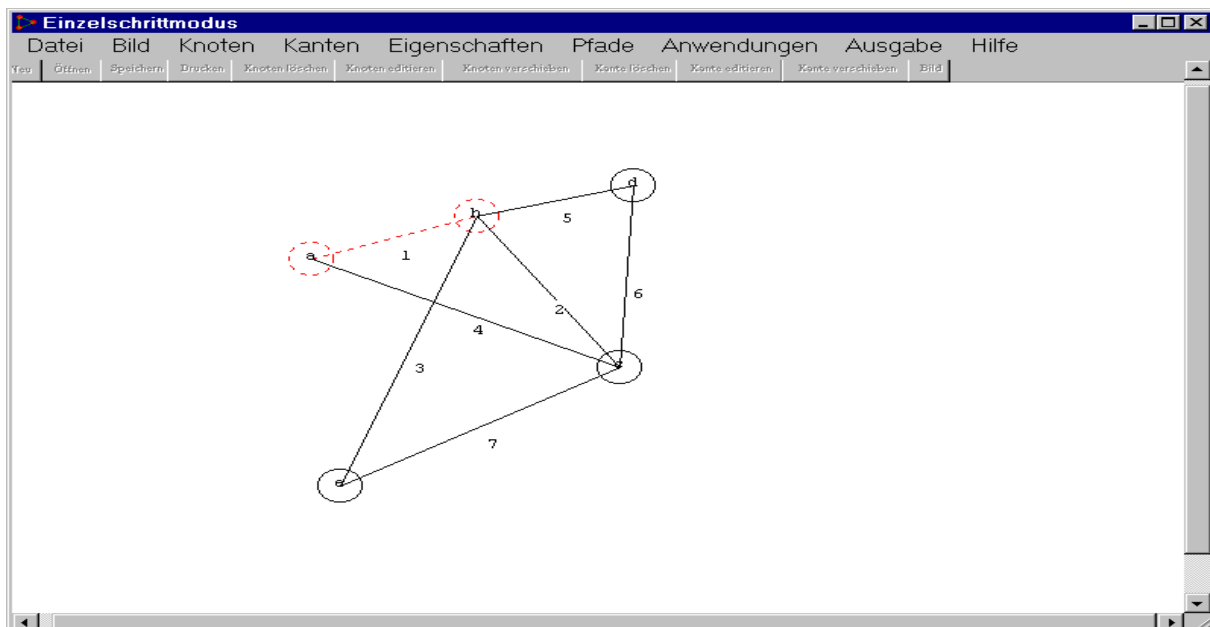


Info:

Nach Anwahl dieses Menüs mit der linken Maustaste erscheint ein Anzeigefenster mit Informationen zum Programmauthor von Knotengraph.

Einzelschrittmodus:(ohne Menü)

Durch Mausklick mit der linken Maustaste auf die rechte Hälfte der Panelfläche unter gleichzeitiger Betätigung der Shift-Taste wird der Einzelschrittmodus gestartet.Beendet wird er durch Mausklick mit der linken Maustaste unter gleichzeitiger Betätigung der Shift-und der Alt-Taste auf die gleiche Fläche.Durch Mausklick mit der linken Maustaste auf die rechte Hälfte der Panel-Fläche unter gleichzeitiger Betätigung der Alt-Taste wird jeweils ein Demo-Schritt ausgeführt.Mit dem Einzelschrittmodus wird auch immer der Demmomodus aktiviert und bei seinem Ende wieder abgeschaltet.



F III Eigenschaften der verwendeten Programmiersprache Delphi

Als Programmiersprache für das Programm Knotengraph wurde Delphi in den Versionen 1 bis 6 verwendet.Im folgenden sollen kurz die Eigenschaften dieser Programmiersprache erläutert werden,soweit sie für die dieser Arbeit zu Grunde liegenden Programme von Bedeutung sind.

Delphi als didaktische Erweiterung von Turbo-Pascal

Delphi ist eine Programmiersprache und Programmierungsumgebung für Windows 3.1 (Version 1) oder Windows 95/98 (alle Versionen,ab Version 2 nur noch Windows 95/98),die die Standardbefehle von Pascal stark erweitert und eine an das Betriebssystem Windows anpaßte Programmentwicklung ermöglicht.

Delphi ermöglicht objektorientierte,visuelle und ereignisgesteuerte Programmierung.Die beiden letztgenannten Eigenschaften wurde als erstes in der Sprache Visual Basic implementiert.In letzter Zeit gibt es immer mehr Programmiersprachen mit dieser Eigenschaft,z.B:Visual C++ und J++ (Java),wobei diese Programmiersprachen ebenfalls wie Delphi objektorientiert ausgerichtet sind.

Pascal wurde 1972 von Wirth als Modell für eine Programmiersprache entwickelt.Ihre besonderen (imperativen) Eigenschaften sind der modulare Aufbau mittels Proceduren und Functionen und die Möglichkeit sehr übersichtlich mittels geeigneter Kontrollstrukturen wie z.B. den Schleifen sowie der verschiedenen Strukturmöglichkeiten für Daten zu programmieren.

Aus diesem Grund hat sich die Sprache im Informatikunterricht an vielen Schulen als Standard zur Einführung in das Programmieren etabliert.Und auch die Lehrerfortbildung NRW Gymnasiale Oberstufe favorisiert(e) und verwendet(e) die Programmiersprache Pascal allerdings inzwischen mehr in der Form ihres Nachfolgers Delphi als Leitsprache für den Unterricht.Die neuen Richtlinien Informatik NRW,die im Jahre 1999 in Kraft treten,sehen u.a. in den Jahrgangsstufen 11 bis 13 eine Unterrichtsreihe visuelle,objektorientierte Programmierung vor,die auf die Sprache Delphi zugeschnitten ist.Der Vorteil,Delphi statt einer anderen Sprache dabei zu benutzen,liegt darin,dass sich Delphi als echte Erweiterung anbietet und eine bereits verwendete Syntax von Turbo Pascal unverändert beibehalten werden kann.Beispielsweise ist es möglich,die elementare Grundlagen imperativer Programmierung wie Kontrollstrukturen und einfache Datentypen sowie das Procedurenkonzept mit Hilfe der einfacheren Turbo-Pascal-Syntax zu vermitteln,um dann als Fortsetzung Programme mit graphisch gestalteten Oberflächen objektorientiert und visuell mittels der wesentlich weiterführenden,komplexeren Delphi-Quellcodemöglichkeiten zu erstellen.

Spätestens seit dem Erscheinen von Windows 95 und dem Nachfolger Windows 98 bzw. Windows NT/2000 wurde das DOS-Betriebssystem,unter dem die meisten der bisherigen Turbo-Pascal-Versionen liefen,durch Windows ersetzt.

Da Windows eine graphische Benutzeroberfläche beinhaltet,muss für eine adäquate Programmerstellung unter Windows eine großer Teil der zu erledigenden Arbeit in das Erstellen einer graphischen Oberfläche investiert werden.

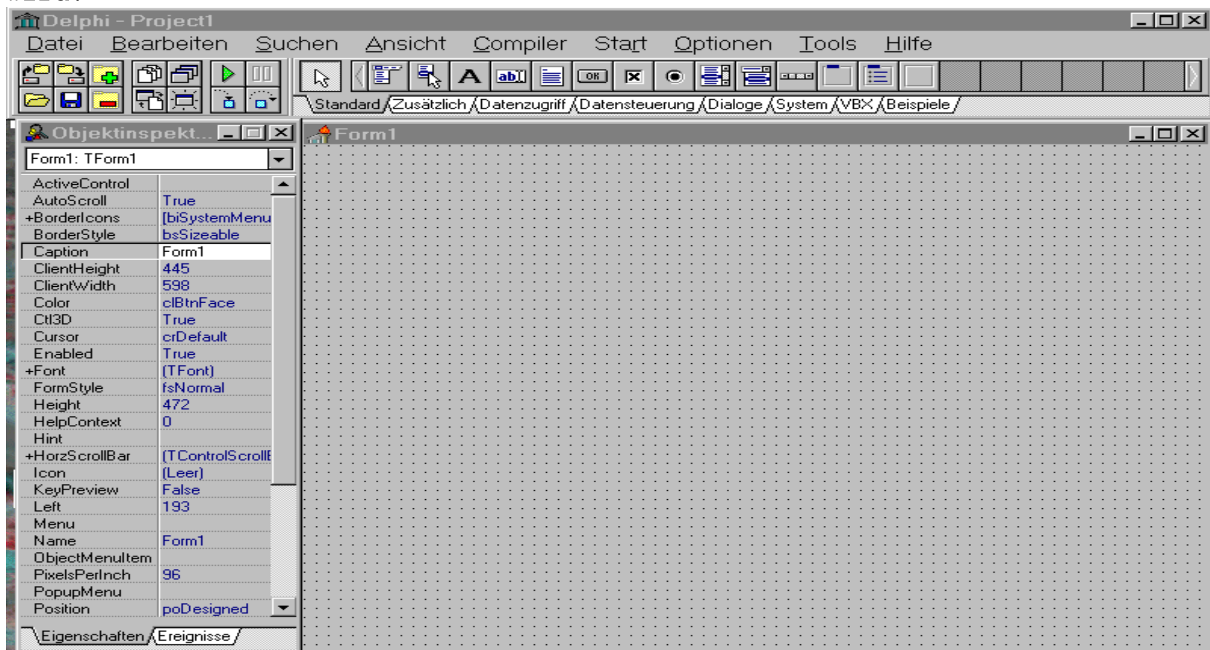
Dies geschieht bei Programmiersprachen ohne visuelle und objektorientierte Programmierungsmöglichkeiten wie z.B. C oder auch der (früheren) Version Turbo-Pasal für Windows mittels des Aufrufs von Windows-Betriebssystemfunktionen,wobei ein Programm,das nur das Wort Hallo in einem Fenster ausgibt,schon mindestens 100 Programmzeilen Code erfordert.

Wegen des Wechsels vom DOS-Betriebssystems auf die neuen graphischen Betriebssysteme wird eine Programmiersprache gesucht,bei der zur Programmierung der graphischen Oberfläche nicht soviel Aufwand nötig ist.Ansonsten erstreckt sich ein Informatikunterricht nur in der Erstelllung der üblichen Komponenten wie Fenster,Buttons,Rollbalken usw.,und die bisher im Informatikunterricht vermittelten Unterrichsinhalte treten in den Hintergrund.

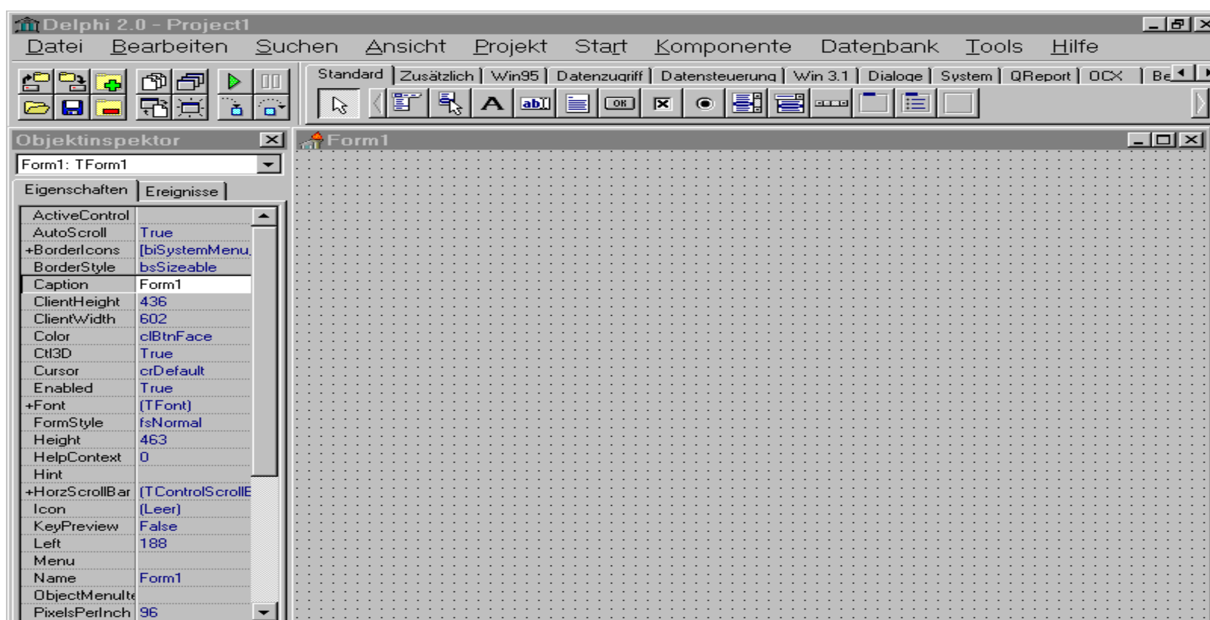
Die Programmiersprache Delphi vermag diese Lücke zu schließen.Durch das visuelle Erstellen von graphischen Komponenten durch Anklicken und Ziehen mit der Maus und das automatische Erzeugen von dazugehörigem Quellcode wird in die Erstellung der graphischen Oberfläche nur wenig Arbeit investiert,und der Programmierer (Schüler) kann sich hauptsächlich wieder der Programmierung der eigentlichen algorithmisch interessanten Aufgabenstellung widmen,die durch die visuell erstellte Oberfläche lediglich bedient wird.

Die Entwicklungsumgebung von Delphi

Die visuelle Programmierung wird ermöglicht durch die Entwicklungsumgebung von Delphi,über deren Benutzeroberfläche zunächst ein Überblick gegeben wird:

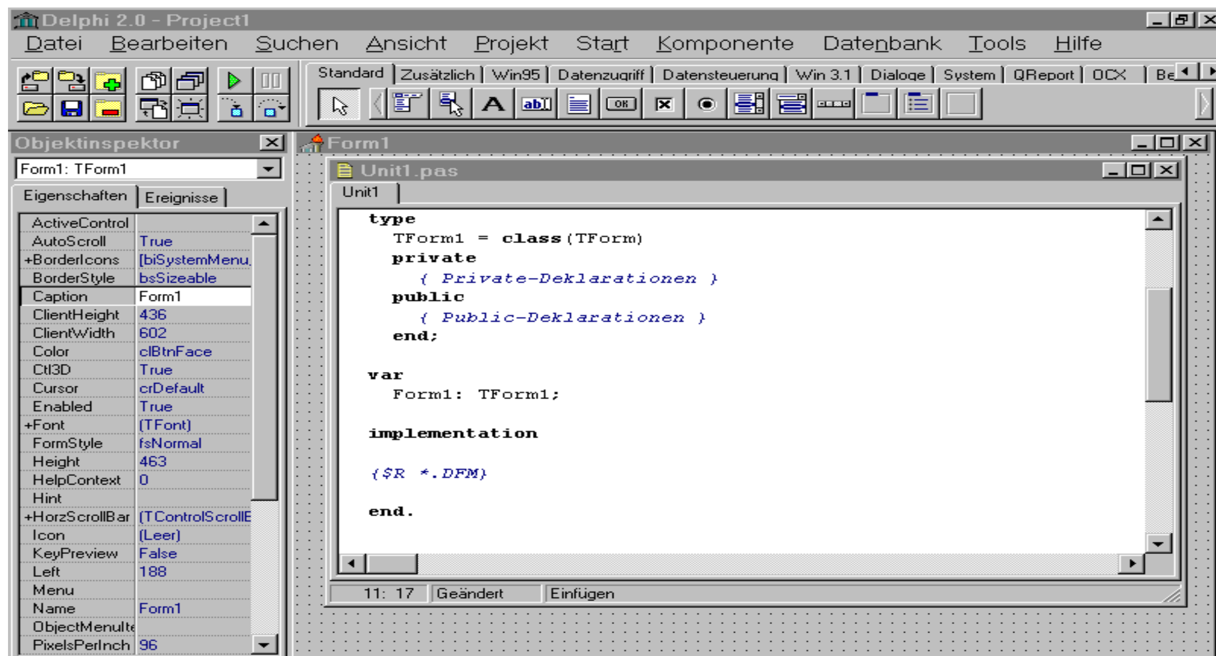


Benutzeroberfläche von Delphi 1



Benutzeroberfläche von Delphi 2

Beim Starten des Programms erscheint die integrierte Benutzeroberfläche.Diese gliedert sich in das Hauptfenster,den Objektinspektor,das Programmfenster (Form) und den Quelltext-Editor.



Quelltext-Editor

Das Hauptfenster besteht einerseits aus einer Reihe von Menüs in der oberste Reihe und Icons im linken Teil,mit denen Programmdateien geladen und gespeichert,Programme compiliert und gestartet sowie diverse Optionen eingestellt werden können,wie dieses schon aus früheren Turbo-Pascal-Versionen bekannt ist.

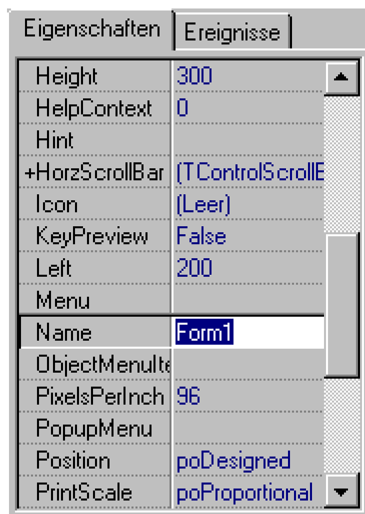
Form Knotenform mit Komponenten

Andererseits befindet sich im rechten Teil eine Symbolleiste mit diversen Icons für die verschiedenen Komponenten aus der symbolischen Komponentenbibliothek,die das Programmieren und das automatische Erzeugen von Quelltext für die für Windows typischen Komponenten wie Fenster,Rollbalken,Buttons, Menüs usw. ermöglichen.

Das Programmfenster (Form) stellt das Start-und Hauptfenster der zu erstellenden Windowsanwendung dar.Zu dem Fenster gehört eine Unit,deren Quelltext im Quelltexteditor mittels Umschaltung (durch die F12-Taste) sichtbar gemacht und editiert werden kann.Bei Bedarf lassen sich auch mehrere Hauptfenster (Forms) erzeugen,die nacheinander oder gleichzeitig in einem Programm aufgerufen,d.h. sichtbar gemacht werden können.In die Fenster eingebunden werden die interaktiven Komponenten der Komponentenbibliothek dadurch,dass diese Komponenten (visuell) mit der Maus auf das Programmfenster (die Form) gezogen werden,wodurch der dazugehörende Quelltext in die Unit der Form automatisch eingefügt wird.

Alle Komponenten und die Programmfenster (Form) sind in Delphi Objekte und sind deshalb so leicht zu benutzen,weil sie schon eine Vielzahl von geerbten Datenfeldern (Attributen) und Methoden besitzen.Auf die Datenfelder wird mittels ihrer Propertys zugegriffen.Dies geschieht im Entwurfsmodus durch den Objektinspektor.

Der Objektinspektor besteht wiederum aus zwei Teilen:der Eigenschafts-und der Ereignisliste.



Objektinspektor

Mittels der Eigenschaftsliste lassen sich die Anfangs-Eigenschaften (dies sind die eben erwähnten Propertys) der visuellen Komponenten und der Form (z.B: Breite,Höhe,Farbe, Name, Titel,Sichtbarkeit usw.)zur Entwurfszeit einstellen.Zur Laufzeit sind den entsprechenden Propertys dann vom Programm Werte zuzuweisen.

Die Einstellungen in der Ereignisliste geben an,auf welche Windows-typischen Ereignisse eine Komponente (z.B. Anklicken mit der Maus,Taste drücken usw.) in welcher Weise reagieren soll.

Dazu wird eine Rahmenmethode nach Mausklick auf die gewünschte Art der Ereignismethode im Objektinspektor und im Quelltexteditorfenster erzeugt,die dann vom Programmierer mit dem entsprechenden Programmcode gefüllt werden kann.Der Programmcode bestimmt, was passieren soll,wenn das entsprechende Ereignis eingetreten ist.Diese Eigenschaft von Delphi heißt ereignisgesteuerte Programmsteuerung.Die zugehörigen Methoden heißen Ereignismethoden.

Für jedes entworfene Fenster (Form) wird eine eigene Quelltextdatei in Form einer Unit (Extension PAS)angelegt.Bei der Gestaltung eines Programmfensters (Form) wird außerdem eine Datei (mit der Extension DFM) angelegt,in der die graphische Gestaltung der Komponenten und deren Eigenschaften gespeichert sind.

Es besteht die Möglichkeit,auch Units,denen keine Formen entsprechen, anzulegen und auf deren Proceduren und Functionen sowie deren Objekte und Objektmethoden (Uses-Anweisung) zurückzugreifen.

Alle Dateien zusammen werden am besten in einem eigenen Verzeichnis gespeichert und bilden zusammen ein Projekt.Eine Projektdatei (mit der Extension DPR),die von der Entwicklungsumgebung beim Speichern selber erzeugt wird,enthält alle Informationen über die zu dem Projekt gehörenden Dateien.Durch sie wird das Projekt in die Entwicklungsumgebung geladen.

Das in ihr enthaltende Hauptprogramm besteht im wesentlichen aus einem Constructor-Aufruf zur Erzeugung des Startfensters und dem Application.Run-Befehl:

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

(Bei mehreren Formen sind die beiden letzten Anweisungen entsprechend mehrfach vorhanden.)

Statt dieser Form der Projektdatei lassen sich in der Delphi-Version 1 (und nur in dieser Version) mittels Einbindung der Unit WINCRT ähnlich wie in Turbo Pascal für Windows Projektdateien entwerfen,die im Aufbau einem konventionellem Pascal-Hauptprogramm entsprechen und unter Windows in einem einzigem (Consolenanwendungs-)Fenster ablaufen.Dann ist jedoch keine visuelle Erstellung einer Benutzeroberfläche möglich.Es lassen sich so Zeilen- und Textorientierte Programme entwerfen.

```
program CrtApp;

uses WinCrt;

begin
  Writeln('Delphi');
end.
```

Konzept der objektorientierten Programmierung

Besonders bemerkenswert ist,dass die visuelle Programmerzeugung von Delphi durchgehend mit objektorientierten Strukturen arbeitet.Die Form selber ist schon ein Objekt und die in ihr enthaltenden Komponenten werden als Objektattribute dieses Objekts durch visuelles Ziehen auf dieses Form automatisch implementiert.Die Ereignisse sind Methoden der Form bzw. der Komponenten.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin

end;

procedure TForm1.Button1Click(Sender: TObject);
begin

end;

end.
```

Automatische objektorientierte Quellcodeerzeugung:Unit Form1 mit Komponente Button1 und den Methoden Form1.create sowie Button1click

Unter einem Objekt versteht man die Zusammenfassung (Kapselung) von Daten (Attributen) und Methoden zu einer Einheit.

Aus einem Objekt läßt sich durch Vererben ein neues Objekt gewinnen,das alle Eigenschaften des Vorgängerobjekts hat und zusätzliche Daten (Attribute) und Methoden erhält.Bei Bedarf lassen sich Methoden des Vorgängers durch Überschreiben neu definieren (statische Vererbung).

Andererseits läßt sich die gewünschte Funktionsweise der Methode eines Objekts auch erst zur Laufzeit des Programms festlegen (Polymorphie).

Delphi führt im Gegensatz zu der Vorgängerversion Turbo-Pascal für Windows ein neues Objektmodell ein.

Das Wort object wird durch class ersetzt.

Ein Objekt wird definiert durch:

```
type TNeuesObject=class
       Datenfeld1_:Integer;
       .
       .
       .
       Methode1;
       .
       .
       .
     end;
```

Ein Objekt,das von einem Vorgängerobjekt erbt,wird definiert:

```
type TVererbtesObject=class(TNeuesObject)
        .
        .
        .
      end;
```

Eine Instanz eines Objekts wird definiert durch:

```
Var Instanz:TNeuesObject;
```

Auf die Datenfelder wird zugegriffen durch:

```
Instanz.Datenfeld1_:=3;
```

(wobei ein direkter Zugriff durch einen Anwender zwar syntaktisch möglich ist,aber besser vermieden werden und nur durch geeignete Methoden erfolgen sollte,z.B. mittels des Property-Konzepts)

Eine Methode wird aufgerufen durch:

```
Instanz.Methode1;
```

Alle Objekte leiten sich von einem gemeinsamen Vorfahren TObject durch Vererbung ab.Unter anderem ist durch das Objekt-Modell von Delphi auch die rekursive Deklaration eines Objekts möglich.

Das Objekt-Konzept ist auf die Anforderungen der graphischen Oberfläche zugeschnitten:Die (visuell zu erzeugenden) Komponentenelemente von Delphi sind objektorientiert ausgerichtet und bilden die Blätter eines weitverzweigten Objektbaumhierachie.

Durch das Vererben ihrer Eigenschaften auf von ihnen abgeleitete Objekte ist das Erstellen eigener graphischer Komponenten so leicht möglich,wobei nur der Programmcode geschrieben werden muss,der das neue Objekt von dem Vorgänger unterscheidet.

Delphi erlaubt die mehrfache Vererbung (jedoch nicht die Mehrfachvererbung) von Objekten,was bedeutet,dass eine vererbte Klasse wiederum Nachfolgerklassen bilden kann.Davon wird im Programm Knotengraph wesentlich Gebrauch gemacht.Beispielsweise läßt sich aus dem in Delphi vordefinierten Datentyp TList (als Liste von Objekten, genauer: Objektzeigerliste) zunächst der Datentyp TListe ableiten,und aus TListe lassen sich dann die Datentypen TKantenliste und TKante sowie TKnotenliste und TKnoten erstellen:

```
TListe=class(TList)
    .
    .
    .
end;

TKantenliste = class(TListe)
    .
    .
    .
end;

TKante =  class(TKantenliste)
    .
    .
    ,
end;

TKnotenliste = class(TListe)
    .
    .
    .
end;

TKnoten = class(TKnotenliste)
    .
    .
    .
end;
```

Der Datentyp TGraph leitet sich hingegen von dem obersten Objekttyp TObject in Delphi ab und enthält (hat) zwei Datenfelder vom Typ der zuvor definierten Objekttypen:

```
TGraph = class(TObject)
  Knotenliste_:TKnotenliste;
  Kantenliste_:TKantenliste;
  .
  .
  .
end;
```

Objekte lassen sich also ineinanderschachteln.(„hat-Beziehung")

Durch Vererbung leiten sich von den bisher definierten Typen die Objekte TInhaltsknoten,TInhaltskante und TInhaltsgraph ab:

```
TInhaltsknoten = class(TKnoten)
   .
   .
   .
end;

TInhaltskante = class(TKante)
   .
   .
   .
end;

TInhaltsgraph = class(TGraph)
   .
   .
   .
end;
```

Diese Objekttypen bilden wiederum das Ausgangsobjekt zur Ableitung der bei den (Graphen-)Anwendungen benutzten Knoten,Kanten und Graphen:

Z.B. sind die bei der Berechnung von absorbierenden Markovketten benutzten Objekttypen:

```
TMarkovknoten = class(TInhaltsknoten)
   .
   .
   .
end;

TMarkovkante = class(TInhaltskante)
   .
   .
   .
end;

TMarkovgraph = class(TInhaltsgraph)
   .
   .
   .
end;
```

Innerhalb einer Objektfamilie ist Typkonversion (Typecasting) möglich:

Z.B. kann eine mittels

```
var Kno:TKnoten
```

definierte Instanz in den Datentyp TInhaltsknoten durch die Syntax

```
Knoneu:=TInhaltsknoten(Kno);
```

umgewandelt werden.

Methoden und Propertys

Instanzen von Objekte sind in Delphi automatisch Zeigertypen.Jedes Objekt kann Datenfelder (Attribute) und Methoden (deren Syntax einerseits an die Deklaration von Variablen und andererseits an die von Proceduren oder Functionen in Pascal erinnert) enthalten,auf die,falls sie als private (und nicht als public) deklariert sind,nur durch Instanzen des Objekts in der Deklarationsunit zugegriffen werden kann.Die Syntax für Datenfelder und Methoden ist,dass im Objekt zuerst die Datenfelder und danach die Methoden festgelegt werden:

```
TKnoten = class(TKnotenliste)
   .
   .
   .
    Besucht_:Boolean;
   .
   .
   .
    function Istbesucht:Boolean;
    procedure Setzebesucht(B:Boolean);
```

```
    .
    .
end;
```

Das Objekt TKnoten enthält z.B. das Feld Besucht_ vom Typ Boolean.Mittels Var Kno:TKnoten wird eine Instanz vom Typ TKnoten deklariert und mittels Kno.Besucht_:=true kann der Wert des Feldes gesetzt werden.

Guter objektorientierter Programmierstil ist es dagegegen,wie gesagt,nur mittels Methoden auf die Datenfelder zuzugreifen.Dazu werden die Methoden Setzebesucht und Istbesucht definiert:

```
function TKnoten.Istbesucht:boolean;
begin
  Istbesucht:=Besucht_;
end;

procedure TKnoten.Setzebesucht(B:boolean);
begin
  Besucht_:=B;
end;
```

Durch Verwendung dieser Methoden,die durch Kno.Setzebesucht(true) bzw. Abfrage:=Kno.Istbesucht aufgerufen werden (mit var Abfrage:Boolean),kann dann indirekt mit dem Datenfeld gearbeitet werden.

Da jedoch die direkte Wertzuweisung eine sehr anschauliche und prägnante Art der Veränderung eines Datenfeldes ist,lassen in Delphi sogenannte Propertys und zugehörige Propertymethoden definieren,die die Vorzüge beider Zugriffsmöglichkeiten auf ein Datenfeld (Attribut) kombinieren:

Dies geschieht durch folgende Deklaration:

property Besucht:Boolean read Istbesucht write Setzebesucht;

Dadurch sind jetzt mit var Kno:TKnoten folgende Zuweisungen möglich:

```
Kno.Besucht:=true;
Abfrage:=Kno.Besucht;  mit var Abfrage:Boolean;
```

Diese beiden Zeilen sind jedoch keine wirklichen Wertzuweisungen,sondern bei der Zeile Kno.Besucht:=true wird die Methode SetzeBesucht aufgerufen,um auf Kno.Besucht_ den Wert true abzuspeichern und bei Abfrage:=Kno.Besucht wird die Funktionsmethode IstBesucht aufgerufen,um den in Kno.Besucht_ gespeicherten Wert in der Variablen Abfrage zu speichern.

Die Read und Write-Methode muß bei der Deklaration einer Property angegeben werden:

```
property Besucht:Boolean read Istbesucht write Setzebesucht;
```

Die Read- und Write-Methoden brauchen dabei nicht nur wie hier aus einer Wertzuweisung bestehen,sondern können beliebige Anweisungen enthalten.Die Read-Methode muß immer eine Function ohne Parameter vom Datentyp der zu definierenden Property und die Write-Methode muß eine Procedure mit einem Werteparameter vom Datentyp der zu definierenden Property sein.Eine Property kann auch nur **eine** Read-oder **eine** Write-Methode enthalten.(Dann kann entweder nur gelesen oder aber nur geschrieben werden.)

Durch das Definieren von Propertys wird also mittels einer scheinbaren Wertzuweisung auf die Datenfelder mittels Methoden zugegriffen.

Da auf das Datenfeld von einem Benutzer nicht direkt zugegriffen werden sollte,wird der Deklarationsteil der Datenfelder einschließlich ihrer Read- und Write-Methoden als private definiert,während die Deklaration der Property im Public-Teil erfolgt.

Als Beispiel hierzu sei die Property X vom Objekt TInhaltsKnoten erläutert.Dabei ist X die x-Koordinate des Mittelpunkts des Knotens auf dem Bildschirm.

```
TInhaltsknoten = class(TKnoten)
private
  .
  .
  X_:Integer;
  .
  .
  .
  procedure Schreibex(x:Integer);
  function Lesex:Integer;
  .
  .
public

property x:Integer read Lesex write Schreibex;
    .
    .
    .
 end;
```

Zur Aufnahme der x-Koordinate dient das Datenfeld X_ vom Typ Integer.Um auf dieses Datenfeld mittels Methoden zugreifen zu können,werden die Methoden Schreibex und Lesex im private-Teil definiert:

```
procedure TInhaltsknoten.Schreibex(x:Integer);
begin
  X_:=X;
end;

function TInhaltsknoten.Lesex:Integer;
begin
  Lesex:=X_;
end;
```

Zusätzlich wird jetzt X als Property im public-Teil definiert durch:

```
property X:Integer read Lesex write Schreibex;
```

Dadurch sind jetzt mit var Kno:TInhaltsknoten folgende Zuweisungen möglich:

```
Kno.X:=50;
XKoordinate:=Kno.X;  mit var XKoordinate:Integer
```

Eine besondere Property wird auch für den Zugriff auf die Elemente der Liste vom Datentyp TListe definiert:

```
TListe=class(TList)
    .
    .
    .
    function Element(Index:Integer):TElement;
    property Items[Index:Integer]:TElement read Element;
    .
    .
    .
end;
```

TElement wird mittels der rekursiven Definition deklariert zu:

```
TElement=class(TListe)
  .
  .
  .
end;
```

Die Funktion Element ist die read-Methode (eine write-Methode wird nicht definiert) der (Array-)Property Items,die indizierten Zugriff ermöglicht. Mittels Items[Index] wird auf das Element der Nummer Index der durch die Read-Methode Element definierten Property zugegriffen:

```
function Tliste.Element(Index:Integer):TElement;
begin
  if (Index<=Self.ZahlderElemente-1)and (Index>=0)
  then
```

```
    result:=TElement(Tlist(self).Items[Index])
  else
    Showmessage('Fehler! Listenindex außerhalb des zulässigen Bereichs');
end;
```

Durch die Typkonversion TElement(TList(self).Items[Index]) (Typecasting) kann dann auf die Elemente der Liste durch Element(Index) (mit Index innerhalb des oben angegebenen Bereichs) zugegriffen werden.
Dasselbe Verfahren wird bei TKante und TKantenliste,TKnoten und TKnotenliste sowie TPfad und TPfadliste angewendet,so dass z.B. auf die Knoten einer Knotenliste einfach mittels Knotenliste.Knoten(Index) zugegriffen werden kann.

Constructoren und Destructoren

Jede Instanz eines Objekts muß zunächst durch den Aufruf seiner Constructor-Methode erzeugt werden.Entweder wird der Constructor von dem Vorgängerobjekt geerbt,oder aber er muß,insbesondere dann,wenn neue Datenfelder (Attribute) zu initialisieren sind,neu definiert werden.Sinnvoll dabei ist, falls ein Vorgängerobjekt existiert,zunächst den Constructor dieses Vorgängerobjekts aufzurufen,um dessen Eigenschaften ausnutzen zu können:

Z.B. ruft der Constructor von TInhaltsknoten zunächst den Constructor von TKnoten durch die Anweisung inherited Create (Vorgängermethode) auf,um danach noch eigene neue Datenfelder zu initiieren:

```
constructor TInhaltsknoten.Create;
begin
  inherited Create;
  Inhalt_:=' ';
  X_:=0;
  Y_:=0;
  Radius_:=15;
  Farbe_:=clblack;
  Stil_:=pssolid;
  Typ_:=' ';
  Wertlisteschreiben;
end;
```

Eine (Zeiger-)Instanz von TInhaltsknoten mit Var Kno:TInhaltsknoten wird dann durch den Constructor-Aufruf Kno:=TInhaltsknoten.Create erzeugt.

(Warnung:Bei Arbeiten mit Objektinstanzen,die nicht durch einen Constructor-Aufruf initiiert wurden,erfolgt meistens ein Absturz der Entwicklungsumgebung von Delphi bzw. des erzeugten Programms und möglicherweise auch des Betriebssystems Windows.)

Das Gegenteil der Wirkung eines Constructors ist das Entfernen der Instanz eines Objekts aus dem Speicher (Destructor).Dies wird bei den von TObject abgeleiteten Klasseninstanzen bewirkt durch die (von TObject vererbte) Methode Free.

(Eine andere Destructor-Möglichkeit ist das Verwenden der Methode Destroy statt Free.Free hat jedoch den Vorteil,dass bei mittels nil belegten Objektzeiger,die demnach auf keine Objektinstanz mehr zeigen,kein Absturz der Entwicklungsumgebung und des erzeugten Programms beim Aufruf von Free erfolgt.)

Beispielsweise lässt sich die oben erzeugt Instanz Kno durch den Aufruf Kno.Free wieder aus dem Speicher löschen.Allerdings werden dadurch nicht die Objektinstanzen,die wiederum als Attribute (Datenfelder) in dem Objekt enthalten sind (Kanten der aus-und eingehenden Kantenlisten),gelöscht.Es ist deshalb nötig die komplexere Methode Freeall zum kompletten Entfernen der Datenstruktur aus dem Speicher zu schreiben:

```
procedure TKnoten.Freeall;
var Index:Integer;
    Ka:TKante;
begin
  if not AusgehendeKantenliste.Leer then
    for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
```

```
    begin
      Ka:=AusgehendeKantenliste.Kante(Index);
      if  Ka.Gerichtet and (Ka.Anfangsknoten<>Ka.Endknoten) then
      begin
        Ka.Free;
        Ka:=nil;
      end;
    end;
    if not EingehendeKantenliste.Leer then
      for Index:=0 to EingehendeKantenliste.Anzahl-1 do
      begin
        Ka:=EingehendeKantenliste.Kante(Index);
        Ka.Free;
        Ka.=nil;
      end;
  Free;
end;
```

Virtuelle Constructoren

Die Definition eines neuen Graphentypen,der vom Typ TInhaltsgraph durch Vererbung abstammt,erfordert meistens auch die Deklaration neuer Knoten-und Kantentypen (z.B. mit neuen Datenfeldern), die von den Objekten TInhaltsknoten und TInhaltskante abgeleitet werden.Trotzdem die Methoden von TInhaltsgraph nur auf den Datentypen TInhaltsknoten und TInhaltskante (und natürlich auf TInhaltsgraph selbst) operieren (weil nur diese zur Entwurfszeit von TInhaltsgraph bekannt sind),möchte man diese an den Nachfolger von TInhaltsgraph vererbten Methoden einsetzen,um durch den von TInhaltsgraph abgeleiteten neuen Graphtyp auch die neuen Knoten-und Kantentypen mittels dieser Methoden verwalten zu können.Diese Problem kann nur durch die Verwendung virtueller Methoden,insbesondere virtueller Constructoren gelöst werden.

Delphi erlaubt die Verwendung von virtuellen Constructoren und die Erzeugung von Klassenrefrenzen:

Eine Klassenreferenz wird als Typ festgelegt durch

```
type TClassref=class of TBeliebigeClass;
```

Dazu enthält die Unit Systems von Delphi den vordefinierten Datentyp type TClass=class of TObject.

Die Unit UInhGrph enthält entsprechend der eben genannten Syntax die Klassenreferenz-Deklarationen:

```
TInhaltsknotenclass = class of TInhaltsknoten;
TInhaltskanteclass = class of TInhaltskante;
TInhaltsgraphclass =class of TInhaltsgraph;
```

Objekte vom Typ TInhaltsgraph definieren dann zwei Datenfelder Inhaltsknotenclass_ und Inhaltskantclass_ von diesem Typ:

```
TInhaltsgraph      =     class(TGraph)
    .
    .
    .
    Inhaltsknotenclass_:TInhaltsknotenclass;
    Inhaltskanteclass_:TInhaltskanteclass;
    .
    .
    .
end;
```

Mittels der Methoden

```
function WelcheKnotenclass:TInhaltsknotenclass;
procedure SetzeKnotenclass(Inhaltsclass:TInhaltsknotenclass);
function WelcheKantenclass:TInhaltskanteclass;
procedure SetzeKantenclass(Inhaltsclass:TInhaltskanteclass);
```

kann auf diese Datenfelder zugegriffen werden.Dazu werden die Propertys definiert:

```
property Inhaltsknotenclass:TInhaltsknotenclass read WelcheKnotenclass
  write SetzeKnotenclass;
property Inhaltskanteclass:TInhaltskanteclass read WelcheKantenclass
   write SetzeKantenclass;
```

Im Constructor eines von TInhaltsgraph abgeleiteten Graphtypen ist in diesen beiden Feldern mittels der Propertys der Objekttyp der verwendeten Knoten und Kanten zu speichern.

Z.B. lautet der Constructor von TNetzgraph:

```
constructor TNetzgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TNetzknoten;
  InhaltsKantenclass:=TNetzkante;
end;
```

Die Constructoren von TInhaltsknoten und TInhaltskante sind als virtual deklariert (und die Constructoren der von ihnen abgeleiteten Objekttypen haben das Attribut override):

```
TKnoten = class(TKnotenliste)
   .
   .
   .
    constructor Create;virtual;
   .
   .
   .
end;

TKante = class(TKantenliste)
   .
   .
   .
    constructor Create;virtual;
   .
   .
   .
end;
```

Bei der Erzeugung eines Knotens in der Methode Knotenzeichnen wird nun z.B. statt des Erzeugens eines Inhaltsknotens mittels des Aufrufs TInhaltsknoten.create der virtuelle Constructoraufruf Inhaltsknotenclass.create verwendet.

```
function TInhaltsgraph.Knotenzeichnen(X,Y:Integer):Boolean;
.
.
begin
  .
  K1:=Inhaltsknotenclass.Create;
  .
  FuegeKnotenein(K1);
  .
end;
```

Dadurch wird der Constructor des dem neuen Graphen zugeordneten Knotentyps und nicht der von TInhaltsknoten aufgerufen und in den Graph eingefügt. Dasselbe gilt analog für die Erzeugung von Kanten.

Auf diese Weise lassen sich die Methoden von TInhaltsgraph für von TInhaltsgraph durch Vererbung abgeleitete neue Graphtypen mit von TInhaltsknoten und TInhaltskante abgeleiteten Knoten-und Kantentypen verwenden.

Speichern und Laden der Daten von Nachfolgerobjekten

Schwierigkeiten treten nur noch bei der Speicherung und dem Laden von Graphen auf,weil hier eventuell neu definierte Datenfelder vom Graphtypen und von den neuen Knoten- und Kantentypen berücksichtigt werden müssen.Dieses Problem kann nicht durch das Prinzip des virtuellen Constructors angegangen werden,sondern wird durch das Definieren einer Wertliste gelöst.

(Für nähere Einzelheiten dazu siehe auch die Erläuterungen im Kapitel F I des Anhangs Beschreibung der Datenstruktur)

Die Lösung sei hier,soweit sie besondere Eigenschaften von Delphi betrifft, kurz an Hand der Wertliste von TKnoten erläutert:

(Anschließend wird besprochen,wie sich das Problem mit Hilfe der Benutzung von Streams lösen ließe,was jedoch für das vorliegende Problem ungünstiger ist.)

```
TKnoten = class(TKnotenliste)
  private
    .
    .
    Wertposition_:Integer;
    Wertliste_:TStringlist;
    .
    .
    procedure SetzeWertposition(p:Integer);
    function WelcheWertposition:Integer;
    function Wertlisteschreiben:TStringlist;virtual;abstract;
    procedure Wertlistelesen;virtual;abstract;
    .
    .
public
  constructor Create;virtual;
  .
  .
  property Position:Integer read WelcheWertposition write SetzeWertposition;
  property Wertliste:TStringlist read WelcheWertliste write SetzeWertliste;
  property Wert:string read Wertlesen write Wertschreiben;
  .
  .
end;
```

TKnoten (Unit UGraph) enthält als Ausgangs-bzw. Vorgängerobjekt aller Knotentypen das Feld Wertliste_ vom Typ TStringlist,das an alle Nachfolgeobjekte vererbt wird.Die Wertliste_ dient dazu,um die Inhalte von Datenfelder,die bei Nachfolgeobjekten von TKnoten benutzt werden,als Elemente der Wertliste_ vom Datentyp string zu speichern.Falls diese Datenfelder von anderen Datentypen sein sollten,müssen sie zunächst in den Datentyp string konvertiert werden bzw. als string dargestellt werden.

Diese Aufgabe sowie das Erzeugen der Einträge der Stringliste übernimmt die Methode Wertlistelesen.Da TKnoten noch keine zusätzlichen Datenfelder besitzt,wird die Methode hier als abstracte und virtuelle Methode deklariert.

Abstracte Methoden dienen nur der Bereitstellung der Methode zwecks virtueller Vererbung beim Ausgangsobjekt der Vererbungskette und benötigen lediglich die Deklaration eines Methodenkopfs im Interfaceteil der Unit,nicht jedoch der Implementierung eines Quelltextes.Die Methoden gleichen Namens der Nachfolgeobjekte (mit Quelltext) werden dann mit dem Zusatz override versehen,der bedeutet,dass die Methode neu definiert wird.Es werden dadurch verschiedene Methodenimplementierung mit gleichen Bezeichner für die verschiedene Objektypen bereitgestellt (Polymorphie).

Die Methode Wertlisteschreiben dient zur Aktualisierung und Rückgabe der Wertliste.Da TKnoten noch keine Zusatzdatenfelder hat,wird diese Methode ebenfalls als virtual und abstract deklariert.

Außerdem wird,um auf die Elemente der Wertliste zugreifen zu können,für alle Nachfolgeobjekte von TKnoten die Methode Wert als Property bereitgestellt,die mittels des Datenfeldes Position_,das durch die Property Position (Methoden SetzeWertposition und WelcheWertposition) verändert werden

kann und auf den der Wertposition entsprechenden Eintrag der Wertliste mittels der Methoden Wertlesen und Wertschreiben zugreift:

```
procedure TKnoten.Wertschreiben(S:string);
begin
  if (Position>=0) and (Position<=Wertliste.Count-1)
  then
    Wertliste.Strings[Position]:=S
  else
    ShowMessage('Fehler Wertposition Wertlesen');
  Wertlistelesen;
end;

function TKnoten.Wertlesen:string;
var Stringliste:TStringlist;
begin
  Stringliste:=Wertlisteschreiben;
  if Stringliste.count=0
  then
    Wertlesen:=''
  else
    if Stringliste.Count-1<Position
   then
     Wertlesen:=''
   else
     Wertlesen:=Stringliste[Position];
end;

property Wert:string read Wertlesen write Wertschreiben;
```

Die Methoden Wertschreiben und Wertlesen sind die Write-und Read-Methoden der Property Wert.

Durch Setzen von verschiedenen Positionen kann dann mit der gleichen Property Wert auf jedes Datenfeld eines Knoten zugegriffen werden.

```
TInhaltsknoten = class(TKnoten)
  .
  .
  .
  public
    Inhalt_:string;
    constructor Create;override;

    function Wertlisteschreiben:TStringlist;override;
    procedure Wertlistelesen;override;

    procedure Wertschreiben(S:string);
    function Wertlesen:string;
    property Wert:string read Wertlesen write Wertschreiben;
  .
  .
  .
  end;
```

Das Nachfolgeobject TInhaltsknoten (Unit UInhgrph) von TKnoten enthält das zusätzliche Datenfeld Inhalt_ zur Aufnahme eines Knoteninhalts.

```
function TInhaltsknoten.Wertlisteschreiben:TStringlist;
begin
  Wertliste.Clear;
  Wertliste.Add(Inhalt_);
  Wertlisteschreiben:=Wertliste;
end;
```

Die Methode Wertlistescheiben fügt den Inhalt des Datenfeldes Inhalt_ der Wertliste hinzu.(Bei mehreren Datenfelder werden die Inhalte aller Datenfelder hinzugefügt.Bei Datenfelder anderen Typs ist hier eine Typumwandlung des Typs nötig.)

```
procedure TInhaltsknoten.Wertlistelesen;
begin
 Inhalt_:=Wertliste.Strings[0];;
end;
```

Die Methode Wertlistelesen speichert den/die Eintrag/Einträge der Wertliste wieder in den entsprechenden Datenfeldern.(Eventuell ist hier eine Typkonversion nötig.)

Die Property Wert wird durch Vererbung an alle Nachfolgeobjekte übertragen.Für diese Nachfolgeobjekte brauchen dann lediglich die Methoden Wertlisteschreiben und Wertlistelesen mit neuem Quellcode,der das Speichern und Lesen der neuen Datenfelder berücksichtigt,neu implementiert zu werden,die als virtuelle Methoden (override) an die Stelle der bisherigen treten.

Dasselbe Prinzip der Wertliste wird auch für Graphen- und Kantenobjekte verwendet.Da die Wertlisten jeweils mittels der Methoden LadeGraph und SpeichereGraph von TInhaltsgraph geladen und gespeichert werden,werden so automatisch die (Datenfeld-)Inhalte von Instanzen von Nachfolgeobjekten dieser Objekttypen geladen oder abgespeichert.

Die Standardmethode,um Instanzen von polymorphen Objekttypen und ihre Daten abzuspeichern und wiedereinzulesen,ist die Benutzung von Streams.

(Lit 61,S.367ff)

Dazu wird zunächst eine Instanz der in der Unit Classes vordefinierten Objektklasse TStream bzw. deren Nachfolgeobjekt TFilestream erzeugt:

```
Uses Classes;

Var St:TStream;

St:=TFilestream.create(Filename,fmcreate);
```

Außerdem ist die Benutzung eines von dem Objekttyp TFiler abgeleiteten Objekts TReader bzw. TWriter notwendig,wodurch die Methoden Readstring, ReadFloat,ReadInteger,Readchar und ReadBoolean sowie die entsprechenden Schreibmethoden Writestring,WriteFloat,WriteInteger,WriteChar und WriteBoolean zum Lesen und Schreiben von Daten der fünf einfachen Datentypen (als Container) zur Verfügung gestellt werden.Dazu wird eine Instanz des entsprechenden Objekts erstellt,z.B. zum Schreiben von TWriter,wobei auf die oben definierte Streamobjektinstanz als Parameter Bezug genommen wird:

```
Var W:TWriter;

W:=TWriter.Create(St,2048);
```

Der zweite Parameter ist die Größe des Puffers,die das Objekt seinem internen Datenpuffer zuweisen soll.

Die Speicherung der Objektklasse der Instanz eines Objekts erfolgt jetzt an Hand des Objektnamens mittels der Methode Writestring von W,und beim Laden muß dieser Objektklassenname wieder mittels Readstring gelesen werden.Aus dem Namen muß dann anschließend die Objektklasse rekonstruiert werden.Dazu dient die in der Unit Classes vordefinierte Function Findclass,die als Parameter den Namen der Objektklasse erwartet und die Objektklasse selber zurückgibt.Diese Function kann aber nur dann eingesetzt werden,wenn die Objektklasse zuvor mittels der vordefinierten Procedure Registerclass registriert wurde,wobei - und das bedeutet eine starke Einschränkung - nur Nachfolgerobjekttypen vom Objekttyp TPersistent,welcher ein Nachkomme von TObject ist,und nicht TObject selber registriert werden können.

(Von TPersistent leiten sich wiederum u.a. die Komponentenobjektypen der visuellen Komponentenbibliothek ab,so daß ihre Instanzen auf diese Weise gespeichert und gelesen werden können.)

Mittels der folgenden Typdeklaration ergibt sich also:

```
Type TneuesObject=class(TPersistent);
Var  Objectinstanz:TneuesObject;

Objectinstanz:=TneuesObject.Create;
```

```
W.Writestring(Objectinstanz.Classname);
```

Classname erzeugt den Namen der Objektklasse zu einer vorgegebenen Objektinstanz.

Danach könnten mittels W.Writestring bzw. den anderen Schreibmethoden die Daten des Objekts im Stream gespeichert werden.

Das Lesen der Datei erfolgt dann folgendermaßen:

```
Var Klassenname:string;
    KLasenreferenz:TClass;

RegisterClass(TNeuesObject);
S:=TFilesteam.create(Filename,fmOpenRead);
R:=TReader.create(S,2048);
Klassenname:=R.Readstring;
Klassenreferenz:=FindClass(Klassenname);
Objectinstanz:=Klasenreferenz.create;
```

Danach können die Daten des Objekts (bzw. der Instanz) mittels R.Readstring bzw. den anderen Lesemethoden des Objektes aus dem Stream gelesen werden.

Um nun diese Art des Lesens und Speicherns des Graphen als Stream auf den im Programmsystem Knotengraph verwendeten Graphtypen anwenden zu können,müßte sich TGraph statt von TObject vom Objecttyp TPersistent ableiten:

```
TGraph=class(TPersistent)
.
.
.
end;
```

Aber auch die Datenfelder der neuen (vom späteren Benutzer definierten) Knoten- und Kantenobjekttypen des Graphen müssen abgespeichert werden,und deshalb müßten sich auch die Typen TKnoten und TKante von TPersistent ableiten.

Durch die Verwendung von Listen werden jedoch TKnoten und TKante auf den Objekttyp TListe zurückgeführt,wodurch, wie weiter oben erläutert der Vorteil entsteht,dass der Zugriff auf die Knoten und Kanten der entsprechenden Knoten-und Kantenliste indiziert mittels der einfachen Syntax Knotenliste(Index) und Kantenliste(Index) erfolgen kann.TListe leitet sich durch Vererbung wiederum von TList ab.

Deshalb ist die eben geforderte Vererbung von TPersistent her leider nicht zu realisieren,da TList und TPersistent in der Objekthierachie der Objekthierachie von Delphi zwei verschiedene Nachfolger von TObject sind.

Außerdem bietet die Verwendung eines Streams auch nicht sehr viele Vorteile,da doch auch bei Verwendung von Streams das Speichern und Lesen der neuen Datenfelder eines neuen Objekttypen von einem späteren Anwender extra durch Bereitstellung von eigenen dazu geeigneten Methoden programmiert werden müßte.Dann kann man auch gleich das Prinzip der Wertliste benutzen.

Hier muß lediglich eine mögliche Datenkonversion in den Datentyp String vom Programmierer selber vorgenommen werden,während man bei Verwendung des Streamkonzepts auf die fertigen Methoden Read/WriteFloat, Read/WriteInteger,Read/Writechar und Read/WriteBoolean von TReader und TWriter zurückgreifen könnte.

Zur Datenkonvertierung stehen aber schon im Programm Knotengraph in der Unit UGraph die fertigen Funktionen Integertostring,Realtostring sowie StringtoInteger und StringtoReal zur Verügung.Der Datentyp char kann ebenfalls leicht als string und umgekehrt geschrieben werden und die Werte des Datentyp Boolean werden einfach als string ‘true’ oder ‘false’ gespeichert.Daher lassen sich solche Datenkonvertierungen für das Konzept der Wertliste leicht erstellen.

Für die Verwendung des Wertlistenkonzept contra Stream spricht auch die Tatsache,daß im Programm Knotengraph nicht völlig neue Objektypen definiert werden (deren Typen wiederhergestellt werden müssen),sondern sich alle Graphtypen von TGraph,alle Knotentypen von TKnoten und alle Kantentypen von TKante als Vorgängerobjekt ableiten.Sie unterscheiden sich von diesen Vorgängertypen lediglich durch neu hinzugekommene Datenfelder und Methoden.Die Inhalte der neu hinzugekommenen Datenfelder werden dann (nur) in der Wertliste gespeichert.

Ein weiterer Vorteil liegt schließlich darin,dass keine Registrierung der Datentypen stattfinden muß und der programmtechnische Aufwand sich dadurch vereinfacht.

Proceduren und Functionen als Parameter

Nun zu einem weiteren Eigenschaft von Delphi,die im Programm Knotengraph ausgenutzt wird.

Delphi erlaubt die Übergabe einer Function oder Procedure als Parameter einer anderen Procedure oder Function.Von dieser Möglichkeit wird im Quelltext des Programms Knotengraph öfter Gebrauch gemacht.Zu beachten ist dabei,dass leider keine Methode als Parameter übergeben werden kann.

Z.B. enthält die Methode Kantensumme von TGraph die Function Wert vom TWert mit der Definition TWert=function(Ob:TObject):Extended als Parameter.Die Methode selber ruft wiederum die Methode Wertsumme der Elemente von TListe mit diesem Parameter auf:

```
function TGraph.Kantensumme(Wert:TWert):Extended;
begin
  Kantensumme:=Kantenliste.WertsummederElemente(Wert);
end;


function TListe.WertsummederElemente(Wert:TWert):Extended;
var Index:Integer;
    Wertsumme:Extended;

  procedure SummiereWert(Ob:TObject);
  var Z:Real;
  begin
    try
      if abs(Wertsumme+Wert(Ob))<1E40
      then
        Wertsumme:=Wertsumme+Wert(Ob)
      else
      begin
        ShowMessage('Wertsumme zu groß!');
        Z:=0;
        Wertsumme:=1E40/Z;
      end;
    except
    end;
  end;

begin
  Wertsumme:=0;
  if not Leer then
    for Index:=0 to Anzahl-1 do
      SummiereWert(Element(Index));
  WertsummederElemente:=Wertsumme;
end;
```

Für die Benutzung der Methode Kantensumme muß zuvor eine geeignete Function Bewertung bereitgestellt werden,die zurückgibt,wie das ihr übergebende Objekt bewertet werden soll:

```
function Bewertung (Ob:TObject):Extended;
begin
  if  Ob is TInhaltsKante then
  begin
    if (TInhaltskante(Ob).Typ='i') or (TInhaltskante(Ob).Typ='r')
    then
      Bewertung := StringtoReal(TInhaltskante(Ob).Wert)
    else
```

```
      Bewertung:=1
  end
  else
    if Ob is TInhaltsknoten
    then
      Bewertung:=StringtoReal(TInhaltsknoten(Ob).Wert)
    else
      Bewertung:=0;
end;
```

Diese Funktion bewertet ein Objekt vom Typ TInhaltskante des Typs ‘i‘ oder ‘r‘ (Integer oder Real),indem es den Inhaltsstring der Kante in eine Integer- oder Realzahl (Extended) umwandelt.Bei einer anderen Typvorgabe (‘s‘/string) wird die Kante mit der Bewertung 1 versehen,d.h. dann werden durch die Methode Kantensumme die Anzahl der Kanten gezählt.Objekte vom Typ TInhaltsKnoten werden ebenfalls dadurch bewertet,dass ihr Inhaltsstring in eine Real-(Extended-)Zahl umgewandelt wird.Alle anderen Objekte erhalten die Bewertung Null.(Aufruf: r:=Graph.Kantensumme(Bewertung))

Je nach Anwendung können andere Bewertungsfunctionen für den jeweiligen Zweck (durch Überschreiben dieser Methode) erstellt werden.

Auf einen gravierenden Nachteil bei der Parameterübergabe von Functionen oder Proceduren in Delphi sei besonders hingewiesen.Die als Parameter übergebenden Functionen oder Proceduren müssen stets global definiert sein und können keine Unterproceduren/Functionen einer anderen Procedure oder Function oder Methode sein.

Dem Anwender werden eine Reihe von Iteratoren des Objekts TListe in der Unit Ugraph zur Verfügung gestellt:

Beispielsweise lassen mittels des Iterators FuerjedesElement die Anweisungen einer Procedure Vorgang vom Typ TVorgang für jedes Element der Liste (z.B. einer Knoten-oder Kantenliste) ausführen.

Notwendig dazu ist die globale Deklaration einer Procedure vom Typ TVorgang,die nicht Unterprocedure sein darf.

```
TVorgang=procedure(Ob:TObject);


procedure TListe.FuerjedesElement(Vorgang:TVorgang);
var Index:Integer;
begin
  If not Leer then
    for  Index:=0 to Count-1 do
      Vorgang(Items[Index]);
end;
```

Bei Verwendung dieser Iteratoren müssen also Methoden auf globale Proceduren oder Functionen zurückgreifen und bilden keine im Quelltext optisch abgeschlossene Einheit mehr,was relativ unschön ist.

Deshalb wurde an allen Stellen des Quelltextes des Programms Knotengraph bzw. der Entwicklungsumgebung Knotengraph der Iterator durch eine entsprechende For-Schleife über die Elemente der Knoten-oder Kantenliste unter Ausnutzung der indizierten Zugriffsmöglichkeiten dieser Objekte ersetzt.Bei dieser Syntax ist dann natürlich wieder der Aufruf einer Unterprocedure innerhalb der jeweiligen Methode zulässig.

Z.B.:

```
var Kliste:TKnotenliste;

Kliste.FuerjedesElement(Vorgang);
```

wird ersetzt durch:

```
var Index:Integer;

for I:=0 to Kliste.ZahlderElemente-1 do
  Vorgang(Kliste.Knoten(Index));
```

(Die oben genannten Iteratoren stehen aber natürlich jedem Anwender zur Verfügung.Siehe dazu den nächsten Abschnitt)

Im Folgenden wird jetzt zunächst ein Überblick über die weiteren,als Parameter bereitgestellten Functionen und Proceduren der Unit UGraph und danach ein Beispiel für das Handling visueller,ereignisgesteuerter Programmierung gegeben.Zum Schluss wird die besondere vorteilhafte Möglichkeit visueller Vererbung von Formen erörtert sowie auch die Nachteile des Objektmodells von Delphi gegenüber anderen objektorientierten Sprachen besprochen.

Weitere Proceduren und Functionen als Parameter

Die Procedure Handlung vom Typ THandlung erlaubt z.B. eine gleichzeitige Anweisungsfolge für einen Knoten und für jede von ihm ausgehende Kante:

```
procedure TKnoten.FueralleausgehendenKanten(Handlung:THandlung);
var Index:Integer;
begin
  for Index:=0 to AusgehendeKantenliste.ZahlderElemente-1 do
    Handlung(self,AusgehendeKantenliste.Kante(Index));
end;
```

mit `THandlung=procedure(Ob1,Ob2:TObject)`

Die Function TBedingung mit

```
TBedingung=function(Ob:TObject):Boolean;
```

erlaubt z.B. die erste Position eines Knotens in einer Knotenliste zu finden,der einer bestimmten Bedingung genügt.

```
function TListe.ErsterichtigePosition(Bedingung:TBedingung):Integer;
var Index,Stelle:Integer;
begin
  Index:=0;
  while (Index<=Count-1)and not Bedingung(Items[Index]) do
    Index:=Index+1;
  Stelle:=Index;
  if Stelle>Count-1 then Stelle:=-1;
  ErsterichtigePosition:=Stelle;
end;
```

Die Function Vergleich vom Typ TVergleich erlaubt z.B. zusammen mit Wert vom Typ TWert das Sortieren einer Liste aus beliebigen Objekten (z.B. einer Pfadliste aus Einzelpfaden) mittels Quicksort:

```
TVergleich=function(Ob1,Ob2:TObject;Wert:TWert):Boolean;

procedure TListe.Sortieren(Vergleich:TVergleich;Wert:TWert);

  procedure Quicksortieren(L,R:Integer);
  var I,J,M:Integer;
  begin
    I:=l;
    J:=R;
    M:=(L+R) div 2;
    repeat
      while Vergleich(Element(I),Element(M),Wert) do
        I:=I+1;
      while Vergleich(Element(M),Element(J),Wert) do
        J:=J-1;
      if I<=J then
      begin
        VertauscheElemente(I,J);
        I:=I+1;
        J:=J-1;
      end
    until I>J;
    if L<J then Quicksortieren(L,J);
    if I<R then Quicksortieren(I,R);
  end;

begin
  if Anzahl>1 then
    Quicksortieren(0,Anzahl-1)
end;
```

Die Function Sk vom Typ TString erzeugt zu einem Objekt einen string und erstellt z.B. eine Liste mit den Knoteninhalten einer Knotenliste:

```
TString=function(Ob:TObject):string;


function TGraph.ListemitKnotenInhalt(Sk:TString):TStringlist;
var Stringliste:TStringlist;
    Index:Integer;
begin
  Stringliste:=TStringlist.Create;
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      Stringliste.Add(Sk(Knotenliste.Knoten(Index)));
  ListemitKnotenInhalt:=Stringliste;
end;
```

Visuelle und ereignisgesteuerte Programmierung:

Das eingangs erwähnte Konzept der ereignisgesteuerten Programmierung,das in Delphi eng mit dem Prinzip der objektorientierten Programmierung verknüpft ist,soll jetzt an Hand eines Beispiels verdeutlicht werden.

Die Komponente der Komponentenbibliothek PaintBox vom Typ TPaintbox bietet eine Möglichkeit,in einem vorzugebenden rechteckigen Bereich innerhalb einer Form zu zeichnen.Dazu wird das entsprechende Symbol aus der Komponentenbiliothek auf die Form gezogen und anschließend wird der rechteckige Bereich der Zeichenfläche (Canvas vom Typ TCanvas) in der gewünschten Größe durch Ziehen mit der Maus bestimmt.Im Objektinspektor kann dann durch Anklicken der Property OnMouseDown eine entspechende Ereignismethode (vom Objekt der Form TKnotenform) PaintBoxMouseDown erzeugt werden,deren Quellcode ausgeführt wird,wenn mit der linken oder rechten Maustaste auf die Zeichenoberfläche des Bereichs der Paintbox geklickt wird.Vom System wird zunächst dabei nur eine leere Rahmenprocedure erstellt,die vom Programmierer mit Quellcode zu füllen ist:

```
procedure TKnotenform.PaintBoxMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin

 end;
```

Sender vom Typ TObject ist dabei ein Parameter,der darüber informiert,welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.Button vom Typ TButton bestimmt,welche Maustaste gedrückt wurde,Shift vom Typ TShiftState wird für Tastatur- und Mausereignissen zur Ermittlung des Status der Tasten Alt,Strg,der Umschalttasten,sowie der Maustasten beim Eintreten des Ereignisses verwendet.Die Parameter X und Y vom Datentyp Integer sind die x-und y-Koordinaten der Stelle,an der der Mausklick erfolgt.

Die Ereignisprocedure wird im Programm Knotengraph mit dem folgenden Quellcode gefüllt:

```
procedure TKnotenform.PaintBoxMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var Kn:TInhaltsknoten;
begin
  try
    if Button=mbleft then                                                    1)
    begin
      Graph.Knoteninhaltzeigen(X,Y);                                         2)
      if Graph.FindezuKoordinatendenKnoten(X,Y,Kn)=false then               3)
        Graph.Kanteninhaltzeigen(X,Y);                                       4)
      if Graph.FindezuKoordinatendenKnoten(X,Y,Kn)=true then
        Graph.LetzterMausklickknoten:=Kn;                                    5)
    end;
    if Button=mbright then
    begin                                                                    6)
      GraphK.Freeall;
      GraphK:=Graph.InhaltskopiedesGraphen(TInhaltsgraph,TInhaltsknoten,TInhaltskante,false);
                                                                             7)
      Ausgabe1.Caption:='1. Knoten wählen!';
      Ausgabe1.Refresh;
```

```
      Paintbox.OnMouseDown:=KanteerzeugenMouseDown;                         8)
      Paintbox.OnDblClick:=nil;                                             9)
      Paintbox.OnMousemove:=nil;                                           10)
      Graph.Graphistgespeichert:=false;
    end;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
  except
    Fehler;
  end;
end;
```

Wenn ein Mausklick mit der linken Maustaste erfolgte (1),wird,wenn sich in der Nähe der Stelle (x,y) des Mausklickpunktes ein Knoten befindet,der Knoteninhalt in einem Ausgabefenster angezeigt (2).Dieser Knoten wird dann bei 5) als letzter mit der Maus angewählter Knoten gespeichert.Wenn sich dort kein Knoten befindet (3),wird falls sich an der Stelle (x/y) eine Kante befindet,der Inhalt der Kante in einem Ausgabefenster angezeigt (4).

Wenn die rechte Maustaste betätigt wurde,wird der momentane Graph zunächst zwischengespeichert (7),um die nun folgende Erzeugung und Einfügung einer Kante in den Graph rückgängig machen zu können.Bei (8) wird nun der Property OnMouseDown der Paintbox eine neue Ereignisbehandlungsmethode KanteerzeugenMouseDown (von TKnotenform) zugewiesen,die exakt dieselbe Parameterliste wie die Methode PaintboxMouseDown aufweisen muß.Das Reagieren auf die Ereignisse Mausdoppelklick und Verschieben der Maus über der Paintbox werden durch die Anweisungen (9) und (10) Paintbox.OnDblClick: =nil und Paintbox.OnMousemove:= nil deaktiviert.

Auf diese Weise ist jetzt die Methode KanteerzeugenMousedown für das Reagieren auf Mausklicks zuständig:

```
procedure TKnotenform.KanteerzeugenMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  try
    Listbox.Clear;
    if Graph.Kantezeichnen(X,Y) then                                       11)
    begin
      Paintbox.OnMouseDown:=PaintboxMousedown;                             12)
      Paintbox.OnDblClick:=PaintboxDblclick;                               13)
      Paintbox.OnMousemove:=PaintboxMousemove;                             14)
      Ausgabe1.Caption:='';
      Ausgabe1.Refresh;
    end
    else
      Ausgabe1.Caption:='2. Knoten wählen!';                               15)
    Aktiv:=true;
    Graph.ZeichneGraph(Paintbox.Canvas);
  except
    ShowMessage('Fehler');
  end;
end;
```

Beim ersten Klicken mit der Maus auf einen Knoten auf der Paintboxzeichnenfläche wird jetzt durch die Methode Kantezeichnen des Graphen Graph von TKnotenform dieser Knoten als Anfangsknoten der Kante ausgewählt und der Rückgabewert dieser Functionsmethode ist false,so dass jetzt bei (15) die Ausgabeaufforderung `2. Knoten wählen!`erscheint.Bei erneutem zweiten Mausklick auf einen zweiten Knoten innerhalb der Paintboxfläche wird durch die Methode Kantezeichnen des Graphen Graph dieser Knoten als Endknoten der Kante ausgewählt,die Kante durch diese Methode erzeugt und in den Graph eingefügt.Dann ist der Rückgabewert dieser Functionsmethode true und mittels der Zuweisungen (12) bis (14) werden die ursprünglichen Ereignisbehandlungsmethoden wieder aktiviert.

Wie das Beispiel zeigt,ist es möglich Ereignisbehandlungsmethoden an die entsprechenden (Ereignis-)Propertys von Komponenten zuzuweisen.Die Parameterliste dieser Ereignisbehandlungsmethoden muß exakt mit denen der Standardereignisbehandlungsmethoden übereinstimmen.Soll das Reagieren auf ein Ereignis unterbunden werden,ist die entsprechende Property nil zu setzen.

Die Ereignisproperty des Objektes TPaintbox sind z.B.:

OnClick Einfachklick mit der Maus
OnDblClick Doppelklick mit der Maus
OnDragDrop Ziehen und Loslassen mit der Maus
OnDragOver Über eine Symbolfläche ziehen mit der Maus
OnMouseDown Maustaste drücken
OnMouseMove Mit gedrückter Maustaste ziehen
OnMouseUp Maustaste loslassen
OnPaint Dieses Ereignis tritt ein,wenn Fläche der Paintbox neu gezeichnet werden muß.Z.B.dann,wenn die Größe des Fensters vom Benutzer geändert wird,oder die Überdeckung durch ein anderes Windows-Fenster aufgehoben bzw. verändert wird:

```
procedure TKnotenform.PaintBoxPaint(Sender: TObject);
begin
  if Aktiv
  then
    Graph.ZeichneGraph(Paintbox.Canvas)
  else
    GraphH.ZeichneGraph(Paintbox.Canvas);
  Listbox.Width:=Screen.Width-20;
  Paintbox.Width:=Screen.Width;
  Paintbox.Height:=Screen.Height;
  HorzScrollBar.Range:=Screen.Width-20;
  VertScrollBar.Range:=Screen.Height-65;
end;
```

Je nach Wert des Parameters Aktiv (Property von TKnotenform) wird entweder der Graph Graph (ursprünglicher Graph) oder der Graph GraphH (Ergebnisgraph von Anwendungsalgorithmen) von TKnotenform gezeichnet.Außerdem werden die Größen von Listbox,Paintbox sowie der Scrollbars angepaßt.

Die Ereignismethode OnPaint (die es in ähnlicher Form z.B. auch in Java gibt und bei jedem notwendigen Neuzeichnen aufgerufen wird) sollte bei jeder Anwendung unbedingt mit Quellcode ausgefüllt werden,da sonst bei einer Veränderung der Fenstergröße oder Lage eventuell ein leeres Fenster entsteht.

Neben den Ereignisbehandlungsmethoden,die zu vorgegebenen Propertys von Komponenten gehören,lassen sich auch eigene Botschaftsbehandlungsmethoden schreiben.Botschaftsbehandlungsmethoden werden mit Hilfe der Anweisung message in der Methodendeklaration festgelegt.Hinter dem Wort message muß eine Integer-Konstante mit einem Wert zwischen 0 und 32767 aus der Unit Messages folgen.Diese Konstante ist der Botschaftsbezeichner (im folgenden Beispiel die vordefinierte Konstante WM-Menuselect,die einen Mausklick auf die Menüs der Menüleiste beschreibt):

Botschaftsbehandlungsmethoden müssen genau einen Referenzparameter enthalten.Als Standardbezeichner wird hier meistens das Wort message gewählt.Er ist der zu der Botschaftskonstanten zugehörige Botschaftsrecord (Unit Messages).

```
type TKnotendorm=class(TForm)
.
.
.
private

    procedure WMMenuselect(var Message:TWMMenuselect);message WM_Menuselect;
.
.
.
end;
```

Die Implementierung einer Botschaftsbehandlungsmethode sollte die ererbte Implementierung mit Hilfe einer inherited Anweisung aufrufen.Dadurch wird der ererbten Methode der Botschaftsrecord als Parameter übergeben.Wenn es

keine Vorfahrensklasse gibt,wird die virtuelle Methode Defaulthandler von TObject aufgerufen,die stets vorhanden ist.

```
procedure TKnotenform.WMMenuselect(var Message:TWMMenuselect);
begin
  if (Message.IdItem=Knotenerzeugen.Command)or(Message.Iditem=Kantenloeschen.Command)    1)
      or(Message.Iditem=Knoteneditieren.Command)or(Message.IdItem=Knotenverschieben.Command)
      or(Message.IdItem=Kantenerzeugen.Command)or(Message.Iditem=Kantenloeschen.Command)
      or(Message.Iditem=Kanteeditieren.Command)or(Message.IdItem=Kanteverschieben.Command)
      or(Message.Iditem=Graphhinzufgen.Command)or (Message.Iditem=Graphladen.Command)
      or (Message.Iditem=NeueKnoten.Command)or (Message.IDitem=UngerichteteKanten.Command)
  then
  begin
    GraphK.Freeall;
    GraphK:=Graph.InhaltskopiedesGraphen(TInhaltsgraph,TInhaltsknoten,TInhaltskante,false);
  end;
  if Graph.Abbruch then Menuenabled(true);                                                2)
  if Message.IDitem <> 0 then                                                             3)
    {if Message.IDitem<100 then}
    if (not (Message.IDitem in [1..9])){and (Message.IDitem<>8)}{Ausgabe.Command} then
    begin
      if (Message.IDitem<> Ausgabefenster.Command)and
         (Message.IDitem<> Abbruch.Command)and
         (Message.IDitem<> Demo.Command)and
         (Message.IDitem<> Pausenzeit.Command)and
         (Mainmenu.Items[0].Items[0].Enabled=true)
      then
      begin
        Paintbox.OnMouseDown:=PaintboxMousedown;
        Paintbox.OnDblClick:=PaintboxDblclick;
        Paintbox.OnMousemove:=PaintboxMousemove;
        Listbox.Height:=0;
        Listbox.Visible:=false;
        Aktiv:=true;
        GraphZ:=Graph;
        if (GraphH<>nil) and (GraphH<>Graph) then                                         4)
          if (Message.IDitem>=Knotenerzeugen{Kantenanzeigen}.Command {23?/38})
             and(Message.IDitem<>Inhalt.Command)and(Message.IDitem<>Info.Command)
          {and (Message.IDitem<=64)and (Message.IDitem<>47)} then
               {AllePfade.command=38 ,Chinesischer Brieftraeger.command=64}
          begin
            GraphH.Freeall;
            GraphH:=nil;
            Listbox.Clear;
          end;
        {if (Message.IDitem>=38)and
           (Message.IDitem<=64)and (Message.IDitem<>47)
           then
             Listbox.Clear; }
        if GraphH<>nil then
        begin
          GraphH.Zustand:=false;
          GraphH.letzterMausklickknoten:=nil;
        end;
        Ausgabe1.Caption:='';
        Ausgabe1.Refresh;
        Ausgabe2.Caption:='';
        Ausgabe2.Refresh;
      end;
      if (Message.IDitem<> Ausgabefenster.Command) and                                    5)
         (Message.IDitem<> Abbruch.Command)and
         (Message.IDitem<> Demo.Command)and
         (Message.IDitem<> Pausenzeit.Command)
      then
      begin
        Graph.Abbruch:=false;
        Abbruch.Caption:='Abbruch an';
        if Graph.demo = false then Caption:='Knotenform';
      end;
    end;
  inherited;
end;
```

Es gibt keine von Delphi vordefinierte Ereignisbehandlungsmethode,die auf das Anwählen irgendeines beliebigen Menüeintrages mit der Maus reagiert.Deshalb empfiehlt es sich,selber eine solche (obige) Ereignisbehandlungsmethode WMMenuselect (Unit UKnoten) zu definieren.

Mittels des Feldes IdItem des Records Message kann abgefragt werden,welcher Menüpunkt angewählt wurde (z.B. mittels Message.IdItem = Knotenerzeugen.Command).
So wird der letzte Graph stets zwecks eventueller Wiederherstellung zwischengespeichert,wenn eines der Untermenüs der Menüs Knoten bzw. Kanten angewählt wird,aber auch bei Graph hinzufügen,Graphladen sowie Neue Knoten,ungerichteter Graph und Neu (1).

Außerdem werden bei (2) deaktivierte Untermenüs wieder aktiviert.

Bei (3) wird,wenn nicht gerade das Menu Ausgabe gewählt wurde,die Ereignispropertys der Paintbox auf die Standardmethoden zurückgesetzt,die Listbox unsichtbar gemacht,die Ausgabelabel Ausgabe1 und Ausgabe2 gelöscht und sonstige Konstanten auf ihre Defaultwerte zurückgesetzt,z.B. Aktiv:=true (Graph Graph wird gezeichnet).

Der vorige Ereignisgraph GraphH einer Anwendung sollte vor Ausführung eines neuen Anwendungsalgorithmus aus dem Speicher gelöscht werden.Dies geschieht bei (4).

Bei (5) wird auch das Abbruch-Flag (Graph.Abbruch:=false) gelöscht,falls nicht gerade ein Untermenü des Menüs Ausgabe mit der Maus gewählt wurde.

Visuelle Vererbung von Formularen:

Jetzt soll die Möglichkeit der visuellen Vererbung von Formularen erörtert werden,die allerdings erst seit der Delphi-Version 2.0 implementiert ist.

Um die visuelle Vererbung des Formulars Knotenform auszuführen,ist zunächst das Programm Knotengraph mit der Hauptform Knotenform vom TypTKnotenform zu laden.Danach wird das Menü Datei Neu angewählt,wodurch sich sich ein Fenster mit auszuwählenden Formen öffnet.Dort ist schon das Programm (beispielsweise) unter dem Namen KProjekt als Menüauswahl eingetragen.Unter diesem Menü finden sich dann die Formen des Programms Knotengraphs.Wenn man nun die Form vom Typ TKnotenform anwählt und als Option Vererben wählt,wird ein neues Formular Knotenform1,das alle Eigenschaften des Formulars Knotenform erbt,sowie eine zugehörige Unit erzeugt

```
unit Unit1

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type

  TKnotenForm1= class(TKnotenform)
    procedure PaintBox1Click(Sender: TObject);                              (*)
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

var
  KnotenForm1:TKnotenForm1

implementation

{$R *.DFM}

procedure TKnotenForm1.PaintBoxClick(Sender: Tobject);                     (*)
begin                                                                       (*)
  inherited;                                                                (*)
end;                                                                        (*)

end.
```

(Die mit * bezeichneten Zeilen sind bei der Erzeugung der Unit nicht vorhanden.)

Die Zeile TKnotenform1=class(TKnotenform) bedeutet,dass die Form zu TKnotenform1 von TKnotenform durch Vererbung als Objekt abgeleitet wird und al-

le Eigenschaften dieser Form erbt.Zwecks Verwendung besserer Bezeichner sollte sie z.B. folgendermaßen abgändert werden:

```
TNeueForm = class(TKnotenform)
  .
  .
  .
end;
```

(Im der Entwicklungsumgebung Knotengraph wird statt TNeueForm TKnotenformular verwendet.)

Die neue Form enthält außer der Deklaration keinen Quelltext.Wenn eine Methode von TKnotenform angeklickt wird und so die Anzeige von Quelltext erzwungen wird,wie z.B. im obigen Beispiel bei der Methode Paintboxclick,so wird diese Methode so dargestellt,dass der Quelltext nur das Wort inherited enthält und lediglich die Vorgängermethode aufgerufen wird.

Die visuelle Vererbung funktioniert deshalb nur,wenn auch die Quelltextdatei der Vorgänger-Form-Unit (und nicht nur die compilierte Version),von der vererbt wird,im Projektverzeichnis vorhanden ist.

Der didaktische Vorteil der visuellen Vererbung liegt darin,dass ausgehend von einer neuen Quelltext-leeren Formular-Unit vom Anwender (Schüler) ein neues Projekt auf der Grundlage eines vorgegebenen Vorgängerprojektes erstellt werden kann,das schon alle gewünschten Eigenschaften wie Menüs oder andere visuelle Komponenten mit fertigem ausführbarem Quellcode durch Vererbung vom Vorgängerprojekte her besitzt.So ist es von Schülern möglich,ausgehend von einer vorgegebenen Form des Lehrers,d.h. einem Anwendungsfenster,das durch Vererbung schon eine Reihe von Anwendungsalgorithmen mittels seiner Menüs und Komponenten bereitstellt,in dessen leere Quelltextunit zu in diesem Anwendungsfenster nachträglich visuell erzeugten Komponenten oder Menüs weiteren Quellcode neu hinzuzufügen.

Für weitere nähere Einzelheiten hierzu siehe das Kapitel F I des Anhangs Beschreibung der Datenstruktur.

Nachteile des Objektmodells von Delphi

Auf einige Nachteile wurde schon oben hingewiesen:z.B.,dass es nicht möglich ist,Methoden als Parameter von Methoden oder Proceduren sondern nur als global definierte Proceduren zu benutzen.(In Java ist dies allerdings überhaupt nicht möglich.)Ein anderer Nachteil ist das umständliche Handling von Streams.

Ein weiterer Nachteil ist,dass Delphi (2.x) keine Mehrfachvererbung von Objekten gestattet,d.h. ein Objekt kann von mehreren Objektklassen erben bzw. mehrere Superklassen haben.Auch gibt es keine Möglichkeit wie in Java,wenigsten die Methoden von mehreren abstrakten Klassen,die noch implementiert werden müsssen,zu erben.(Schnittstellendeklaration oder Interface,vgl. dazu Kapitel B II.Nachträgliche Bemerkung:In Version Delphi 5.0 ist auch in Anlehnung an Java das Interfacemodell implementiert.)

Sehr wünschenswert für den Aufbau des Programms Knotengraph wären auch erweiterte Möglichkeit der Parametrisierung von Klassen gewesen,wodurch z.B. der Aufbau der Objektlisten Knotenliste und Kantenliste aus den unterschiedlichen Knoten-und Kantenobjekttypen und deren Verwaltung mit Hilfe von Containern bedeutend einfacher gewesen wäre.(vgl. auch dazu Kapitel B II)

Letztlich zwingt Delphi nicht nur zur ausschließlichen objektorientierten Programmerstellung,wodurch es auch möglich ist,von der reinen Objekt-Lehre abzuweichen und beispielsweise auf Datenfelder direkt zuzugreifen.

F IV Unterrichtspläne zu ausgewählten Unterrichtsstunden

1)Plan zur einführenden Unterrichts(doppel)stunde der Unterrichtssequenz „Einführung in die systematische Programmierung mit Objekten an Hand der zeichnerischen Darstellung einer Graphenstruktur“ (Konzeption EWK oder CAK)

Didaktisch-methodische Bemerkungen:

Aus der Programmierung mit Delphi und dem Umgang mit Komponenten und Formen ist den Schülern schon die grundlegende Syntax der Deklaration eines Objektes als Einheit aus Datenfeldern und Methoden sowie die Vererbung von Objekten bekannt.An Hand eines Projektes,in dem geradlinige Figuren (Häuser als Rechtecke und Quadrate) mit Hilfe der Befehle Moveto und Lineto auf einer Canvas-Fläche gezeichnet worden sind,wurden eigene Objekte schon mit Hilfe eines Constructors erzeugt sowie entsprechende Methoden als Zugriff auf die Datenfelder entwickelt.

Das Vererbungsprinzip wurde bisher jedoch noch nicht besprochen,ebenso ist noch unbekannt,dass Datenfelder von Objekten wiederum Objekte sein können.Mit dieser Stunde beginnt eine Unterrichtssequenz,die als Ziel verfolgt,an Hand des Aufbaus einer objektorientierten Graphenstruktur einerseits systematisch in die objektorientierte Programmierung einzuführen und andererseits zu zeigen,wie ein Graph mit Hilfe dieser Struktur zeichnerisch dargestellt werden kann.

Da bei der späteren Benutzung der Entwicklungsumgebung EWK die graphische Darstellung objektorientiert fertig zur Verfügung gestellt wird,sind die in dieser Unterrichtsreihe gewonnenen Kenntnisse der Möglichkeit der zeichnerischen Darstellung eines Graphen sehr nützlich,um die Wirkungsweise der in EWK enthaltenen Methoden prinzipiell zu verstehen.

Auch ist die in dieser Unterrichtsreihe enthaltene Objektstruktur des Graphen der Ausgangspunkt,auf der gemäß der Konzeption CAK die komplexe Graphenstruktur(,die erweitert auch in der Entwicklungsumgebung EWK enthalten ist,) aufgebaut wird.Daher ist dieses Einführungsprojekt grundlegend für alle weiteren Projekte zum Thema objektorientierte Programmierung und Graphenstruktur.

Die Struktur eines Graphen eignet sich infolge der Vielzahl der gegenseitige Abhängigkeiten besonders gut,um Objektbeziehungen zu verdeutlichen und somit das Programmierung mit Objekten systematisch an Hand eines nichttrivialen Problems zu lernen.

Ausgehend von der zeichnerischen Darstellung eines Graphen,wobei die Knoten als Kreise und die Knotenbezeichner als Beschriftung der Kreise in ihrem Mittelpunkt dargestellt werden,ergibt sich der Objektbegriff und die Vererbungsbeziehung (ist-Beziehung) zwischen Objekten fast von selbst,wenn man die „natürlichen“ Objekte „Mittelpunkt“ (Punkt) in dem der Knoten gezeichnet wird, “Knoten ohne Bezeichner“ und „Knoten mit Bezeichner“ betrachtet.Jedes Objekt entsteht aus dem vorigen durch Hinzufügen einer weiteren Eigenschaft.Die Modellierung des Zeichenelementes Kante in eine Objektstruktur führt auf die hat-Beziehung zwischen Objekten dadurch,dass die Anfangs-und Endknoten dem Objekt Kante zweckmäßigerweise als Datenfelder zugeordnet werden.

In dieser ersten Unterrichtsstunde der einführenden Unterrichtssequenz zum Thema Objekte und Graphen soll einerseits die gegenseitige Objektbeziehung von Knoten und Kanten sowie deren Ausgangsobjekten syntaxmäßig skizziert werden.Wegen der Kürze der Zeit können andererseits in dieser Doppelstunde nur die Ausgangsobjekte TFigur und TPunkt exakt implementiert werden.

(Statt einer Doppelstunde können natürlich auch zwei aufeinanderfolgende Einzelstunden gewählt werden.)

Beschreibung des Unterrichtsverlaufs:

Erste Stunde:

An Hand eines an die Tafel gezeichneten Beispielgraphen wird die Frage diskutiert,durch welche Angaben bzw. Eigenschaften seine zeichnerische Darstellung eindeutig bestimmt ist.

Als Ergebnis ergibt sich:

Unterscheidung der Zeichenobjekte Knoten und Kanten

Eindeutige zeichnerische Festlegung des Objekts Knoten:Ort,d.h. X/Y-Koordinaten des Mittelpunkts,Kreisradius,Bezeichner

Eindeutige zeichnerische Festlegung des Objekts Kante:Verbindungsstrecke von Anfangsknoten und Endknoten,zusätzlich mit Bezeichner

Folgende Abhängigkeiten werden erarbeitet:

Punkt mit X,Y-Koordimaten -->Knoten zusätzlich mit Radius --> Inhaltsknoten zusätzlich mit Bezeichner

Kante setzt sich aus schon zwei vorhandenen Objekten,nämlich zwei Knoten mit einer zusätzlichen Verbindungsstrecke zusammen,ist also als eigenes neues Zeichen-„Objekt“ aufzufassen und ergibt sich nicht nur durch Hinzufügen **einer** neuen Eigenschaft:

Anfangsknoten,Endknoten --> Kante -->Inhaltskante zusätzlich mit Bezeichner

Danach wird eine Liste der Zeichenobjekte in der Reihenfolge der Anzahl ihrer Eigenschaften aufgestellt und als Tabelle an die Tafel geschrieben:

Zeichen-„Objekte“:

Figur	Figur
Knoten	Kante
Mittelpunkt des Kreises der Knoten mit X,Y-Koordinaten	Verbindungsstrecke mit Anfangs-und Endknoten
Knoten als Kreis mit Radius	Kante mit Bezeichner als Inhalt
Knoten mit Bezeichner als Inhalt	

Zum Darstellen der Zeichenobjekte ist jeweils eine zu wählende Farbe sowie eine Objektzeichenfläche nötig.Diese Eigenschaften sind allen Objekten gemeinsam.Daher ist es sinnvoll zunächst ein Zeichenobjekt Figur als Ausgangsobjekt einzuführen,das diese Eigenschaften besitzt.

Auf Grund der Tabelle wird eine vorläufige Objektsyntax im Lehrer-Schüler-Gespräch an der Tafel entwickelt:(Kennenlernen der Ist-und Hat-Beziehungen)

```
type

TFigur=class(TObject)
    Zeichenflaeche_:TCanvas;
    Farbe_:TColor;
  end;

  TPunkt=class(TFigur)
    X_,Y_:Integer;
  end;

  TKnoten=class(TPunkt)
    Radius_:Integer;
   end;

  TInhaltsknoten=class(TKnoten)
    Inhalt_:string;
  end;
```

```
  TKante=class(TFigur)
    Anfangsknoten_,Endknoten_:TKnoten;
  end;

  TInhaltskante=class(TKante)
    Inhalt_:string;
  end;
```

Das Objekt TPunkt wird auf der Zeichenfläche in den Koordinaten (X,Y) gezeichnet und gelöscht.Es soll außerdem zu neuen Koordinaten (X',Y') verschoben werden können.Jedes Objekt muß außerdem einen Constructor zu seiner Erzeugung erhalten.Der Zugriff auf die Datenfelder soll jeweils über eine eigene Methode (Procedure bzw. Function) erfolgen.Die Schüler erhalten die Aufgabe den Objekten TFigur und TPunkt entsprechende Methoden zuzuordnen:

Die von den Schülern erarbeiteten Vorschläge werden danach in (ca.) folgender Form an der Tafel zusammengefaßt:

```
type

  TFigur=class(TObject)
  private
    Zeichenflaeche_:TCanvas;
    Farbe_:TColor;
  public
    constructor Create;
    function Zeichenflaeche:TCanvas;
    function Farbe:TColor;
    procedure NeueFarbe(F:TColor);
  end;

  TPunkt=class(TFigur)
  private
    X_,Y_:Integer;
  public
    constructor Create(Xk,Yk:Integer);
    function X:Integer;
    procedure NeuesX(Xk:Integer);
    function Y:Integer;
    procedure NeuesY(Yk:Integer);
    procedure Zeichnen;
    procedure Loeschen;
    procedure NeuePosition(Xneu,Yneu:Integer);
  end;
```

Zweite Stunde:

Die Schüler erhalten die Aufgabe den Quelltext der Objekte TFigur und TPunkt (in einer eigenen Unit) selbständig zu implementieren.Die Ergebnisse werden anschließend besprochen und eventuell korrigiert.

```
constructor TFigur.Create;
begin
  inherited create;
  Zeichenflaeche_:=Oberflaeche;
  Farbe_:=clblack;
  Zeichenflaeche_.Pen.Color:=Farbe_;
  Zeichenflaeche_.Font.Color:=Farbe_;
end;

function TFigur.Zeichenflaeche:TCanvas;
begin
  Zeichenflaeche:=Zeichenflaeche_;
end;

function TFigur.Farbe:TColor;
begin
  Farbe:=Farbe_;
end;

procedure TFigur.NeueFarbe(F:TColor);
begin
  Farbe_:=F;
  Zeichenflaeche_.Pen.Color:=F;
  Zeichenflaeche_.Font.Color:=F;
end;
```

```
constructor TPunkt.Create(Xk,Yk:Integer);
begin
  inherited Create;
  X_:=Xk;
  Y_:=Yk;
end;

function TPunkt.X:Integer;
begin
  X:=X_;
end;

procedure TPunkt.NeuesX(Xk:Integer);
begin
  X_:=Xk;
end;

function TPunkt.Y:Integer;
begin
  Y:=Y_;
end;

procedure TPunkt.NeuesY(Yk:Integer);
begin
  Y_:=Yk;
end;

procedure TPunkt.Zeichnen;
var HilfFarbe:TColor;
begin
  Hilffarbe:=Farbe;
  NeueFarbe(clwhite);
  Zeichenflaeche.Moveto(X-1,Y-1);
  NeueFarbe(clblack);
  Zeichenflaeche.Lineto(X,Y);
  NeueFarbe(HilfFarbe);
end;

procedure TPunkt.Loeschen;
begin
  NeueFarbe(clwhite);
  Zeichenflaeche_.Moveto(X-1,Y-1);
  Zeichenflaeche.Lineto(X+1,Y+1);
end;

procedure TPunkt.NeuePosition(Xneu,Yneu:Integer);
begin
  Loeschen;
  NeuesX(XNeu);
  NeuesY(Yneu);
  Zeichnen;
end;
```

Zum Schluß wird für Formularunit im Lehrer-Schüler-Gespräch folgender Quelltext an der Tafel erarbeitet und anschließend an den Rechnern implementiert:

```
var
  Knotenformular: TKnotenformular;
  Punkt:TPunkt;


procedure TKnotenformular.FormCreate(Sender: TObject);
begin
  Oberflaeche:=Knotenformular.Canvas;
  Punkt:=TPunkt.Create(320,240);
 end;


procedure TKnotenformular.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  Punkt.NeuePosition(X,Y);
end;
```

Übersicht über den Unterrichtsverlauf mit Zeitangaben:

Erste Stunde:

1)Erörterung der gegenseitigen Abhängigkeiten der Zeichenobjekte Knoten und Kanten eines Graphen sowie Modellierung der Objektstruktur und Objekthierachie aus einfachen Anfangsobjekten.Zusammenfassung der Ergebnisse an Hand einer Tabelle an der Tafel 15 Min

2)Umsetzung der Ergebnisse in eine vereinfachte und vorläufige Delphi-Objekttyp-Deklaration,die noch keine Methoden enthält 10 Min

3)Gruppenarbeit der Schüler:Welche Methoden sind konkret den Objekten TFigur und TPunkt zuzuordnen? 10 Min

4)Sammeln der Ergebnisse 10 Min

Zweite Stunde:

5)Implementation der Objekte samt Methoden an den Rechnern (Gruppenarbeit der Schüler) 15 Min

6)Besprechung der Ergebnisse,Korrektur 15 Min

7)Erstellen des Quellcodes der Ereignismethoden procedure TKnotenformular.FormCreate und TKnotenformular.FormMouseDown der Formularunit an der Tafel 7 Min

8)Implementation der Ereignismethoden an den Rechnern 8 Min
(Testen und Debuggen evtl. erst in der nächsten Stunde)

Lernziele:

1)Die Schüler sollen eine vorläufige vereinfachte Objektstruktur,die nur die gegenseitigen Abhängigkeiten der Objekte (ohne Methoden) zur Darstellung eines Graphen beschreibt,nennen können.

2)Die Schüler sollen die zu den Objekten TFigur und TPunkt gehörenden Methoden nennen und ihren Quellcode aufschreiben können.

3)Die Schüler sollen den Quellcode der bei 2) genannten Objekte sowie die Ereignismethoden FormCreate und FormMouseDown in der Delphi-Entwicklungsumgebung implementieren können.

4)Die Schüler sollen die Syntax der Vererbungsbeziehung von Objekten nennen und anwenden können.Insbesondere sollen sie die Vererbungskette,die zum Typ TInhaltsknoten führt,aufschreiben können.

5)Die Schüler sollen Typdeklarationen aufschreiben können,in denen Objekte als Datenfelder eines anderen Objektes enthalten sind.Insbesondere sollen sie die Typdeklaration von TKante angeben können.

2)Plan zur Unterrichtsstunde „Aufbau einer Objektliste als Grundlage der Objekt-Graphenstruktur“ (Konzeption EWK oder CAK)

Didaktisch-methodische Bemerkungen:

Die Datenstruktur des Objekt-Graphen beruht auf der Datenstruktur der Objektliste,in der Daten beliebigen Datentyps gespeichert werden können.Objekt-Listen dienen nämlich dazu,Knoten und Kanten des Graphen aufzunehmen und zu verwalten.Auch Pfade des Graphen können als Liste von Objekten des Typs Kante gespeichert werden.Ebenso werden die aus-und eingehenden Kanten eines Knotens in den Objekt-Listen AusgehendeKantenliste und EingehendeKantenliste gespeichert.Da von einem späteren Anwender noch objektorientiert Knoten und Kanten mit neu definierten Eigenschaften dem Graphen

hinzugefügt werden sollen,müssen die Listen so flexibel sein,auch veränderte Knoten-und Kantenobjekte aufzunehmen und zu verwalten.

Damit ist die Kenntnis der Struktur einer Objekt-Liste für das Verständnis der Graphenstruktur von entscheidender Bedeutung und muß sorgfältig besprochen werden.Zwar wird in der Datenstruktur des Graphen auf die von Delphi schon vorgegebene Liste vom Typ TList zurückgegriffen,jedoch ist es nötig die dort schon implementierten Methoden für die Schüler verständlich und anwendbar zu machen,damit der Aufbau des Graphen vollständig durchschaut werden kann.Um einen vollständigen Überblick zu erhalten,ist es am günstigsten eine solche Objektliste selber zu implementieren.

Dabei wird Wert darauf gelegt,dass die bei Implementation dieser Listenstruktur (TListe) verwendeten Methoden im wesentlichen dieselben Bezeichner haben,wie sie später im Projekt CAK oder EWK verwendet werden,damit die Schüler nicht umlernen brauchen.

Zurückgegriffen kann dabei auf die Implementation der schon im Unterricht als Anwendung der Zeigerstruktur entwickelten Nicht-Objekt-Liste (Zusammenfassung von Zeiger auf den Nachfolger und dem Datenfeld Inhalt mit Hilfe eines Records) sowie dessen Verwaltungsproceduren.

Durch die Anwendung des den Schülern bekannten Prinzips der Vererbung bei Objektstrukturen,ergibt sich jetzt leicht das Prinzip,die Listenelemente ohne konkretes Datenfeld zu realisieren,den einzufügenden Datentyp als Objekt zu kapseln und einem Nachfolgerobjekt des Listenelement-Objekts,nämlich TElement hinzuzufügen,sowie die Verwaltungsmethoden der Liste (TListe) unabhängig vom Datentyp zu gestalten.

Beschreibung des Unterrichtsverlaufs:

An Hand der Aufgabenstellung,einerseits die Verwaltung einer Zahlenliste sowie andererseits einer Zeichenkettenliste zu erstellen,wird die den Schülern bekannten Datenstruktur einer Liste,die ein Inhaltsfeld und einen Zeiger auf das nächste Element durch einen Record verknüpft,wiederholt und an die Tafel geschrieben:

```
type Zeiger = ^Element;
     Element = RECORD
                 Weiter : zeiger;
                 Inhalt    : INTEGER/STRING;
               end;
```

Die Diskussion der Nachteile dieser Lösung ergibt,dass der Datentyp des Inhaltsfeld von Anfang an festgelegt werden und dass die Verwaltung der Liste damit zweifach implementiert werden muß.

Als Resultat der Diskussion ergibt sich:Das Inhaltsfeld sollte fehlen und später durch Vererbung hinzugefügt werden.Dazu muß die Liste aus Objekten bestehen.

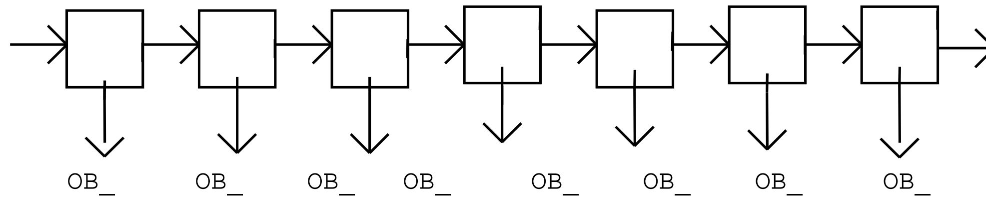
Die folgende Deklaration wird dazu im fragend-entwickelnden Unterichtsgespräch an der Tafel entwickelt:

```
type TList = class(TObject)
     private
      Weiter_:TList;
     end;

TElement=class(TList)
   private
     OB_:TObject;
   public
     constructor Create(Ob:TObject);
     function Objekt:TObject;
   end;
```

Die Datenstruktur wird graphisch veranschaulicht:

TList/TElement



Die gleichzeitige Bedeutung von TList auch als TElement wird an Hand der Zeichnung verdeutlich.Das Notwendigkeit der Typ-Konversion (je nach Auffassung) wird besprochen.

An Hand der Frage,bei welchem Objekt die Methoden zur Verwaltung der Liste zu plazieren sind,ergibt sich die Notwendigkeit ein weiteres Objekt zu definieren:

```
TListe = class(TList)
    constructor Create;
    .
    weitere Methoden
    .
    .
  end;
```

Die Bedeutung dieses Objekts als Leerelement der Liste am Anfang,das die Verwaltungsmethoden aufnimmt,wird diskutiert.
Schließlich werden mögliche Methoden zum Verwalten der Liste gesammelt und den Schülern wird als Aufgabe das Schreiben der Quelltexte zu den folgenden vier Methoden gestellt:

```
constructor TElement.Create(Ob:TObject);
begin
  inherited Create;
  Ob_:=Ob;
end;

constructor TListe.Create;
begin
  inherited Create;
  Weiter_ := nil;
end;

function TListe.Leer: Boolean;
begin
  Leer := (Weiter_=nil);
end;

procedure TListe.AmAnfangAnfuegen(Ob:TObject);
var X:TElement;
begin
  X:=TElement.Create(Ob);
  X.Weiter_:=Weiter_;
  Weiter_:=X;
end;
```

Zum Schluß wird der von den Schülern erarbeitete Quelltext an die Tafel geschrieben.

<u>Übersicht über den Unterrichtsverlauf mit Zeitangaben:</u>

1)Wiederholung der den Schülern bekannten Nicht-Objekt-Datenstruktur.Diskussion der Nachteile 5 Min

2)Diskussion einer Verbesserung durch Verwendung einer Objekt-Struktur.Entwurf der Objekte TList und TElement an der Tafel 8 Min

3)Graphische Darstellung dieser Datenstruktur.TList und TElement als Zeiger auf das gleiche Objekt.Notwendigkeit eines Leerelements am Anfang vom Datentyp TListe zur Aufnahme der Verwaltungsmethoden der Liste 7 Min

4)Sammeln von möglichen Verwaltungsmethoden der Liste:TElement.Create,TListe.Create,TListe.Leer,TListe.AmAnfangAnfuegen 8 Min

5)Schreiben des Quelltextes der bei 4) genannten Methoden durch die Schüler 10 Min

6)Besprechung der Lösungen 7 Min

Lernziele:

Die Schüler sollen die Struktur einer Objekt-Liste,in die auch noch nachträglich definierte Objekte beliebigen Typs eingefügt werden können,nennnen können.

Die Schüler sollen die Unterschiede und Gemeinsamkeiten der Objekte TList,TElement und TListe nennen können.Die Schüler sollen Namen von Verwaltungsmethoden der Liste TListe sowie von TElement angeben können.

Die Schüler sollen die Methoden TListe.Create,TElement.Create,TListe.Leer und TListe.AmAnfangAnfuegen selbstständig implementieren können.

Die Schüler sollen die Struktur und die Methoden des von Delphi vordefinierten Objekts TList zur Verwaltung von konkreten Listen benutzen können.

Die Schüler sollen die Listenstruktur der Objektliste TListe bzw. des Delphi-Objekts TList bei der (späteren) Entwicklung der Graphen-Objektstruktur anwenden können.

3)Plan zur Unterrichtsstunde „Entwicklung der Objektdatenstruktur eines Graphen" (Konzeption CAK)

Didaktisch-methodische Bemerkungen:

Die Objektdatenstruktur des Graphen bildet die Grundlage für die Implementierung des Programms Knotengraph in den Konzeptionen CAK und EWK.Dieser Stunde kommt daher eine herausragende Bedeutung zu,da in ihr einerseits das Gerüst zum weiteren Vorgehen gelegt wird.

Andererseits baut sie auf der Grundlage des vorigen Unterichtsabschnitts,nämlich der Datenstruktur der Objektliste auf und ist gewissermaßen deren Fortsetzung bzw. Anwendung.

Als weitere Voraussetzung fließen die Kenntnisse aus dem Einstiegsprojekt ein,in dem die Grundlagen zur Programmierung mit Objekten an Hand der zeichnerischen Darstellung von Knoten,Kanten eines Graphen vermittelt wurden und auf dessen Datenstrukturen jetzt aufgebaut werden kann.Die Array-Verwaltung von Knoten und Kanten ist jetzt durch die günstigere Listen-Verwaltung zu ersetzen.Es liegt nahe dazu analog wie bei der schon bekannten Objekt-Liste vorzugehen.

Durch die Vorkenntnisse ist zu erwarten,dass die entscheidenden Ideen von den Schülern selber eingebracht werden können,und es keiner starken Lenkung durch den Lehrer bedarf.Dazu ist es günstig noch einmal am Anfang die aus den vergangenen Stunden bekannte Struktur der Objektliste zu wiederholen und an die Tafel zu schreiben.

Auch die weitere Implementierung der Property-Methoden kann,da im Prinzip schon bekannt,den Schülern in Gruppenarbeit überlassen werden.Die so gestalteten Objekte werden in den folgenden Stunden um die weiteren notwendigen Methoden ergänzt.

Beschreibung des Unterrichtsverlaufs:

Zunächst wird die Datenstruktur der Objekt-Liste,wie Sie in der bisher durchgeführten Unterichtssequenz entwickelt wurde,wiederholt.Als Ergebnis wird nochmals die Datenstruktur

```
type TList = class(TObject)
     private
      Weiter_:TList;
     end;


TElement=class(TList)
   private
     OB_:TObject;
   public
     constructor Create(Ob:TObject);
     function Objekt:TObject;
   end;


TListe = class(TList)
    constructor Create;
    procedure Freeall;
    .
    .
    .
   end;
```

der Liste an die Tafel geschrieben.

An Hand eines einfachen Beispielgraphen,der ebenfalls an die Tafel gezeichnet wird,wird dann die Aufgabenstellung erläutert,eine Objekt-Datenstruktur für Graphen zu entwickeln.Dazu wird an die Datenstruktur des Graphen des Einstiegsprojekts zur objektorientierten Programmierung erinnert:

```
TGraph=Class(TFigur)
  private
    Knotenliste_:Array[1..Max] of TInhaltsKnoten;
    Kantenliste_:Array[1..Max] of TInhaltsKante;
    Knotenindex_:Integer;
    Kantenindex_:Integer;
   public
    Constructor Create;
    procedure Freeall;
    .
    .
  end;
```

Die Vorteile,die Array-Deklaration durch Listen zu ersetzen,wird im Unterrichtsgespräch herausgestellt und führt auf Grund der am Anfang der Stunde als Wiederholung erörterten Listenstruktur zu folgender Deklaration:

```
type

TGraph = class(TObject)

    Knotenliste_:TKnotenliste;
    Kantenliste_:TKantenliste;
.
.
.
end;

TKantenliste = class(TListe)
    function Kante(Index:Integer):TKante;
    property Items[Index:Integer]:TKante read Kante;
end;

TKnotenliste = class(TListe)
    function Knoten(Index:Integer):TKnoten;
    property Items[Index:Integer]:TKnoten read Knoten;
 end;
```

Die Property Items (einschließlich Syntax) wird dabei vom Lehrer als Delphi-spezifisch vorgegeben:Sie erlaubt die Adressierung der Listenelemente analog Array-Elementen.

Den Schülern wird dann die Aufgabe gestellt,die Objekte um die nötigen Propertys und Property-Methoden geeignet zu ergänzen.

```
type

TGraph = class(TObject)
  private
    Knotenliste_:TKnotenliste;
    Kantenliste_:TKantenliste;
    function WelcheKnotenliste:TKnotenliste;
    procedure SetzeKnotenliste(Kl:TKnotenliste);
    function WelcheKantenliste:TKantenliste;
    procedure SetzeKantenliste(Kl:TKantenliste);
  public
    property Knotenliste:TKnotenliste read WelcheKnotenliste Write SetzeKnotenliste;
    property Kantenliste:TKantenliste read WelcheKantenliste Write SetzeKantenliste;
    constructor Create;virtual;
    procedure Free;
    procedure Freeall;
    .
    .
    .
  end;

TKantenliste = class(TListe)
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    function Kante(Index:Integer):TKante;
    property Items[Index:Integer]:TKante read Kante;
  end;


TKnotenliste = class(TListe)
    constructor create;
    procedure Free;
    procedure Freeall;
    function Knoten(Index:Integer):TKnoten;
    property Items[Index:Integer]:TKnoten read Knoten;
  end;
```

Abschließend wird den Schülern die Aufgabe gestellt,den Quelltext der Property-Methoden zu schreiben.Die Ergebnisse werden am Schluß der Stunde besprochen.

<u>Übersicht über den Unterrichtsverlauf mit Zeitangaben:</u>

1)Wiederholung des Datentyps Liste als Objekt 5 Min

2)Erläuterung der Problemstellung:Objektdarstellung eines Graphen an Hand der Vorgabe eines an der Tafel gezeichneten Beispielgraphen 5 Min

3)Unterrichtsgespräch und Analyse: 5 Min

Die Bestandteile des Graphen sind Knoten und Kanten,die dargestellt und gespeichert werden müssen.
Herausarbeiten der Aussage „Der Graph besitzt (hat) Knoten und Kanten.".
Verwaltung der Knoten und Kanten als Elemente von Listen.

4)Ergebnis:Entwurf der Datenstruktur des Graphen an der Tafel 5 Min

5)Erste Schüleraufgabe:Ausgestaltung des Objekte durch die zugehörigen Propertys (Objektliste als Muster) 8 Min

6)Zweite Schüleraufgabe:Schreibe den Quellcode zu allen Property-Methoden. 12 Min

7)Besprechung der Schülerlösungen 5 Min

Lernziele:

Die Schüler sollen die grundlegende Datenstruktur für die weitere Implementation des Objekt-Graphen kennen und anwenden können.

Die Schüler sollen mittels Transfer die Datenstruktur der Objektliste auf den Datentyp des Graphen übertragen und deren Vorteile gegenüber der Verwendung von Arrays nennen können.

Die Schüler den Zugriff auf die Datenfelder mittels Propertys und deren Methoden implementieren können.

Die Schüler sollen die Möglichkeiten der indizierten Adressierung mittels der Property Items kennen und anwenden können.

4)Plan zur Unterrichtsdoppelstunde „Hamilton-Problem und Backtracking-Algorithmus“ (Konzeptionen CAK oder EWK)

Didaktisch-methodische Bemerkungen:

Das Backtracking-Verfahren ist ein wichtiger Algorithmus,der dazu dient in einem komplexen Problem alle Möglichkeiten nach einer Lösung zu durchsuchen,die einer oder mehreren Bedingung genügen,wobei sich eine der Bedingungen dadurch beschreiben läßt,dass eine ganzzahlige Stufenzahl einen vorgegebenen Wert erreicht,wobei die Stufenzahl mit jedem Durchlaufen einer weiteren Möglichkeit um eins zunimmt.Von diesem Verfahren leiten sich dann auch die wichtigen Algorithmen des Branch und Bound ab,wobei bestimmte Wege auf Grund von Bedingungen als Lösungsmöglichkeiten ausgeschlossen werden können.

Also sollte es Ziel des Informatikunterrichts sein,dass Schüler das Prinzip dieses wichtigen Lösungsalgorithmus kennenlernen.Da er rekursiv arbeitet,ist er für Schüler nicht leicht zu verstehen und noch schwerer durch Transfer auf neue Probleme zu übertragen.

Deshalb ist ist es wichtig,ein allgemeines Schema zu erarbeiten,das an Hand eines Beispiels anschaulich entwickelt werden kann.

Dazu eignet sich besonders das Suchen von Hamilton-Pfaden in Graphen.Ein ähnlich geeignetes,sehr verwandtes Problem ist auch das Suchen von Eulerpfaden.

Hierbei ergibt sich für die Schüler nämlich das Prinzip des Backtrackingverfahrens fast von selber,weil das intuitive manuelle Suchen in einem Graphen im wesentlichen auf das Prinzip des Backtrackingverfahrens hinausläuft.Es braucht deshalb nur,wenn es von Schülern an Hand eines Beispielgraphen durchgeführt worden ist,geeignet verbalisiert und abstrahiert zu werden.Durch Führung des Lehrers wird dann noch zusätzlich der Gedanke der rekursiven Programmierung hinzugefügt.

Die Graphenstruktur ist auf Grund ihrer vielfältigen Pfadmöglichkeiten,die an jedem Knoten jeweils neu gewählt werden können,besonders geeignet,in das Prinzip dieses wichtigen Algorithmenprinzips anschaulich einzuführen.

Statt einer Doppelstunde können natürlich auch zwei aufeinander folgende Einzelstunden gewählt werden.

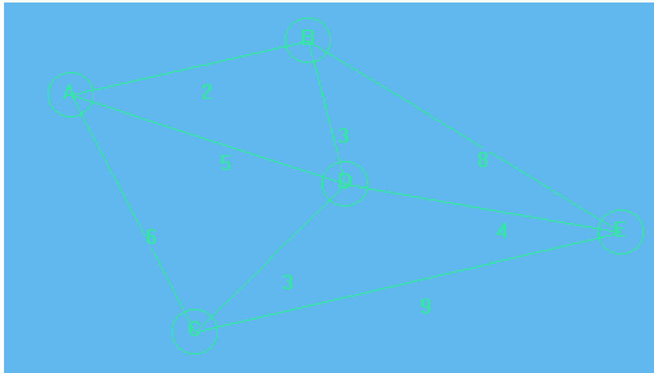
In der folgenden zweiten Stunde soll die Verbalform des Backtrackingalgorithmus als Quelltext unter Deklaration geeigneter Objekte kodiert werden.

Beschreibung des Unterrichtsverlaufs:

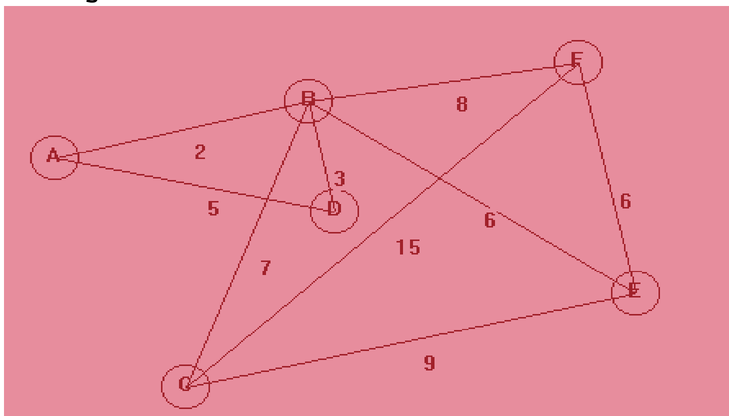
Erste Stunde:

Das Problem eines Handlungsreisenden wird erläutert.Im Anschluß daran wird als Einstieg folgende (durch Probieren zu lösende) Aufgabe gestellt:

Gesucht wird in den folgenden beiden Graphen ein Hamiltonkreis,d.h. ein geschlossener Pfad,der jeden Knoten genau einmal enthält (Startknoten jeweils A):

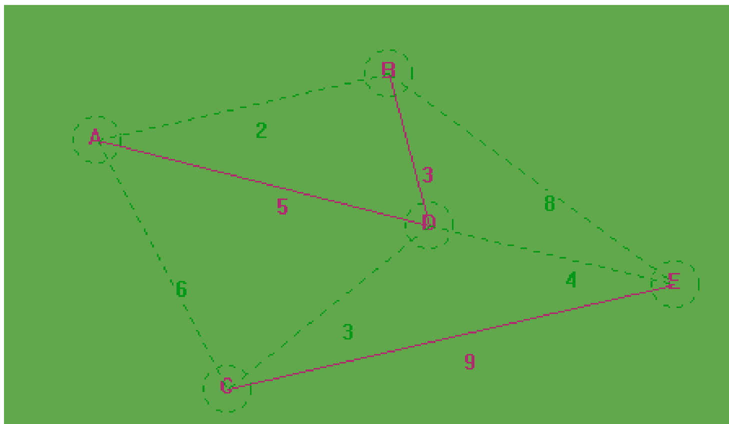


G028.gra



G029.gra

Von den Schülern wird zum ersten Graphen beispielsweise folgende Lösung gefunden:



Als Lösung zum zweiten Graph ergibt sich:Es gibt keinen Hamiltonkreis.

Aufgabe:Wie lassen sich beim ersten Graph alle Hamiltonkreise finden?

Die Versuche der Schüler,einen Pfad zu finden,werden diskutiert und danach in verbaler Form an der Tafel in dreifacher Form als Algorithmus notiert:

Diskussion:

Verfolgt man zunächst einfach einen Pfad an Hand der ausgehenden Kanten eines schon erreichten Knotens immer weiter,bis man keine Möglichkeit mehr

hat,als wieder auf einen schon besuchten Knoten zu treffen,ergibt sich beispielsweise als einfachster Rundweg die Knotenfolge ABECA.Dieser Pfad hat jedoch den Nachteil,dass nicht alle Knoten eingebunden werden konnten.

Es empfiehlt sich daher,vom Endnoten A zum Knoten C zurückzukehren,und stattdessen von dort aus den Pfad über die zweite ausgehende Kante zum Knoten D zu suchen.Von D kann man dann wiederum über die Kante DA nach A gelangen und hat damit einen möglichen Hamiltonkreis gefunden.

Nun kann man,um weitere Hamiltonkreise zu finden,den Pfad wiederum um die Kante DA verkürzen (d.h. zu D zurückgehen) und nach einer anderen Möglichkeit suchen,den Startknoten A zu erreichen.Eine solche Möglichkeit existiert jedoch nicht,da alle anderen Knoten zu denen Kanten führen,schon besucht wurden.Also wird man wiederum den Pfad um die schon durchlaufende Kante CD verkürzen,und steht dann beim momentanen Knoten C vor der gleichen Situation wie oben.Bei einer nochmaligen Verkürzung um die Kante EC hat man nun aber die Möglichkeit von E aus über D und C den neuen Gesamtpfad ABEDCA zu finden.(vgl. obige Lösung)

Der Algorithmus besteht also aus Vor-und Rückwärtsgehen,sobald der Pfad nicht mehr weiter verfolgt werden kann.Außerdem ist eine Besucht-Markierung für Knoten beim Vorwärtsgehen vorzusehen,die auf dem Rückweg wieder gelöscht wird.

Erarbeitung der Verbalform des Algorithmus:

1)Vorstufe,wird gemeinsam mit den Schülern erarbeitet:

```
procedure backtrack(var Knotenzahl:Integer;
aktuellerKnoten:TKnoten;Startknoten:TKnoten....);
var Kante:TKante;

begin

  Wiederhole

    Wähle die nächste Kante vom Typ TKante beim
    aktuellen Knoten

    Ist ein Schritt mit Hilfe dieser Auswahl möglich,um einen
    noch nicht besuchten Zielknoten (Knotenzahl+1) zu
    erreichen?

    Wenn ja

    begin

      Wähle diese Kante und diesen Zielknoten  und nimm
      diese Kante in die  Momentane Kantenliste auf.Markiere
      den Zielknoten als besucht.

      Ist damit der Startknoten wieder erreicht
      und hat Knotenzahl die Anzahl der Knoten des
      Graphen erreicht?

      dann

        Gib die Kanten der Momentanen Kantenliste
        bzw. deren Knoten aus.

      andernfalls

        Wenn Knotenzahl<Anzahl Knoten des Graphen dann

          Rufe backtrack(Knotenzahl+1,Zielknoten,
          Startkoten,...)
          auf.(Dann backtrack!)
```

```
      Markiere den Zielknoten als unbesucht.Lösche die Kante
      aus der Momentanen Kantenliste.

    end {wenn ja}

    andernfalls {wenn nein}

      Bis alle Kanten vom aktuellen Knoten aus gewählt

end;
```

Zweite Stunde:

2)Abstrakte Form:Änderungen werden vom Lehrer neben die Zeilen des vorigen Schemas parallel an die Tafel (oder Folie,falls zwei Einzelstunden) geschrieben und anschließend erläutert:

```
procedure backtrack(var Stufe:Integer;evtl. weitere Parameter);
var Auswahl:TTyp;

begin

  Wiederhole

    Wähle die nächste Auswahl vom Typ TTyp auf der aktuellen
    Stufe.

    Ist ein Schritt mit Hilfe dieser Auswahl möglich,um die
    nächste Stufe (Stufe+1) zu erreichen?

    Wenn ja

    begin

      Wähle diese Auswahl und zeichne den Schritt auf.

      Ist dann eine Lösung des Problems gefunden,
      und hat Stufe seinen maximalen Wert Max erreicht?

      dann

        Gib die Lösung aus.

      andernfalls

        Wenn Stufe<Max dann

          Rufe backtrack(Stufe+1;evtl.weitere Parameter) auf.
          (Dann backtrack!)

      Lösche die Aufzeichnung des vorigen Schrittes
      auf Grund von Auswahl.

    end {wenn ja}

    andernfalls {wenn nein}

  Bis alle Auswahlen auf dieser Stufe ausgewählt.

end;
```

3)Konkrete Form (wird mit den Schülern erarbeitet und an der Tafel notiert,hier etwas ausführlicher als an der Tafel wiedergegeben):

I)

Setze Stufe gleich 1. Wähle beliebigen Knoten als Start-und Zielknoten.Wähle leere momentane Kantenliste.

Aufruf Backtrackingmethode:

II)

Wähle vom aktuellen Knoten die nächste ausgehenden Kante aus.Wenn alle ausgehenden Kanten des aktuellen Knotens untersucht worden sind,verlasse diese Rekursionsebene der Backtrackingmethode.Falls diese die Anfangsebene ist, beende den Algorithmus.

III)

Prüfe:Stufe <= Anzahl der Knoten und Zielknoten der aktuellen Kante noch nicht besucht.
Wenn ja,gehe nach IV).Wenn nein gehe nach VIII)

IV)

Markiere Zielknoten als besucht.
Einfügen der Kante in die Kantenliste.
Stufe :=Stufe+1

V)

Prüfe:Zielknoten=Startknoten und Stufe =Anzahl der Knoten des Graphen +1 und Stufe > 2

VI)

Wenn ja:Ausgabe Lösung.Gehe nach VIII).
Wenn nein,gehe nach VII).

VII)

Aufruf der nächsten Rekursionsebene mit Stufe +1,Zielknoten und Startknoten als Parameter
(Dann backtrack!)

VIII)

Stufe:=Stufe-1,
Lösche die letzte Kante der Kantenliste.
Lösche die Besucht-Markierung dieser Kante.
Gehe nach II).

Zum Schluß wird den Schülern folgende Aufgabe gestellt:

Schreibe nach der Methode des obigen Algorithmus alle Änderungen der aktuellen Kantenliste für den obigen ersten vorgegebenen Beispielgraphen mit Startknoten A auf.

(Notierung der Kanten in der Reihenfolge:Anfangsknoten,Endknoten.Die Kanteneinfüge-Reihenfolge sei bei jedem Knoten jeweils von oben nach unten in der Zeichnung.)

Lösung: AB, ABE, ABED, ABEDA (kein Kreis),ABEDC, ABEDCA (Hamiltonkreis),ABED,ABE,ABEC,ABECA (kein Pfad),ABEC,ABE,AB

Übersicht über den Unterrichtsverlauf mit Zeitangaben:

Erste Stunde:

1)Erläuterung des Hamiltonproblems als Problem einer Rundreise 5 Min

2)Stellen und Bearbeiten der Einführungsaufgabe:Suchen eines Hamiltonkreises 15 Min

3)Diskussion des manuellen Lösungsverfahren 10 Min

4)Erarbeitung der Vorstufe des Backtrackingverfahrens und Notierung an der Tafel 15 Min

Zweite Stunde:

5)Notierung und Erörterung des abstrakten Form des Backtrackingverfahrens 15 Min

6)Erarbeiten und Notieren der konkreten Form des Backtrackingverfahrens 15 Min

7)Die Schüler bearbeiten in Gruppen die Aufgabe:Welche Pfade sind in der momentanen Kantenliste nacheinander gespeichert? 8 Min

8)Besprechung der Aufgabe 7 Min

Lernziele:

Die Schüler sollen das Prinzip des Backtrackingalgorithmus durch einen Verbalalgorithmus in konkreter und abstrakter Form beschreiben können.

Die Schüler sollen das Backtracking-Prinzip auf das Hamilton-Problem anwenden können und zu einem vorgegebenen Graphen konkret die Durchlaufreihenfolge der Pfade bzw. Knoten nennen können.

Die Schüler sollen durch die erworbenen Kenntnisse ähnliche Probleme mittels Backtracking durch Transfer lösen können.

Die Schüler sollen das Prinzip des Backtrackingverfahrens zur Erstellung von entsprechendem Quellcode zur Progammierlösung des Hamiltonproblems benutzen können.(Konzeptionen CAK oder EWK)

5)Plan zur Unterrichtsstunde „Eulerbeziehung und ebene Graphen“ (Konzeption DWK,Konzeptionen CAK oder EWK bei Fortsetzung der Unterrichtsreihe durch Programmierung des Algorithmus Bestimmung der chromatischen Zahl eines Graphen)

Didaktisch-methodische Bemerkungen:

Die Vorhersage,ob ein vorgegebener Graph isomorph zu einem planaren Graph ist,spielt eine wichtige Rolle für praktische Anwendungen,z.B.,wenn es darum geht,ob eine bestimmte elektronische Schaltung auf einer (ebenen) Schaltplatine realisiert werden kann.Während es leider kein einfaches hinreichendes Kriterium für die Planarität eines Graphen gibt,lassen sich einfache notwendige Kriterien aus der Eulerbeziehung herleiten.

Da die Eulerbeziehung auch die Kanten-Ecken-Flächen-Beziehung bei Polyedern beschreibt,ist sie schon unter diesem Aspekt ein wichtiger Unterrichtsgegenstand.Zudem ist es eine Beziehung,die Schüler an Hand von vorgegebenen Graphen selber entdecken können.In Verbindung mit dem aufbauenden Thema Planarität und Färbbarkeit ist eine Unterrichtsreihe möglich,die entdeckendes Lernen nach der Unterrichtsmethode von Wagenschein und Wittenberg ermöglicht.

Das Thema der Unterrichtsstunde baut auf zwei Sätzen über die Knoten-und Kantenzahl bei Bäumen und über die Anzahl der aus einem Graphen zu entfernenden Kanten zur Erzeugung eines Gerüsts auf.Diese Sätze sind den Schülern aus dem Unterricht der vorigen Stunden bekannt und werden anfangs wiederholt.

Das Programm Knotengraph kann als didaktisches Werkzeug eingesetzt werden,um durch Verschieben von Knoten und Kanten isomorphe Graphen zu erzeugen.Dadurch kann der Gas-Wasser-Elektrizitätsgraph,auch $K_{3,3}$-Graph genannt,auf Planarität untersucht werden.(Den für das hinreichende Kriterium nach dem Satz von Kuratowski wichtige zweite Basisgraph,den K_5 -Graph lernen die Schüler später in dieser Unterrichtsreihe bei der Einführungsaufgabe zur Färbbarkeit mittels des Erbteilungsproblems kennen,so dass damit die beiden wichtigen nichtplanaren Graphen bekannt sind)

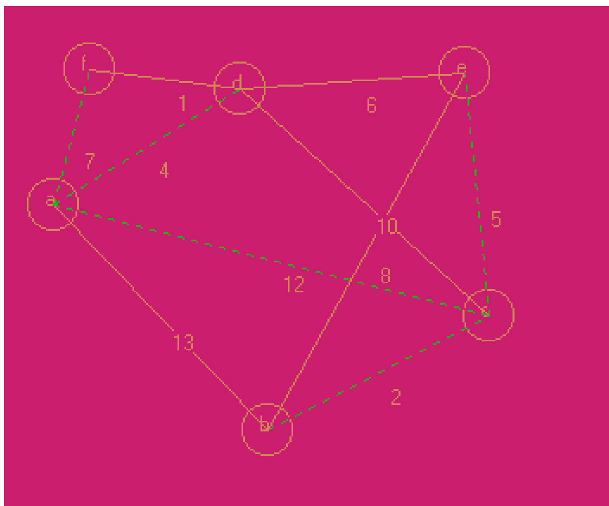
Im Vergleich dazu werden zwei andere,nämlich plättbare Graphen vorgegeben,die sich in isomorphe,ebene Graphen umformen lassen.Dies gibt Motivation zu untersuchen,worin sich die Graphen unterscheiden und führt zur Definition der Planarität von Graphen sowie zur Einführung des Gebietsbegriffs.

Durch Vergleich der Graphen ergibt sich,dass der Graph $K_{3,3}$ anscheinend zu viele Kanten für seine Knotenzahl besitzt.Als muß die Anzahl der Knoten in Relation zu den Kanten untersucht werden.Die Eulerbeziehung kann durch Darstellung der Kanten-,Knoten- und Gebietszahl in einer Tabelle leicht von den Schülern entdeckt werden.Der Beweis gestaltet sich dann relativ leicht,weil die am Anfang der Stunde wiederholten Sätze schon eine gute Grundlage für den Beweis liefern,und es so möglich ist,dass die Schüler die Beweisidee mittels leichter Lenkung durch den Lehrer weitgehend selber finden können.

Beschreibung des Unterrichtsverlaufs:

Die Unterrichtssequenz des Kapitels C I ist den Schülern aus den letzten Unterrichtsstunden bekannt.

Zunächst werden die folgenden beiden Sätze an Hand des den Schülern schon bekannten Graphen G005.gra wiederholt,der nochmals an die Tafel gezeichnet wird:



G005.gra

C Satz I.3:

In einem Baum ist stets k-a=1.

(k:Knotenzahl ,a:Kantenzahl)

l=a-k+1

C Satz I.4:

Jeder zusammenhängende schlichte Graph,der kein Baum ist,kann durch Entfernen von Kanten auf einen Baum (Gerüst des Graphen genannt) reduziert werden.Es müssen dazu stets genau l=a-k+1 Kanten entfernt werden.

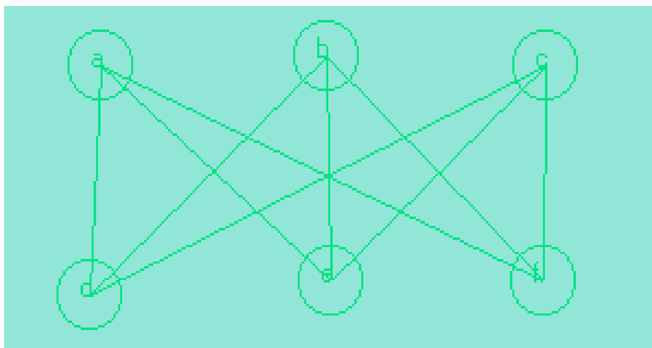
Den Schülern wird nun folgende Einstiegsaufgabe gestellt:

Gas-Wasser-Elektrizitäts-Graph:

Drei Häuser d,e und f sollen mit dem Gaswerk a,dem Wasserwerk b und dem Elektrizitätswerk c durch Leitungen verbunden werden.Stelle die Werke und Häuser durch Knoten und die Leitungen durch Kanten mittels des Programms Knotengraph dar.

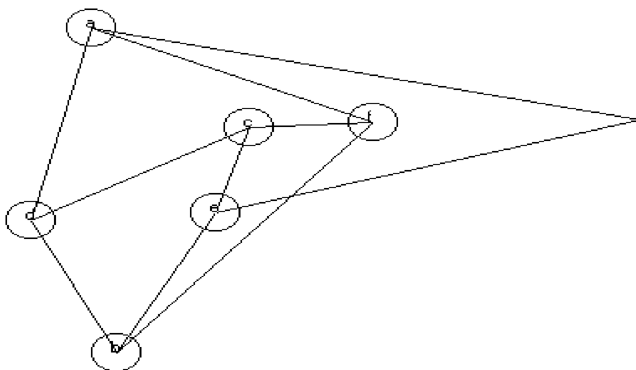
Überprüfe durch Knoten-/Kantenverschiebung mittels des Programms Knotengraph (isomorphe Graphen),ob es möglich ist,die Leitungen durch Verformungen so zu legen,dass sie sich nicht überschneiden.

Lösung:



G016.gra

Auch durch Verschiebungen kann nicht erreicht werden,dass sich die Kanten nicht überschneiden:

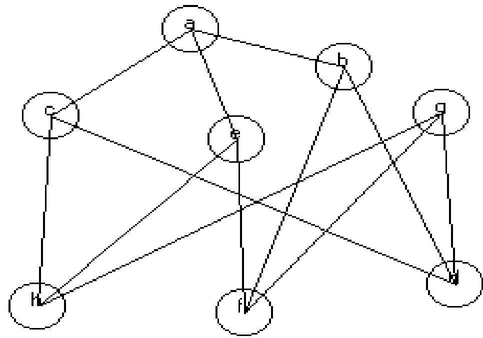


Isomorpher Graph durch Verschieben

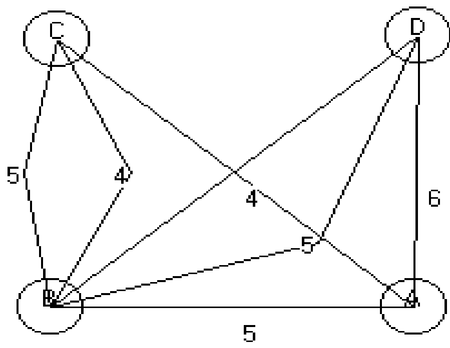
Zweite Aufgabe:

Ist es bei den folgenden Graphen möglich,keine Überschneidung der Kanten durch Erzeugen isomorpher Graphen zu erreichen?(Lösung mit dem Programm Knotengraph)

Vorgabe der Graphen G012.gra und G013.gra in der folgenden isomorphen Form:

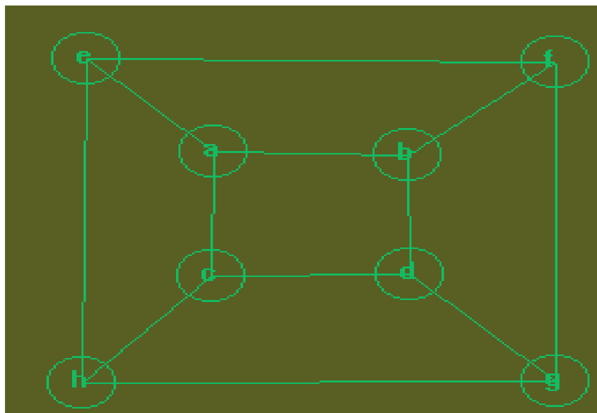


G012.gra

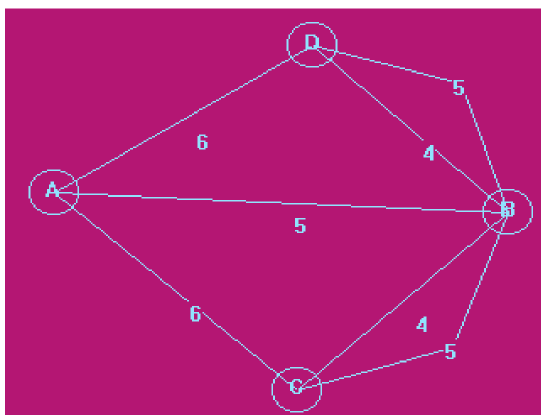


G013.gra

Lösungen:



G012.gra



G013.gra

Beobachtung:

Bei diesen Graphen ist es möglich isomorphe Graphen ohne Überschneidung der Kanten herzustellen.

Es stellt sich die Frage,ob und wie diese Eigenschaft eines Graphen voraus-gesagt werden kann.Dazu wird zunächst folgende Definition gegeben:

Definition:

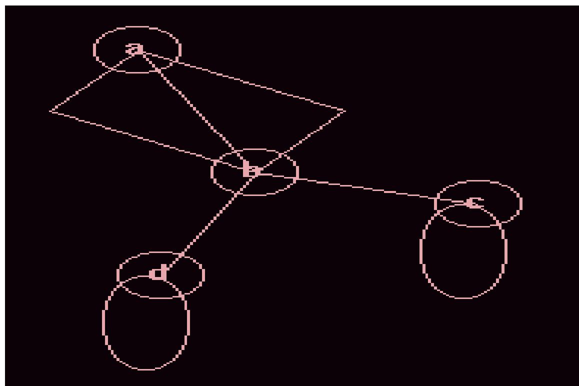
Ein Graph ist eben oder planar,wenn er (durch Verändern der Form der Kanten d.h. durch Erzeugen isomorpher Graphen,die dieselben Kanten-Knoten-Relationen besitzen) beim Zeichnen in der Ebene so dargestellt werden kann,dass sich seine Kanten nicht gegenseitig überschneiden.

Der Begriff des Gebietes wird herausgearbeitet:

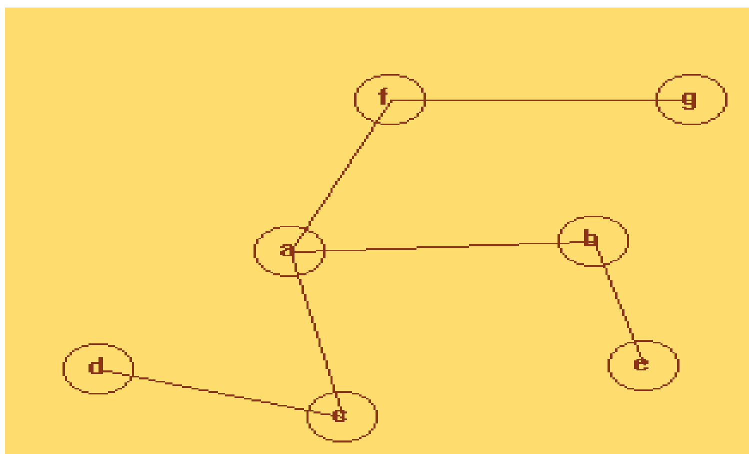
Eine von Kanten jeweils eingeschlossene ebene Fläche,die nicht mehr durch weitere Kanten geteilt wird,heißt Gebiet.

Außerdem zählt auch schon die um den Graph befindliche nach allen Seiten unbegrenzte,anfangs leere Zeichenfläche,die schon vorhanden ist,bevor der Graph gezeichnet wurde,als ein Gebiet nämlich das äußere Gebiet.

Vorgegeben werden dann noch die Graphen G014.gra und G15.gra:



G014.gra



G015.gra

Den Schülern wird zu den Graphen G012.gra bis G015.gra die Aufgabe ge-stellt,die folgende Tabelle auszufüllen.Dabei wird die Spalte k-a+f zu-nächst nicht vorgegeben,sondern soll als Beziehung zwischen f und k sowie a selber entdeckt werden.

Tabelle:

Graph	k	a	f	k-a+f
012	8	12	6	2
013	4	7	5	2
014	4	7	5	2
015	7	6	1	2
016	6	9	nicht bestimmt	nicht bestimmt

Ergebnis:

Satz (Eulersche Formel):

In einem ebenen Graphen ist k-a+f=2.

Die Schüler werden darauf hingewiesen,dass dieses nur eine notwendige Bedingung ist.

Der Beweis wird im fragend-entwickelnden Unterrichtsgespräch unter Hinweis auf die am Anfang wiederholten Sätze entwickelt:

Beweis:

Gegeben sei ein planarer Graph.Wenn der Graph noch kein Gerüst ist,können aus ihm Kanten entfernt werden,ohne dass sich die Anzahl der Komponenten erhöht,bis zum Schluß ein Gerüst entstanden ist.Dabei kann man immer nur solche Kanten entfernen,die zwei verschiedene Gebiete trennen,so dass sich bei jedem Schritt die Anzahl der Gebiete um 1 vermindert.Wenn nur noch ein Gebiet vorhanden ist,ist ein Gerüst entstanden.

Es können also l=f-1 Kanten entfernt werden.

Nach Satz C I.4 ist diese Zahl aber auch durch l=a-k+1 gegeben.

Also gilt: f-1=a-k+1

Daraus folgt: k-a+f=2

Zum Schluß wird folgende Aufgabe gestellt:

Der Algorithmus des Menüs Eigenschaften/Anzahl Knoten und Kanten des Programms Knotengraph bzw. der Entwicklungsumgebung benutzt die Eulersche Formel,um bei einem ebenen Graphen die Anzahl der Gebiete zu ermitteln.Benutze dieses Menü des Programms Knotengraph,um die Anzahl der Gebiete bei den Graphen G012.gra bis G015.gra zu ermitteln.Was zeigt sich bei dem Graphen G016.gra?

Übersicht über den Unterrichtsverlauf mit Zeitangaben:

1)Wiederholung der Sätze CI.3 und C I.4 an Hand eines geeigneten Graphen
5 Min

2)Aufgabenstellung und Lösung:

a)Gibt es die Möglichkeit,dass sich im Gas-Wasser-Elektrizitäts-Graph G016.gra die Leitungen nicht überschneiden?

b)Gibt es diese Möglichkeit bei den Graphen G012.gra und G013.gra?

Schülerlösung mittels des Programms Knotengraph durch Erzeugen isomorpher Graphen. 8 Min

3)Definition ebener Graph und Gebiet 3 Min

4)Zusätzliche Vorgabe der Graphen G014.gra und G015.gra

Ausfüllen der Tabelle zur Eulerbeziehung durch die Schüler. 8 Min

5)Sammeln der Ergebnisse.Angabe der Eulerbeziehung als Lösung. 6 Min

6)Beweisen der Eulerbeziehung im fragend-entwicklenden Unterrichtsgespräch. 10 Min

7)Aufgabe:Benutzung des Programms Knotengraph zur Ermittlung der Anzahl der Gebiete bei den Graphen G012.gra bis G015.gra mittels der Eulerformel.Was ergibt sich bei Graph G016.gra?Lösung durch die Schüler. 5 Min

Lernziele:

Die Schüler sollen die Begriffe Planarität eines Graphen und Gebiet definieren können.

Die Schüler sollen die Eulerformel als notwendige Bedingung für die Planarität eines Graphen nennen können.

Die Schüler sollen die Beweisidee der Eulerformel aufschreiben können.

Die Schüler sollen selbständig den Begriff der Planarität von Graphen sowie die Eulerformel finden können.

Die Schüler sollen den Graph $K_{3,3}$ als Beispiel für einen nichtplanaren Graph nennen können.

Die Schüler sollen die Eulerbeziehung zur Bestimmung der Nichtplanarität von Graphen einsetzen können.

6)Plan zur Doppelstunde „Optimierungsalgorithmen in Graphen (Maximaler Netzfluss und Minimale Kosten)" als Ausgangspunkt für eine Unterichtsreihe gemäß der Konzeption DWK

Didaktisch-methodische Bemerkungen:

Optimierungsprobleme der diskreten Mathematik oder des Operations Research werden im normalen Mathematikunterricht gar nicht oder nur sehr selten behandelt.Dabei würden sie gerade eine gute Ergänzung zu den üblichen Extremwertaufgaben der Analysis bilden,um zu zeigen,dass das Suchen eines Minimums oder Maximums einer Größe in einem vorgegebenen Systems nicht nur mit Hilfe von Ableitungen oder Differentialquotienten erfolgen muß.In dieser Unterrichtsdoppelstunde soll das Programm Knotengraph gemäß der Konzeption DWK als Werkzeug zur selbständigen Erarbeitung und Veranschaulichung der Algorithmen zur Bestimmung des Maximalen Netzflusses und eines Netzflusses mit minimalen Kosten eingesetzt werden,ohne die Algorithmen selber programmieren zu müssen.Durch die Benutzung des Demomodus und durch die schnelle Bestimmung der Lösungen bei Variation von Kosten,Fluss und Schranken oder sogar der Struktur des Graphen kann auf eine mühsame zeichnerische Darstellung immer neuer Graphen mit veränderten Eigenschaften verzichtet werden,und es ist möglich die Vorgehensweise des Algorithmus selber zu entdecken.

Der Algorithmus von Busacker und Gowen zur Bestimmung eines Flusses minimaler Kosten kann als Fortsetzung des Algorithmus von Ford-Fulkerson aufgefaßt werden,weil er es durch Eingabe eines größeren Flusses als des maximalen Flusses gestattet auch den maximalen Fluss zu bestimmen.

Um den Ablauf der Algorithmen noch besser zu verstehen,ist es günstig beide miteinander zu vergleichen und gegenüberzustellen.Daher sollen die Schüler in dieser Doppelstunde an Hand einer anwendungsorientierten Einführungsaufgabe die Wirkungsweise beider Algorithmen erarbeiten.Insbesondere soll die

unterschiedliche Art der Auswahl der Pfade bei beiden Algorithmen deutlich werden.

Die Kenntnis der Algorithmen gestattet es,das einfache Transportproblem ohne Vorgabe von Kosten und das komplexe Transportproblem mit der Berücksichtigung von Kosten zu lösen.

In den weiteren Stunden soll sich dann das Hitchcockproblem,das Problem des maximalen und des optimalen Matchings in bipartiten Graphen und das Chinesische Briefträgerproblem (einschließlich des Bestimmens von Eulerlinien) anschließen.Zum Schluß soll die Lösungsmethode des allgemeinen Simplexprobleme auf Graphen an Hand geeigneter Beispiele erläutert werden (vgl. Kapitel C XIII).

Einen Einstieg gemäß dieser einführenden Unterrichtsdoppelstunde in Probleme des Operations Research eignet sich auch in einer Unterrichtsreihe der Konzeption EWK,die nach Abschluß der Programmierungsprojekte als Weiterführung oder Ergänzung noch einen Ausblick auf weitere Problemstellungen der Graphentheorie geben will.

Beschreibung des Unterrichtsverlaufs der Doppelstunde:

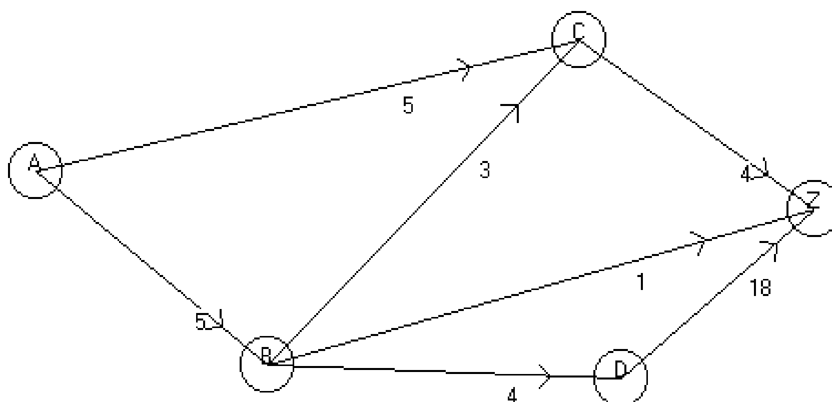
(Statt einer Doppelstunde können natürlich auch zwei aufeinanderfolgende Einzelstunden genutzt werden.)

Erste Stunde:

Den Schülern wird folgende Aufgabe auf einem Arbeitsblatt gestellt:

Gegeben sei der folgende zusammenhängende und gerichtete Graph.Die Knoten des Graphen sollen Städte und die Kanten Transportwege in einem Verkehrsverbund zwischen den Städten per Eisenbahn,Straßenbahn oder Bus bedeuten.Längs jeder Kante fährt jeweils eins der genannten Transportmittel in der durch den Kantenpfeil vorgegebenen Richtung.Vom Knoten (Stadt) A (Quelle) sollen N Personen zum Knoten (Stadt) Z (Senke) transportiert werden.(Der Knoten A hat dabei nur ausgehende Kanten und der Knoten Z nur einlaufende Kanten.)

Die Verkehrsmittel zwischen den Städten haben jeweils eine unterschiedlich große (nichtnegative)) Transportkapazität (in Tausend) pro Tag.Diese Kapazität (auch Schranke genannt) ist die als Kanteninhalt angezeigte Zahl.



G045a.gra

(Die ausgehenden Kanten des Knotens A wurden in der Reihenfolge AE,AC und von E in der Reihenfolge ED,EC,EZ erzeugt.)

Wieviel Personen können maximal pro Tag von Stadt A nach Stadt Z fahren, und auf welchen Wegen fahren sie?

(Dabei wird angenommen,dass ein eventuelles Anschlußproblem von einem Verkehrsmittel zum anderen entweder nicht existiert oder aber schon in der Angabe der Transportkapazitäten enthalten ist.)

Löse die Aufgabe mit dem Menü Anwendungen/Maximaler Netzfluss und ermittle mittels des Demomodus (mit Hilfe einer längeren Pausenzeit oder dem Einzelschrittmodus) Ablauf und Wirkungsweise des Algorithmus.

Erläuterungen:Während des Demomodus wird in den Kanten jeweils in der Form a:b die Schranke a und der vorhandene oder neu gesetzte Fluss b durch die Kante angezeigt.In den Zielknoten wird jeweils nach Untersuchung der Kante der Fluss angezeigt,um den der Fluss durch die Kante noch bis zur Schrankengrenze erhöht werden kann.Vor Untersuchung der Kante wird dieser Fluss auf Unendlich (Symbol i=Infimum) gesetzt.

(Der Quellen-und Senkenknoten wird jeweils bei den Algorithmen automatisch erkannt.)

Die Schüler erzeugen den Graphen mit Hilfe des Programms Knotengraph,speichern ihn,lassen den Algorithmus Maximaler Netzfluss zuerst ohne und dann im Demomodus ablaufen,beobachten die Vorgehensweise und machen sich Notizen.Anschließend werden die Ergebnisse gesammelt.

Ergebnisse:

Mündlich:

Der Algorithmus durchsucht systematisch alle Pfadmöglichkeiten analog zur Tiefensuche,bei mehreren aus- oder eingehenden Kanten in der Reihenfolge der Erzeugung der Kanten.Ein Pfad wird nicht mehr weiter verfolgt,wenn sich in ihm eine Kante befindet,deren Fluss schon gleich der Schranke ist.

An der Tafel:

Pfad: ABDZ Erhöhung des Flusses um 4 wegen der Schranke BD 4 und dem vorigen Fluss durch BD 0.

Pfad ABCZ Erhöhung des Flusses um 1 wegen der Schranke AB 5 und dem vorigen Fluss durch AB 4.

Pfad ACZ Erhöhung des Flusses um 3 wegen der Schranke CZ 4 und dem vorigen Fluss durch Z 1.

Pfad ACBZ (mit Rückwärtskante CB) Erhöhung des Flusses um 1 wegen der Schranke BZ 1 und dem vorhanden Fluss durch BZ 0 sowie der Rückwärtskante CB und dem vorhandenen Fluss durch BC 1.

Maximaler Fluss: 9 (Tausend) können maximal von A nach Z fahren.

Der Name des Algorithmus Ford-Fulkerson wird an die Tafel geschrieben.

Seine Wirkungsweise wird folgendermaßen zusammengefaßt und von den Schülern im Heft notiert:

A)Versuche systematisch alle Pfade vom Senkenknoten A zum Zielknoten Z analog zur Tiefensuche und bei mehreren aus- oder eingehenden Kanten in der Reihenfolge,in der die Kanten erzeugt wurden,zu durchlaufen.Dabei können sowohl Vorwärts-als auch Rückwärtskanten berücksichtigt werden.

B)Wenn auf dem Pfad der schon vorhandene Fluss durch jede Kante bei Vorwärtskanten kleiner als ihre Schranke ist und bei Rückwärtskanten größer als 0 ist,erhöhe den Fluss längs des Pfades um das Minimum der Differenz Schranke minus Fluss bei Vorwärtskanten und Fluss (>0) bei Rückwärtskanten.

C)Wenn ein Pfad eine Vorwärtskante aufweist,bei der der Fluss gleich der Schranke ist oder eine Rückwärtskante,bei der der Fluss gleich Null ist,wird das Durchlaufen des Pfades an dieser Kante abgebrochen.

D)Der Algorithmus ist beendet,wenn es keine durchlaufbare Pfade mehr gibt.Der Fluss aller Kanten aus dem Senken- bzw. in den Zielknoten ist dann der maximale Fluss.

Als Übung wird dann folgende Aufgabe gestellt:

Gegeben ist folgender Graph,dessen Knoten Städte und dessen Kanten Straßen darstellen sollen.In den Fabriken der Städte A,B,C werden Ersatzteile produziert,die von den Fabriken in den Städten D,E und F benötigt werden.

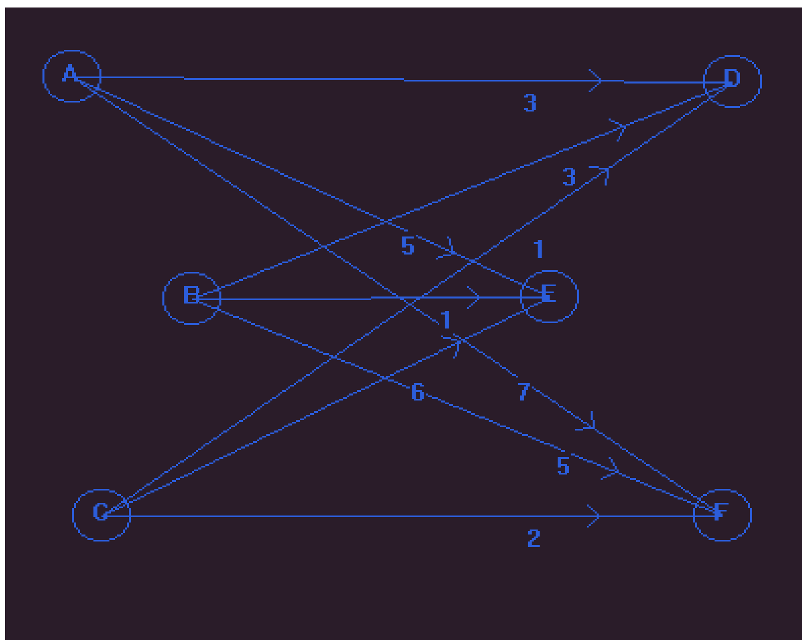
In A werden 5 Tausend,in B 3 Tausend und in C 4 Tausend Ersatzteile pro Tag produziert.In D werden 2 Tausend,in E 4 Tausend und in F 6 Tausend Ersatzteile pro Tag benötigt.Die Abnahmekapazität ist dabei gleich der Produktionskapazität.

Zwischen den Städten A,B und C einerseits und den Städten D,E und F andererseits liegt ein Straßennetz,dessen Straßen jeweils die im folgenden Graphen als Kantenschranken vorgegebenen Transportkapazitäten in Tausend pro Tag haben.

Wie ist der LKW-Verkehr bzw. dessen Ladekapazität zu organisieren,damit die produzierten Ersatzteile bei den Abnehmern in ausreichender Zahl ankommen?

(Bei allen Graphen Zahlenangaben in Tausend)

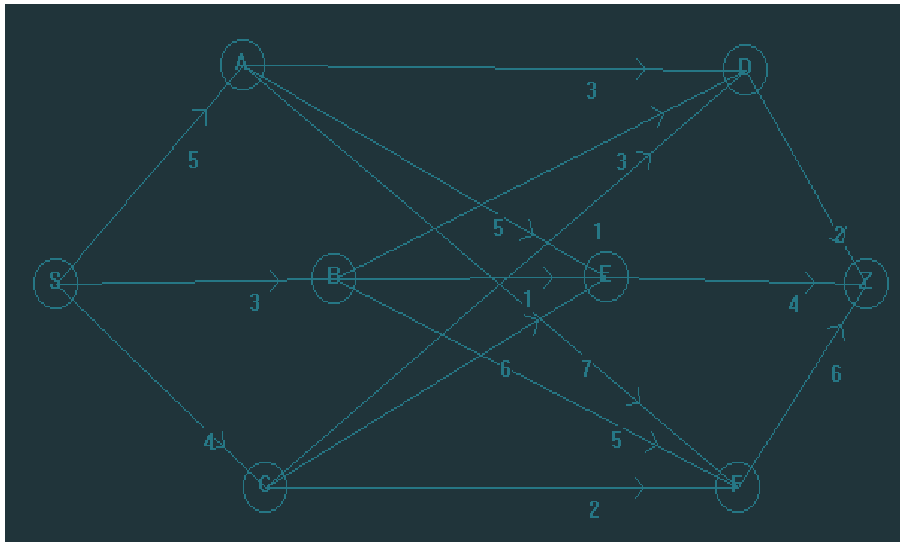
Der Graph wird an die Tafel gezeichnet:



G087.gra

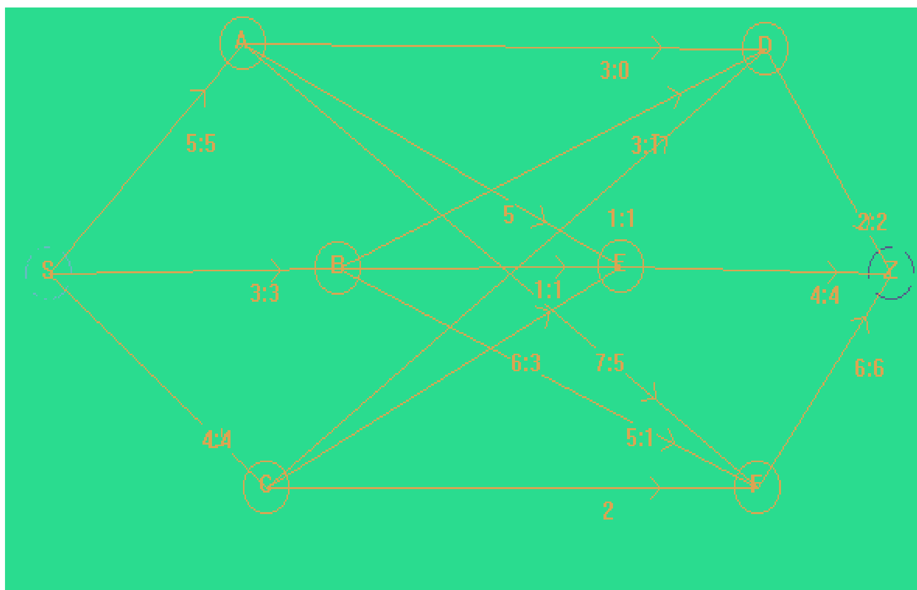
Folgende Lösung wird im Unterrichtsgespräch erarbeitet:

Der Graph wird zu einem Netz durch Startknoten S und Zielknoten Z ergänzt.Diese Knoten werden jeweils durch Kanten mit den Knoten A,B und C bzw. D,E und F verbunden,deren Schranken den Produktionskapazitäten bzw. Verbrauchskapazitäten entsprechen.



G049.gra

Anschließend erzeugen die Schüler mittels Knotengraph den Graphen und bestimmen mit dem Algorithmus Maximaler Netzfluss im Menü Anwendungen die Lösung:



G049.gra

Der Graph soll von den Schülern zwecks nochmaliger Verwendung in der zweiten Stunde gespeichert werden.

Der Fluss durch die einzelnen Kanten gibt die Transportkapazität der Wege pro Tag in Tausend zwischen den einzelnen Städten an.

Die Schülern ermitteln folgende Lösung:

Angabe in Tausend:

Kanten,Schranken und Fluss:

A->F Schranke: 7 Fluss: 5
A->E Schranke: 5 Fluss: 0
A->D Schranke: 3 Fluss: 0
B->E Schranke: 1 Fluss: 1
B->D Schranke: 3 Fluss: 1
B->F Schranke: 5 Fluss: 1

C->D Schranke: 1 Fluss: 1
C->E Schranke: 6 Fluss: 3
C->F Schranke: 2 Fluss: 0
S->A Schranke: 5 Fluss: 5
S->B Schranke: 3 Fluss: 3
S->C Schranke: 4 Fluss: 4
D->Z Schranke: 2 Fluss: 2
E->Z Schranke: 4 Fluss: 4
F->Z Schranke: 6 Fluss: 6

maximaler Gesamtfluss: 12 (Tausend)

Zweite Stunde:

Mittels eines zweiten Arbeitsblattes wird anschließend folgende Fortsetzung der Einführungsaufgabe aus der ersten Stunde gestellt:

Bei dieser Aufgabe werden die Transportkosten pro Kante berücksichtigt,die minimal werden sollen.

a)Zunächst sollen alle Kanten dieselben Transportkosten pro Kante,nämlich 1 DM haben.9 Tausend Personen wollen von Stadt A nach Stadt Z fahren.Benutze das Menü Anwendungen/Minimale Kosten des Programms Knotengraphs,um zu ermitteln,auf welchen Kanten wieviel Personen transportiert werden.Benutze dabei auch den Demomodus bzw. den Einzelschrittmodus und vergleiche mit dem Ablauf des Algorithmus von Ford-Fulkerson.

(Bemerkung:Der Algorithmus benutzt nur Vorwärtskanten.Um ihn mit dem Algorithmus des Menüs Maximaler Fluss vergleichen zu können,wird anfangs zu jeder Kante eine zusätzliche Rückwärtskante erzeugt.)

b)Die Fahrkosten pro Person (in DM oder €) sind auf den verschiedenen Transportwegen verschieden hoch und sind für die verschiedenen Kanten der folgenden Tabelle zu entnehmen:

Für alle Aufgaben:

Kante: A B Kosten: 1
Kante: A C Kosten: 1
Kante: B C Kosten: 1
Kante: B D Kosten: 1

Aufgabe I bis IV:

I)
Kante: C Z Kosten: 2
Kante: D Z Kosten: 2
Kante: B Z Kosten: 1

II)
Kante: C Z Kosten: 1
Kante: D Z Kosten: 1
Kante: B Z Kosten: 2

III)
Kante: C Z Kosten: 3
Kante: D Z Kosten: 1
Kante: B Z Kosten: 3

IV)
Kante: C Z Kosten: 3
Kante: D Z Kosten: 1
Kante: B Z Kosten: 1

Wenn nur 4 Tausend Personen von Stadt A nach Stadt Z fahren wollen,sind Wege für die Fälle I bis IV gesucht,so dass die Summe der Fahrpreise dieser Personen möglichst gering ist.

Löse die Aufgaben mit dem Menü Anwendungen/Minimale Kosten und versuche an Hand der verschiedenen Ergebnisse der Aufgabe I bis IV die Wirkungsweise dieses Algorithmus zu erkennen.

Die Schüler laden mit dem Programm Knotengraph den Graphen,lassen den Algorithmus Maximaler Netzfluss zuerst ohne und dann im Demomodus oder Einzelschrittmodus ablaufen,beobachten die Vorgehensweise und machen sich Notizen.Anschließend werden die Ergebnisse gesammelt.

Mündlich:

Bei 9 Tausend Personen ergibt sich derselbe Fluss wie bei dem Algorithmus von Ford-Fulkerson.Im Demomodus/Einzelschrittmodus ist allerdings zu erkennen,dass nicht mehr die Pfade von A nach Z in der Reihenfolge des Erzeugens der ausgehenden Kanten sondern in einer andereren Reihenfolge durchlaufen werden,nämlich nur 3 Pfade: ACZ,ABZ und dann ADZ

Die Ergebnisse von Aufgabe b) sind:

An der Tafel:

I) Pfad ACZ 3
   Pfad ABZ 1

Kosten 11

II)Pfad ACZ 4

Kosten 8

III)Pfad ABDZ 4

Kosten 12

IV)Pfad ABZ 1
   Pfad ABDZ 3

Kosten 11

Mündlich:

Die verschiedenen Verteilungen des Flusses zeigen,dass der Algorithmus stets die Pfade in der Reihenfolge der Größe der auf diesen Pfaden entstehenden Kosten durchläuft,bis aller Fluss verteilt ist.Im Demomodus oder Einzelschrittmodus wird sichtbar,dass Pfade,in denen der Fluss bei einer Kante gleich der Schranke ist,bzw. bei Rückwärtskanten der Fluss gleich Null ist,mit i d.h. unendlichen Kosten bewertet werden,so dass sie nicht mehr durchlaufen werden.Aufgabe a) demonstriert,dass wenn der maximale mögliche Fluss zu verteilen ist,die Kostenverteilung keinen Einfluß auf das Ergebnis hat.

Der Name des Algorithmus Busacker-Gowen wird an die Tafel geschrieben.

Seine Wirkungsweise wird folgendermaßen zusammengefaßt und von den Schülern im Heft notiert:

A)Suche den Kostenminimalen Pfad von der Quelle A zur Senke Z und erhöhe auf diesem Pfad den Fluss um den größtmöglichen Fluss,wobei zu berücksichtigen ist,dass der Fluss durch die Vorwärtskanten nur bis zur Schranke erhöht und bei Rückwärtskanten nur bis Null reduziert werden kann.Außerdem kann nicht mehr als der vorgegebene Fluss verteilt werden.Die Kosten pro Kante berechnen sich bei Vorwärtskanten als Produkt aus der Differenz aus

Schranke minus Fluss mal Kantenkosten,und bei Rückwärtskanten ist es das negative Produkt aus Fluss und Kantenkosten,weil der Fluss vermindert wird. Setze die Kosten auf den Vorwärtskanten,bei der der Fluss gleich der Schranke ist und bei den Rückwärtskanten,bei der der Fluss gleich Null ist, gleich unendlich (i).

B)Subtrahiere den durch den Pfad geschickten Fluss vom vorgegebenen Fluss.

C)Falls die Differenz von B) noch größer als Null ist,d.h. falls noch Fluss zu verteilen ist,wiederhole A).Sonst beende den Algorithmus.

Danach wird die Übungsaufgabe aus der ersten Stunde erweitert:

Als neue Aufgabenstellung sollen nun den Straßen als Transportwegen noch zusätzlich Kosten (in DM oder €) pro Ersatzteil und pro Tag zugeordnet werden:

Kante: A F Kosten: 6
Kante: A E Kosten: 1
Kante: A D Kosten: 2
Kante: B E Kosten: 5
Kante: B D Kosten: 3
Kante: B F Kosten: 4
Kante: C D Kosten: 2
Kante: C E Kosten: 3
Kante: C F Kosten: 4

Gefragt ist jetzt,wieviel Ersatzteile von A,B,C auf welchen Straßen nach D,E und F transportiert werden sollen,damit die Produktions-und Abnahmekapazität jeweils ausgeschöpft werden,die Transportkapazitäten (Schranken) berücksichtigt werden,und noch zusätzlich die Transportkosten minimal unter allen Möglichkeiten sind.

Auf Grund der Übungsaufgabe der ersten Stunde können die Schüler folgenden Lösungsweg in Gruppen selber erarbeiten:

Es wird wieder wie oben ein Graph mit Quellenknoten A und Senkenknoten Z erzeugt,deren (ausgehende bzw. einlaufende) Kanten wie oben zu den Knoten A,B,C bzw. D,E und F führen und mit den obigen Schranken bewertet sind.

Zusätzlich werden jetzt jeder der Kanten des ursprünglichen Graphen die oben genannten Kosten zugewiesen.

Den neu hinzugekommenen Kanten zum Quellen-und Senkenknoten werden jeweils die Kosten Null zugeordnet.

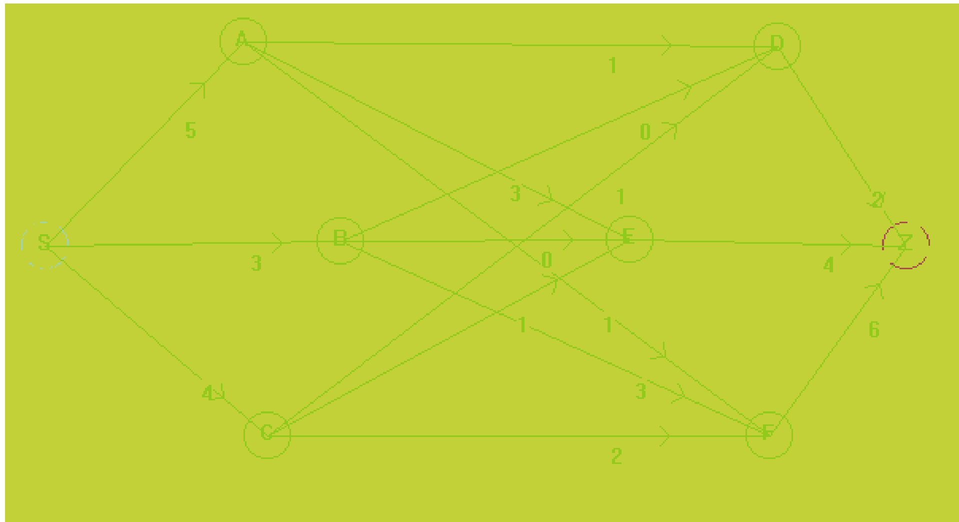
Jetzt wird auf dem Graph ein Fluss gemäß dem Algorithmus Minimale Kosten (Menü Anwendungen) erzeugt.Als Vorgabefluss wird jetzt die Zahl 12 (Tausend) gewählt,die gerade der Summe der Produktions-und Abnahmekapazitäten entspricht.

Die ermittelte Lösung für Fluss und Kosten ist die gesuchte Lösung des gestellten Problems,da es sich um einen Fluss handelt,der minimal bezüglich der Kosten ist,und da die zusätzlichen Kanten keine Kosten verursachen.

Außerdem werden die Schrankenbedingungen eingehalten und durch die zusätzlichen Kanten zum Quellen-und Senkenknoten fließt gerade soviel Fluss,wie es jeweils den Produktions- und Abnahmekapazitäten in den Knoten (Städte) A,B,C und D,E und F entspricht.

Anschließend laden die Schüler den Graph aus der letzten Stunde,starten den Algorithmus des Menüs Anwendungen/Minimale Kosten,geben die Kosten pro Kante ein und ermitteln folgende Lösung:

Der folgende Graph zeigt den Fluss (die Anzahl der Ersatzteile pro Tag in Tausend auf den Transportwegen) an:



G049.gra

Angaben in Tausend bis auf Kosten:

Kante: A F Fluss: 1 Flusskosten: 6 Schranke: 7 Kosten: 6
Kante: A E Fluss: 3 Flusskosten: 3 Schranke: 5 Kosten: 1
Kante: A D Fluss: 1 Flusskosten: 2 Schranke: 3 Kosten: 2
Kante: B E Fluss: 0 Flusskosten: 0 Schranke: 1 Kosten: 5
Kante: B D Fluss: 0 Flusskosten: 0 Schranke: 3 Kosten: 3
Kante: B F Fluss: 3 Flusskosten: 12 Schranke: 5 Kosten: 4
Kante: C D Fluss: 1 Flusskosten: 2 Schranke: 1 Kosten: 2
Kante: C E Fluss: 1 Flusskosten: 3 Schranke: 6 Kosten: 3
Kante: C F Fluss: 2 Flusskosten: 8 Schranke: 2 Kosten: 4
Kante: S A Fluss: 5 Flusskosten: 0 Schranke: 5 Kosten: 0
Kante: S B Fluss: 3 Flusskosten: 0 Schranke: 3 Kosten: 0
Kante: S C Fluss: 4 Flussosten: 0 Schranke: 4 Kosten: 0
Kante: D Z Fluss: 2 Flusskosten: 0 Schranke: 2 Kosten: 0
Kante: E Z Fluss: 4 Flusskosten: 0 Schranke: 4 Kosten: 0
Kante: F Z Fluss: 6 Flusskosten: 0 Schranke: 6 Kosten: 0

Vorgegebener Fluss durch Quelle oder Senke: 12 (Tausend)

Erreichter Fluss durch Quelle oder Senke: 12 (Tausend)

Gesamtkosten: 36 (minimale Kosten) (Tausend)

Übersicht über den Unterrichtsverlauf mit Zeitangaben:

Erste Stunde:

1)Stellen und Erläuterung der Einführungsaufgabe,Zeichnen des Graphen an der Tafel 5 Min

2)Die Schüler bearbeiten die Aufgabe an den Rechnern mit Hilfe des Programms Knotengraph,starten den Algorithmus des Menüs Anwendungen/Maximaler Fluss ohne und mit Demomodus/Einzelschrittmodus.Sie versuchen die Vorgehensweise des Algorithmus zu erkennen und zu notieren. 8 Min

3)Sammeln der Ergebnisse mündlich und an der Tafel 5 Min

4)Zusammenfassung des Algorithmus von Ford-Fulkerson in Verbalform.Die Schüler notieren ihn in ihren Heften. 8 Min

5)Stellen der Übungsaufgabe,Zeichnen des Graphen an der Tafel 5 Min

6)Erarbeitung des Lösungsweges 6 Min

7)Ermittlung der Lösung der Aufgabe mittels des Programms Knotengraph an den Rechnern 5 Min

8)Zusammenfassung der Ergebnisse 3 Min

Zweite Stunde:

1)Stellen und Erläuterung der Weiterführung der Einführungsaufgabe 5 Min

2)Die Schüler bearbeiten die Aufgabe an den Rechnern mit Hilfe des Programms Knotengraph,starten den Algorithmus des Menüs Anwendungen/Minimale Kosten ohne und mit Demomodus/Einzelschrittmodus.Sie versuchen die Vorgehensweise des Algorithmus insbesondere im Vergleich zu dem Algorithmus von Ford-Fulkerson erkennen und zu notieren. 8 Min

3)Sammeln der Ergebnisse mündlich und an der Tafel 5 Min

4)Zusammenfassung des Algorithmus von Busacker-Gowen in Verbalform.Die Schüler notieren ihn in ihren Heften. 8 Min

5)Stellen der Erweiterung der Übungsaufgabe,Angabe der Kosten der Kanten 4 Min

6)Erarbeitung des Lösungsweges durch die Schüler in Gruppen 5 Min

7)Zusammenfassung der Ergebnisse 3 Min

8)Ermittlung der Lösung der Aufgabe mittels des Programms Knotengraphs an den Rechnern 4 Min

9)Zusammenfassung der Ergebnisse 3 Min

Lernziele:

1)Die Schüler sollen die Algorithmen von Ford-Fulkerson und Busacker-Gowen in Verbalform beschreiben können.

2)Die Schüler sollen die Algorithmen von Ford-Fulkerson und Busacker-Gowen zur Ermittlung von Lösungen auf geeignete Graphen anwenden können.

3)Die Schüler sollen das einfache und das komplexe Transportproblem mit Hilfe der entsprechenden Menüs des Programms Knotengraph lösen können und geeignete Graphen dazu erzeugen können.

4)Die Schüler sollen die Unterschiede zwischen den Algorithmen von Ford-Fulkerson und Busacker-Gowen insbesondere bei der Auswahl der Pfade,auf denen der Fluss erhöht wird,nennen können.

5)Die Schüler sollen die Kenntnisse der Algorithmen von Ford-Fulkerson und Busacker-Gowen in den kommenden Unterrichtsstunden auf die Lösungen des Hitchcockprobem,des maximalen und optimalen Matchingsproblems in bipartititen Graphen,des Chinesische Briefträgerproblems und des Simplexproblems auf Graphen anwenden können.

F V Ergebnisse und Antworten von Fragen an Schüler zu den Unterrichtsreihen,Beispiel einer Facharbeit,UML-Diagramme,Update

Fragen zur Unterrichtsreihe Aufbau einer objektorientierten Graphenstruktur und Algorithmen auf Graphen

1)Beurteilen Sie allgemein den Schwierigkeitsgrad der Programmierung mit Objekten in den Jahrgangsstufen 11.2,12 und 13 gegenüber der Ihnen aus der Jahrgangsstufe 11.1 bekannten konventionellen (imperativen) Programmierung mittels der Delphi-1.0-Textfesteranwendung.

a)Objektorientierte Programmierung ist schwerer zu verstehen.

b)Nach einer gewissen Einarbeitungszeit ist das Programmieren mit Objekten von gleichem Schwierigkeitsgrad.

c)Das Programmieren mit Objekten ist leichter als das Programmieren mit imperativer Programmierung in der Jahrgangsstufe 11.1.

Antwort : a b c

Antworten:

Als Antwort wurde zu 5% Möglichkeit a,zu 58% Möglichkeit b und zu 37% Möglichkeit c gewählt.

2)Die Graphenstruktur wurde objektorientiert aus der Listenstruktur abgleitet.Beurteilen Sie Ihr Verständnis für Aufbau und Struktur des Graphen auf dieser Grundlage:

a)eher leicht

b)mittelschwer

c)schwer und anspruchsvoll

Antwort: a b c

Antworten:

Möglichkeit a 32%,Möglichkeit b 54% und Möglichkeit c 14%

3)Für die Programmierung der Graphenalgorithmen stand schon ein fertiger Graph als Objekt sowie dessen Methoden fertig zur Verfügung.Auf dieser Grundlage konnten die neuen Graphentypen und ihre Methoden durch Vererbung abgeleitet werden.Wurde durch diese Objektkonzeption das Erstellen von Quellcode zur Lösung von neuen Graphenproblemen erleichtert oder sollte ein Graphenproblem besser von Grund auf neu programmiert werden?

Antworten:

Für 82% der Schüler ergibt sich eine Erleichterung durch die Objektkonzeption,9% meinen,dass eine komplette Neuprogrammierung günstiger wäre,und bei 9% ist eine Tendenz aus den Antworten nicht erkennbar.

Auswahl aus den Schülerantworten:

„Eine Erleichterung“,„Bei dieser Vorgehensweise wurde die Arbeit sehr erleichtert.“,„Im Prinzip erleichtert,obwohl durch Neuprogrammierung besseres Verständnis.“

4)Für die Programmierung der Graphenalgorithmen stand schon eine fertige Anwenderoberfläche als Objekt Knotenformular zur Verfügung,die um Menüs zum Aufruf selbst zu erstellender Graphenalgorithmen erweitert werden konnte.Wurde durch diese Objektkonzeption das Lösen von neuen Graphenproblemen

erleichtert oder sollte die graphische Oberfläche beim Erstellen eines neuen Algorithmus ebenfalls jedesmal von Grund auf neu programmiert werden?

Antworten:

In etwa ergibt sich sich dieselbe Antworttendenz wie bei der vorigen Frage:86% empfinden die Vorgabe der Oberfläche als Erleichterung,während 5% meinen,sie sollte von neuem programmiert werden.Bei 9% ist die Tendenz der Antwort nicht zu erkennen.

Auswahl aus den Schülerantworten:

„Siehe Frage 3",„Nein,vorhandene Oberfläche ist bessser",Die Oberfläche sollte selbst einmal erstellt werden oder am vorprogrammierten Quelltext besprochen werden.Danach ist es naheliegend stets die vorhandene Oberfläche zu verwenden,da dies die Zielsetzung der objektorientierten Programmierung ist."

5)Ist es für die erfolgreiche Programmierung von Graphenalgorithmen erforderlich,die objektorientierte Graphenstruktur durch eigenes Programmieren zunächst an Hand einer (einfacheren) textorientierten Abwendung kennenzulernen (wie in Ihrem Unterrichtsgang) oder sollte man sofort zu einer graphischen Oberfläche (vgl. Frage 4) unter Bereitstellung der entsprechenden Objekte und Methoden übergehen?

Antworten:

Die Antworten für und wieder textorientierte Anwendungen gegen graphische Oberfläche halten sich mit 52% zu 48% die Waage.

Auswahl aus den Schülerantworten:

„Direkt mit graphischer Oberfläche beginnen,da es anschaulicher ist",„Man sollte zuerst mit einer einfachen Anwendung beginnen,weil man sich dann leichter mit dem Programmieren auskennt und schon von Anfang an Grundlagen hat.",„Der bisherige Unterrichtsgang ist wohl besser (gemeint ist die Reihenfolge Textanwendung,graphische Oberfläche),da man so mehr Hintergrundwissen erhält.",„Sofort graphische Oberfläche,da nur Leute,die wirklich Informatikverständnis und große Kompetenz haben,Informatik wählen sollten."

6)
a)Wie schätzen Sie die Schwierigkeit der Algorithmen Eulerlinie,Hamiltonlinie,Tiefensuche ein?

Eulerlinie:

Hamiltonkreis:

Tiefensuche:

Antworten:

Der Algorithmus zur Eulerlinie wird von allen Schüler als leicht empfunden.Während 62 % der Schüler diese Einschätzung auch bezüglich der Algorithmen zum Hamiltonkreis und der Tiefensuche teilen,geben 30% diesen Algorithmen den Schwierigkeitsgrad mittel.5% bezeichnen diese Algorithmen als schwer und 3% als sehr leicht.

Der Grund für diese Antworten dürfte darin liegen,dass das Backtrackingverfahren an Hand des Hamiltonproblems neu besprochen und eingeführt wurde und dann die die den Schüler nun schon bekannte Art des Algorithmus auf das Eulerproblem mit leichten Modifikationen übertragen wurde. (Transfer).

Auswahl aus den Schülerantworten:

„Die Algorithmen sind leicht nachvollziehbar.",„ziemlich leicht",„Die Programmierungsansätze waren durch die Kopplung an Objekte recht übersichtlich.",„mittel"

b)Wurde das Prinzip des Backtrackingalgorithmus an Hand dieser Algorithmen hinreichend erläutert,so dass Sie es selbständig einsetzen können?

Antworten:

85% der Schüler sind der Meinung,dass die Besprechung so hinreichend war,dass sie diesen Algorithmus noch zum jetzigen Zeitpunkt selbständig einsetzen können.Die anderen trauen sich das momentan nicht mehr zu.

Auswahl aus den Schülerantworten:

„Ich denke mittlerweise wohl nicht mehr.",„Ja",„Die Besprechung hat vollauf genügt.",„Nach kurzer Einarbeitungszeit müßte ich dieses Verfahren eigentlich noch beherrschen."

7)Welche (der im Unterricht behandelten) Algorithmen/Probleme der Graphentheorie finden Sie besonders wichtig bzw. besonders interessant:

Antworten:

Als Antworten wurde überwiegend die Tiefensuche und davon abgeleitete Algorithmen wie Kreissuche oder Minimale Pfadsuche,aber auch Hamilton-und Eulerlinien als spezielle Verfahren und das Backtrackingverfahren allgemein genannt.

Auswahl aus den Schülerantworten:

„Tiefensuche,weil er sehr anschaulich ist als auch von seiner Funktion sehr häufig auftritt.",„Alle Kreise suchen",„Von besonderer Bedeutung halte ich das Bestimmen der kleinsten Entfernung zwischen zwei Städten",„Problem des Handlungsreisenden",„Zeitnetzplan",„Endlicher Automat"

8)Konnten Sie im Unterricht selbständig den Quellcode für die vorgegebenen Aufgabenstellungen zu Graphen entwickeln?

Antworten:

Die Antworten lauten auf diese Frage zu 75% mit einem uneingeschränkten ja und zu 20% mit einem eingeschränkten ja.5% machen keine Angabe.

Auswahl aus den Schülerantworten:

„Ja",„Nicht komplett,im Prinzip und von der Grundstruktur aber schon.", „Die Quellcodeentwicklung war nicht besonders schwer.Schwer war das Finden von Fehlern."

9)Würden Sie jetzt am Ende der Unterrichtsreihe für noch zusätzlich weitere gestellte Graphenprobleme den Quellcode selbständig entwickeln können?

Antworten:

Es ergibt sich hier im wesentlichen dasselbe Ergebnis wie bei Frage 8.

Auswahl aus den Schülerantworten:

„Ja",„Bestimmt",„Eventuell eingeschränkt",„Ich denke schon.",„Kommt auf das Problem an."

10)Verbesserungsvorschläge für die Unterrichtsreihe zum Thema objektorientierte Datenstrukturen und Graphen.

Antworten:

Ein Teil der Schüler hätte gerne noch selbständiger ohne Vorlage zu bestimmten Algorithmen gearbeitet (45%),ein anderer Teil hätte gerne mehr Zeit gehabt,um sich mit einem speziellen Problem ausführlicher zu beschäftigen und dafür lieber ein Graphenproblem weniger behandelt (35%),der Rest ist mit der Unterrichtsreihe,so wie sie stattgefunden hat,zufrieden (30%)

Auswahl aus den Schülerantworten:

„Ohne Vorlage (Lehrer)",„Vielleicht etwas weniger,dafür ausführlicher",„Ich bin ziemlich zufrieden mit den Leistungen von Lehrer und Schülern.",„Ich hätte mich mit dem Vier-Farben-Problem gerne noch ausführlicher beschäftigt.",„Würde gerne Algorithmen völlig selbständig entwickeln."

11)Meinen Sie,dass Themen dieser Unterrichtsreihe auch noch nach dem Abitur für Sie wichtig sind (z.B. im Hinblick auf Ihr künftiges Studienfach/ künftige Ausbildungswahl)?

Antworten:

85% der Schüler sind der Meinung,dass die Themen auch noch nach dem Abitur für sie wichtig sind.15% machen keine Angaben oder geben nicht ernsthaft gemeinte Spaßantworten.

Auswahl aus den Schülerantworten:

„Sicher",„Da ich an einem Beruf in der IT-Branche interessiert bin,denke ich schon.",„Ja",„Bestimmt",„Ich will Gärtner werden,unter Umständen?"

12)Welche Themen hätten zusätzlich in dieser Unterrichtsreihe behandelt werden sollen?

Antworten:

Die Antworten auf diese Frage variieren sehr stark.Die folgende Auswahl beschreibt das Spektrum einigermaßen repräsentativ:

Auswahl aus den Schülerantworten:

„Laden und Speichern eines Graphen auf der Festplatte",„Quelltext der Methoden zur Gestaltung der graphischen Oberfläche",„Probleme,die im normalen Mathematikunterricht vorkommen,lösen.",„Beweis zum Vier-Farben-Problem",„Quelltexterstellung mit einer anderen Programmiersprache", „Wahrscheinlichkeitsrechnung mit Graphen",„Graphen und Computervernetzung", „Internet und Graphen",„Keine Idee","Betriebssystem Windows",„Netzwerke", „Zu endlichem Automat mehr theoretische Informatik",„Sortierverfahren mit Bäumen"

13)Welche Themen der Unterrichtsreihe waren

schwer:

Auswahl aus den Schülerantworten:

„Später immer schwieriger", „Backtracking", „Nichts", „Listen", „Maximaler Fluss",„Endlicher Automat", „Tiefensuche", „Eulerproblem", „Netzzeitplan", „Beweise", „Theoretische Informatik", „Theorie"

leicht:

Auswahl aus den Schülerantworten:

„Die ersten",„Von Listen abgeleitete Graphen",„Eulerlinien", „Netzzeitplan",„Tiefensuche",„Hamilton-Problem",„Alle Pfade",„Eulerformel"

interessant:

Auswahl aus den Schülerantworten:

„Fast alles,da an Hand der graphischen Oberfläche mit dem Demomodus alles nachvollzogen werden konnte.",„Listen",„Graphen",„Tiefensuche",„Hamilton-Kreise",„Endlicher Automat",„Optimierung mit Graphen"

uninteressant:

Auswahl aus den Schülerantworten:

„Das Abschreiben von Aufgaben",„Aufgaben mit viel und unüberschaubarem Quelltext,der schwer zu verstehen ist (Motivationsverlust).",„Nichts", „Theorie",„Beweise"

wichtig:

Auswahl aus den Schülerantworten:

„"Alles",„Für sich betrachtet hat alles eine spezielle Bedeutung.", „Algorithmen,„Objektorientiertes Programmieren",„Backtracking",„Optimierungsprobleme",„Hamilton-und Eulerlinien",„Endlicher Automat",„Maximaler Fluss",„Färbbarkeit eines Graphen"

unwichtig:

Auswahl aus den Schülerantworten:

„Keines",„Wüßte nicht,was wichtiger sein könnte.",„Nichts",„Beweise", „Theorie",„Theoretische Informatik",„Relationen in Graphen"

14)Welche Themen haben Sie nicht richtig verstanden?

Antworten:

Ca. 30% der Schüler lassen diese Frage offen,50% beantworten die Frage mit einem Strich.Der Rest der Antworten verteilt sich auf die verschiedensten Themen.

Auswahl aus den Schülerantworten:

„Listen",„Eigentlich kannte ich alle.",„Backtrackingalgorithmus",„Maximaler Netzfluss",„Planarität von Graphen",„Netzzeitplan",„Programmierung von Hamiltonlinien",„Kreise",„Endlicher Automat",„Relationen",„Theorie", „Theoretische Informatik",„Maximaler Fluss"

15)Läßt sich die Datenstruktur des Graphen Ihrer Meinung nach verbessern.Wenn ja wie?

Antworten:

Fast alle Schüler geben keine Antwort oder als Antwort,dass die Datenstruktur nicht mehr weiter zu verbessern ist.

Eine der wenigen anders lautenden Schülerantworten:

„Als Ergänzung:Prozedur zur selbständigen Erstellung bestimmter Graphen"

16)War die Unterrichtsreihe zum Thema Graphen und Objekte zu lang/zu kurz/ angemessen?

Antworten:

72% der Schüler finden die Länge der Unterrichtsreihe angemessen,13% zu kurz und 15% zu lang.

17)Waren die Klausuren zum Thema Objekte/Graphen angemessen/zu schwer/leicht?

Antworten:

55% geben an,dass die Klausuren angemessen waren,27% bemerken,dass sie die Klausuren nicht mitgeschrieben haben,weil sie Informatik nicht als schriftliches Fach gewählt hatten,10% antworten mit schwer und 8% beantworten die Frage nicht.

Auswahl aus den Schülerantworten:

„angemessen“,„Ich habe keine Klausur mitgeschrieben.“,„zu schwer“

18)War der Anteil Theorie und praktische Arbeiten am Computer angemessen?

Antworten:

45% halten die Anteile für angemessen,32% hätten sich noch mehr praktisches Arbeiten gewünscht,23% beantworten die Frage nicht.

Auswahl aus den Schülerantworten:

„Ja“,„Etwas mehr praktische Arbeit,bei der unter Hilfestellung des Lehrers Probleme selbst gelöst werden.“,„Angemessen“

19)Worin liegt Ihrer Meinung die Bedeutung a)von Objekten b)von Graphen für die Fächer Informatik und Mathematik?

a)

Auswahl aus den Schülerantworten:

„Vereinfachung,Weiterentwicklung und Übersichtlichkeit“,„Sehr wichtig und typisch“,„Vereinfachung der Entwicklung von Algorithmen“,„Bereitstellung von graphischen Oberflächen“,„Übernahme von vorgegebenem Quellcode (Methoden)“,„Wiederverwendung von Quellcode“,„Erstellen größerer Programme“,„Nachträgliches Ändern von Programmen“

b)

Auswahl aus den Schülerantworten:

„Man macht mit ihrer Hilfe die Probleme sichtbar und findet auf Grund dessen schneller eine Lösung.“,„Die mathematischen Beziehungen können abgebildet werden.“,„Für den (normalen) Mathematikunterricht haben Graphen keine Bedeutung.“,„Anschaulicher Algorithmus“,„Mathematik ohne Formeln“

20)Sollten Eigenschaften/Algorithmen von/auf Graphen auch im Mathematikunterricht der Schule (evtl. statt anderer Inhalte) behandelt werden (Graphentheorie ist eigentlich ein Teilgebiet der Mathematik)?Wenn ja,worin sehen Sie den besonderen Vorteil?

Antworten:

78% beantworten diese Frage mit ja,18% mit nein,4% geben keine Antwort.

Auswahl aus den Schülerantworten:

„Ja,weil dieses Thema von wirtschaftlicher Bedeutung ist.“,„Ein Vorteil wäre,dass man die Eigenschaften besser versteht und behält.“,„Nein,weil eine Behandlung von Graphentheorie im Informatikunterricht mehr Zeit und Möglichkeit für die Programmierung bietet.“,„Weil es anschaulicher ist.“, „interessanter“

21)Ist die Lösung von Graphenproblemen eine andere Art von Mathematik als die,die Sie im „normalen“ Mathematikunterricht (Analysis,Algebra,Geometrie usw.) kennengelernt haben?Wenn ja,wo liegen die Unterschiede?

Antworten:

Die Frage wurde von ca. 60% beantwortet.90% davon beantworten die Frage mit den Stichworten „Programmieren“,„Algorithmen“ und „Weniger Theorie“,„Mehr eigenes Handeln“,„Aufgaben können mehr selbständig gelöst werden.“

Auswahl aus den Schülerantworten:

„Der Unterschied liegt darin,dass die Probleme in Mathematik sehr theoretisch sind.“,„Ich denke,sie sind schon unerschiedlich,nämlich einfacher.“,„Ja,die Algorithmen haben nichts mit Formeln zu tun.“, „Anschaulicher“,„Weniger abstrakt“,„Einfachere Aufgaben“

22)Ist die Beschäftigung mit Graphentheorie interessanter als die Ihnen bekannten Themen (Analysis,Algebra,Geometrie usw.) der Mathematik?

Antworten:

73% halten die Beschäftigung mit Graphen für interessanter,5% für weniger interessant und 13% für gleich.Der Rest gibt keine Antwort.

Auswahl aus den Schülerantworten:

„Teils“,“„Nein“,„Beides gleichwertig“,„Ja,da die Arbeit am Computer Spaß macht.“,„Ja,da mit logischem Denken viel erreicht wird (ähnlich Geometrie).“,„Interessanter,da anschaulicher“,„Lösungen können besser erkannt werden.“,„Sehr viel interessanter“

23)Wurde im Mathematikunterricht während Ihrer Schulzeit irgendwann das Thema Graphen oder Graphenalgorithmen behandelt bzw. Graphen als Hilfsmittel benutzt?Wenn ja,wann und welches Thema?

Antworten:

100% der Antworten verneinen diese Frage.

24)Sonstige Kommentare/Anregungen/Kritik:

Antworten:

Kritik richtet sich vor allen daran,dass andere Themen der Informatik wie Internet,Umgang mit Hardware,detailierte Betriebssystemkentnisse usw. zu kurz gekommen sind.Ansonsten finden sich Antworten wie „Die Unterrichtsreihe hat Spaß gemacht.“,„Besonders interessant fand ich,den Ablauf der Algorithmen im Demomodus beobachten zu können.“,„Die Beschäftigung mit Graphen ist eine gute Ergänzung zum normalen Mathematikunterricht.“

Beispiel einer Facharbeit

Möglichkeiten der objektorientierten Darstellung von Graphen, Suchbäume in Graphen und die zugehörigen Algorithmen (Tiefen- und Breitensuche)

Facharbeit im
Fach Informatik, GK If3
Betreuungslehrer: Herr Eidmann

vorgelegt von

Csaba Letay, 12.2
Gymnasium Sedanstraße

Wuppertal 2001

Erklärung

Ich versichere, dass ich die Arbeit selbständig verfasst und keine anderen Hilfsmittel als die angegebenen benutzt habe. Die Stellen der Facharbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Das Gleiche gilt auch für beigegebene Zeichnungen, Kartenskizzen und Darstellungen.

____________________ , 01.04.2001

Inhaltsverzeichnis

1. Einleitung
   1.1 Fragestellung
2. Hauptteil
   2.1 Was sind Graphen und wofür sind sie gut?
   2.2 Datenstrukturen von Graphen
   2.3 Objektorientierte Darstellung von Graphen
   2.4 Bäume
      2.4.1 Binärbäume
      2.4.2 Suchbäume
         2.4.2.1 Funktionen von Suchbäumen
   2.5 Graphendurchlauf
      2.5.1 Tiefensuche
      2.5.2 Breitensuche
3. Schlußteil / Resümee

1. Einleitung

Der Schwerpunkt dieser Arbeit soll auf den Graphenalgorithmen des Graphendurchlaufs und auf den verschiedenen binären Bäumen liegen. Außerdem verzichte möglichst auf die Abschrift von Quelltext, weil sie zum Verständnis relativ wenig beiträgt und ich so Platz einsparen kann, was die Seitenanzahl in einem akzeptablem Maß hält. Der Aspekt der Darstellung von Graphen als Objekte wurde nicht vernachlässigt, jedoch komprimierend zusammengefaßt und auf die Hauptpunkte reduziert.

1.1 Fragestellung
Welche Möglichkeiten gibt es Graphen darzustellen, welche Datenstrukturen werden angewendet? Wie sind die Unterschiede zwischen Tiefen- und Breitensuche? Was ist ein binärer Baum? Was unterscheidet ihn vom binären Suchbaum? Welche Funktionen besitzt ein binärer Suchbaum?

2.Hauptteil

2.1 Was sind Graphen und wofür sind sie gut?

Ein Graph besteht aus sogenannten Knoten (die Punkte des Graphen) und Kanten (die Linien des Graphen), die die einzelnen Knoten verbinden. Kanten können gerichtet oder ungerichtet sein. Durch Aneinanderreihung von Kanten entstehen Pfade.
Graphen sind geeignet um verschiedene Sachverhalte darzustellen und mit entsprechenden Graphenalgorithmen Problemstellungen zu lösen. Anwendungsgebiete sind z.B. Programme, die Routen / Strecken berechnen und darstellen sollen. Graphen können verschiedene Aufgaben lösen. Eine für einen Graphen optimale Aufgabe wäre z.B., wenn jemand wissen möchte, „wie er all seine Kunden mit einer kürzestmöglichen Rundreise besuchen will" (Informatik für die Sek. Stufe 2, Rüdiger Baumann). Auch werden Graphen in Architektur-Programmen für die Darstellung von Räumen benutzt. Des weiteren findet man in den meisten neuen 3D PC-Spielen als Grundbaustein der verschiedenen 3D Welten ganz normale Graphen. In diesem Fall werden die Knoten als „nodes" und die Kanten als „edges" bezeichnet.

2.2 Datenstrukturen von Graphen

Es gibt verschiedene Formen von Graphen, u.a. Listen und Bäume. Jedoch können alle mit der im Unterricht besprochenen Struktur beschrieben werden. Ein Graph, wie wir ihn im Unterricht besprochen / entwickelt haben, hat eine sogenannte Knotenliste, in der die Knoten gespeichert sind. Jeder Knoten besitzt zusätzlich eine ausgehende und eingehende Kantenliste, in der auf die von ihm wegführenden bzw. die zu ihm hinführenden Kanten gezeigt wird. Diese Kanten sind in der allgemeinen Kantenliste des Graphen gespeichert. Bei diesen Listen handelt es sich um Objekte von den Typen TKantenliste und TKnotenliste. Die Knoten haben zusätzlich noch die Property „besucht" vom Typ Boolean, um bei den verschiedenen Algorithmen zwischen den bearbeiteten und nicht bearbeiteten Knoten zu unterscheiden.
In der Informatik werden verschiedene Datenstrukturen für Graphen verwendet.
Zum Beispiel wird in vielen Texten die sogenannte „Adjazenzmatrix" angesprochen. Sie ist ein statisches Array vom Typ ARRAY [1..knotenzahl, 1..knotenzahl] OF 0..1.
Wegführende gerichtete Kanten werden in der Matrix mit einer 1 bezeichnet. Wenn z.B. eine Kante von Knoten 2 nach 3 verläuft, dann steht im Array an der Stelle [2,3] eine 1, andernfalls eine 0.
Mit folgendem kleinen von mir erdachten Quelltext könnte man z.B. alle Kanten und die dazu gehörigen Anfangs und Endknoten ausgeben:
Gegeben ist die Anzahl der Knoten, matrix vom Typ ARRAY[1..n,1..n], x und y als Integer-Variablen.

```
for x:= 1 to anzahlknoten do
for y:= 1 to anzahlknoten do
Begin
If matrix[x,y] = 1 then
Writeln(`Es verläuft eine Kante von Knoten `, x, `nach Knoten `, y);
End;
```

Da diese Datenstruktur viel zu verschwenderisch im Umgang mit Speichereinheiten ist (es gibt selten in einem Graphen so viele Kanten wie Anzahl-Knoten²), wird eher die Adjazenzliste verwendet. Diese Struktur ähnelt sehr der im Unterricht entwickelten. Hier besitzt auch jeder Knoten eine Liste mit allen von ihm wegführenden Kanten.
(Text sinngemäß aus „Informatik für die Sek. Stufe 2", Rüdiger Baumann)

2.3 Objektorientierte Darstellung von Graphen

Das ein Graph Knoten und Kanten besitzt wissen wir bereits, jedoch gibt es noch weitere Aspekte über Knoten und Kanten in objektorientierter Hinsicht. Knoten können Quell- und Zielknoten sein und haben meistens eine Bezeichnung (Caption).

Die Kanten eines Graphen können sowohl auslaufen als auch einlaufend sein.Kanten verlaufen vom Qell- zum Zielknoten, sie können gerichtet und ungerichtet sein. Ungerichtete Kanten sind aus- und einlaufend. Kanten sind nicht nur die reine Beziehung zwischen Nachbarknoten, sie können auch Inhalte tragen, also gewichtet sein. Mehrere Kanten die durch Knoten verbunden sind nennt man wie weiter oben erwähnt Pfade. Pfade können bestimmte Eigenschaften aufweisen: Sie können einen in sich geschlossenen Kreis darstellen (zyklische Pfade). In einem Graphen mit gerichteten Kanten weicht meistens die Anzahl der Quellknoten von der Anzahl der Zielknoten ab. In einem ungerichteten Graphen ist sie gleich.
Komponenten eines Graphen werden voneinander unabhängige, unerreichbare Teile genannt. Dabei kann es sich um nur einen Knoten oder gleich ganze Pfade handeln. Bei der objektorientierten Darstellungen von Graphen werden alle Knoten und Kanten fortlaufend durchnumeriert und in Listen als Objekte gespeichert. Diese Objekte sind abstrakt und können sich in ihren Eigenschaften von Programmierung zu Programmierung unterscheiden im Hinblick auf den Zugang zu den Knoten und Kanten.

(aus: Listen, Bäume und Graphen als Objekte)

2.4 Bäume

2.4.1 Binärbäume

Der binäre Baum ist eine spezielle Form des Graphen. Jeder Knoten besitzt genau zwei Folgekanten und –knoten, so dass sich die Anzahl der Knoten exponentiell vergrößert. Er ist einer der am häufigsten verwendeten Baumstrukturen. Der Ursprungsknoten dieses Graphen wird als Wurzel bezeichnet. Ist dieser Knoten leer, so ist der gesamte Baum leer. Knoten, die keinen Nachfolger haben , werden als äußere Knoten bezeichnet. „Diese Knoten, die sich meistens auf der untersten Ebene eines Baumes befinden, müssen leer sein, um der Definiton nicht zu widersprechen". Wegen ihrer Eigenarten werden Sie oft eckig dargstellt, wie auf Abbildung 1 zu sehen ist, damit man sie von den inneren Knoten unterscheiden kann. Inneren Knoten ist ein Wert zugewiesen und jeder von ihnen hat zwei Nachfolgeknoten.

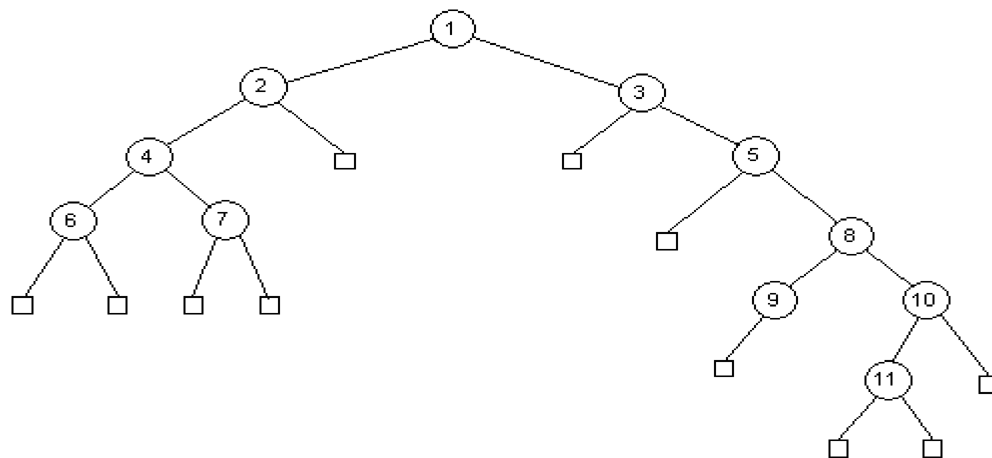


Abb. 1

(Quelle Abbildung und Text (sinngemäß): Internetseite Informatik-Treff der Bez.reg. D.-dorf)

2.4.2 Suchbäume
Ein Suchbaum ist im Prinzip ein Binärbaum, der geordnet ist. Folgende zwei Eigenschaften muss ein Binärbaum besitzen, damit er als Suchbaum gilt:

1. „Jeder Knoteninhalt muss eine Art Schlüssel haben, für den eine Ordnung definiert ist. Die Elemente eines Suchbaumes müssen also vergleichbar sein"(Informatik- Treff, Daniel Bigus u. Torge Cepus, Freiherr-vom-Stein-Gymnasium Oberhausen)

2. „Umsetzung der Ordnung durch die spezielle Struktur des Baumes: Für jeden Knoten, der kein Blatt ist, muss gelten, dass alle Knoten in seinem linken Teilbaum kleiner und alle im rechten Teilbaum größer sind als er selbst" (Informatik- Treff, Daniel Bigus u. Torge Cepus, Freiherr-vom-Stein-Gymnasium Oberhausen)

Ein Suchbaum ist immer sortiert. Die Ordnung muss immer beibehalten werden daher bei allen Operationen und Zugriffen auf die Inhalte oder Struktur des Baumes. Wenn man ein neues Element bzw. Blatt in den Baum einfügt, dann geschieht dies direkt an der passenden Stelle. Um einen neuen Suchbaum zu erstellen, wird zuerst ein leerer Baum erzeugt, in den dann die Elemente bzw. Blätter in geordneter Reihenfolge eingefügt werden.

2.4.2.1 Funktionen von Suchbäumen

Ein Suchbaum hat eine Reihe von verschiedenen Funktionen bzw.Methoden.

Im einzelnen sind diese: Suchen, Einfügen und Löschen

Suchen

Aufgrund seiner Struktur ist es einfach den Suchbaum nach einem bestimmten Wert durchsuchen zu lassen. Das Suchen fängt bei der Wurzel an und arbeitet sich in die Teilbäume vor. Die Knoten werden auf ihre Werte überprüft. Wenn der gesuchte Wert größer ist als der jeweilige Knoten, wird auf der rechten Seite weitergesucht, weil alle Werte in allen Teilbäumen, die links von diesem Knoten sind, kleiner sind als der Wert des aktuellen Knotens. Dies ist der geordneten Struktur des Suchbaumes zu verdanken. Wenn der Wert kleiner ist als der Wert des aktuellen Knotens, dann wird dementsprechend links von diesem Knoten weitergesucht. Durch diese Methode wird die Durchsuchung aller Knoten vermieden. Es können von vornherein komplette Teilbäume weggelassen werden.

Die Suche ist dann abgeschlossen, wenn entweder der gesuchte Wert bzw. gesuchte Knoten gefunden ist oder wenn die Suche an einen (leeren) äußeren Knoten stößt, von dem aus keine weiteren Kanten ausgehen.

Suchaufwand

Ein wichtiger Faktor ist der Suchaufwand. Im binären Suchbaum hängt er hauptsächlich vom Aufbau des Baumes ab, der wiederum sehr von der Reihenfolge der Einfügung der Knoten abhängt. Dieses Phänomen läßt sich anhand von Abbildung 2 erklären

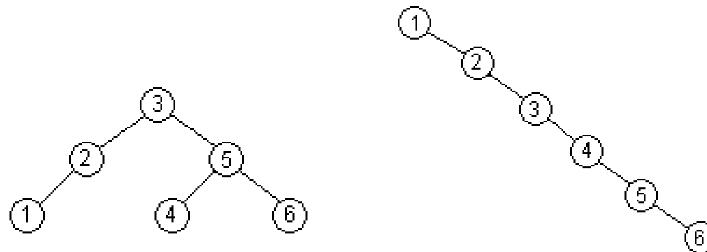


Abb. 2

Beide Bäume haben zwar den selben Inhalt, jedoch ist ihr Aufbau sehr verschieden. Damit verbunden variiert auch der Aufwand einen bestimmten Knoten zu finden. Wenn man z.B. Knoten 6 finden will, muß im linken Beispiel die Suchfunktion 3 knoten überprüfen (3, 5, 6) . Im rechten Baum müssen jedoch alle sechs Knoten (1, 2, 3, 4, 5, 6) geprüft werden. Der rechte Baum wird als degenerierter oder entarteter Suchbaum bezeichnet. Im Prinzip entspricht dieser ungünstige Fall einer Liste der Elemente, daher ist der Suchaufwand linear. Im ausgeglichenen Baum, der als idealer Fall bezeichnet wird, ist der Aufwand logarithmisch.
Der durchschnittliche Suchaufwand in einem Suchbaum wird durch die mittlere Weglänge bestimmt. Man kann sie errechnen, indem man die Weglängen zwischen den Knoten des Baumes addiert und dann durch die Anzahl der Knoten dividiert. So beträgt der durchschnittliche Suchaufwand im linken Baum (0+1+1+2+2+2):6 = 1,3, wobei 1 jeweils dem Weg von Knoten 3 zu Knoten 2 und Knoten 5 entspricht und 2 dem Weg zwischen Knoten 3 und Knoten 6, 4 oder 1. Im rechten Baum ist der Aufwand zweimal so hoch (0+1+2+3+4+5):6 = 2,5 .

Einfügen

Dem Einfügen eines neuen Elementes in den Suchbaum geht immer ein Suchvorgang nach dem einzufügenden Element voraus, wobei die Position zum Einfügen festgelegt wird.Wenn das einzufügende Element bereits im Baum vorhanden ist, wird es entweder überhaupt nicht oder, je nach Programmierung,im rechten oder linken Teilbaum des vorhandenen Knotens angehängt. Wenn das Element in der Suche nicht gefunden wird, wird das neue Element als neuer Knoten an der Stelle angehängt, an der die Suche zu Ende war.

Löschen

Auch vor dem Löschen findet ein Suchvorgang nach dem zu löschenden Element statt. Wird das Element nicht gefunden, wird der Löschvorgang abgebrochen. Verläuft die Suche jedoch positiv, muss das gefundene Element aus dem Baum gelöscht werden.
Aufgrund der Notwendigkeit, dass die Ordnung beibehalten wird, müssen drei Fälle des Löschens unterschieden werden:
1.Löschen eines Blattes (äußerer Knoten): Der gefundene Knoten wird einfach entfernt.
2.Löschen eines Knotens mit einem nicht-leeren Teilbaum (innerer Knoten): Der gefundene Knoten wird entfernt, und die Wurzel, also der Nachfolgeknoten, des nicht-leeren Teilbaumes rückt an die

Stelle des gelöschten Elements. Mit diesem Vorgang rücken auch alle nachfolgenden Knoten bzw. der gesamte Teilbaum eine Ebene höher.
3.Löschen eines Knotens mit zwei nicht-leeren Teilbäumen (innerer Knoten): Der gefundene Knoten muss durch den größten Knotenwert seines linken Teilbaums oder durch den kleinsten Knotenwert seines rechten Teilbaums (die Blätter sind und dabei gelöscht werden), ersetzt werden.

(Quelle Abbildungen, Text (sinngemäß): Internetseite Informatik-Treff der Bez.reg. D.-dorf)

2.5 Graphendurchlauf

Mit Graphendurchlauf ist die systematische Bearbeitung aller Knoten und Kanten eines

Graphen gemeint. Dabei werden die Knoten in drei verschiedene Zustände unterteilt:

- Unbesuchte Knoten
- Anwärterknoten
- Besuchte Knoten

Unbesuchte Knoten sind noch nicht bearbeitet worden und auch noch nicht für die Bearbeitung vorgesehen. Anwärterknoten werden die Knoten bezeichnet, die in eine Liste oder Menge aufgenommen werden, die Schritt für Schritt abgearbeitet wird. Besuchte Knoten wurden bereits bearbeitet. Beim Durchlauf sind zunächst alle Knoten unbesucht, es wird ein unbesuchter Knoten zum Anwärterknoten ausgewählt (Startknoten), bearbeitet und dann besucht gesetzt. Alle von ihm aus direkt erreichbaren Knoten (Nachbarknoten) werden in die Anwärterliste aufgenommen. Wenn ein Anwärterknoten besucht gesetzt wurde, werden seine Nachbarknoten in die Anwärterliste aufgenommen. Dieser Vorgang wiederholt sich immer wieder, bis die Anwärterliste keine Elemente mehr hat.

Der zugehörige Algorithmus sieht so aus (wörtlich aus „Informatik für die Sek.Stufe 2"):

„Anwärtermenge := {Startknoten}, kein Knoten als abgearbeitet markiert

SOLANGE Anwärtermenge nicht leer WIEDERHOLE

Nimm Knoten p aus Anwärtermenge und markiere ihn als abgearbeitet (*)

FÜR ALLE Nachbarn q von p WIEDERHOLE

WENN NICHT(q abgearbeitet ODER q in Anwärtermenge) DANN

Füge q in Anwärtermenge ein (**)

ENDE WENN

ENDE-WIEDERHOLE

ENDE WIEDERHOLE"

Diese allgemeine Prozedur kann man auf zwei verschiedene Weisen modifizieren.

Wenn wir die Anwärterliste als Stapel bearbeiten und werden neue Knoten immer mit höherer Priorität behandelt als alte, dann erhalten wir einen Tiefendurchlauf bzw. Tiefensuche. Wenn wir sie als Schlange verwalten und die alten Elemente der Anwärterliste den neuen vorgezogen, handelt es sich um eine Breitensuche. Dies ist im Algorithmus bei (**), also beim Anfügen an die Liste bzw. bei (*), beim Auslesen aus der Liste festzulegen.

(Quelle: Text (sinngemäß), Quellcode, Algorithmen (wörtlich) aus (Informatik für die Sek. Stufe 2, Rüdiger Baumann)

2.5.1 Tiefensuche

Beim Tiefendurchlauf wird nachdem der Startknoten besucht wurde ein beliebiger Nachbarknoten ausgewählt. Von diesem wird nach der Bearbeitung sofort der erste Nachbarknoten ausgewählt, denn dieser wurde an den Anfang der Anwärterliste gesetzt. So erreicht man ein schnelles Vordringen in den Graphen. Wenn ein Knoten keine Nachbarknoten mehr hat, dann wird eine backtracking- Prozedur ausgeführt, wie wir sie auch im Unterricht besprochen haben- So geht der Algorithmus einen Schritt zurück und setzt seine Arbeit an dem zuvor bearbeiteten Knoten fort. Beim Tiefendurchlauf entsteht ein Tiefensuchbaum, der eigentlich der gleiche Graph ist wie vorher, jedoch umstrukturiert und sortiert worden ist.

(Quelle: Informatik für die Sek. Stufe 2)

2.5.2 Breitensuche

Bei der Breitensuche wird die nähere Umgebung des Graphen zuerst bearbeitet. Der Vorgang arbeitet also immer Ebene für Ebene ab. Denn zuerst werden alle dem Graphen am nächsten stehende Knoten besucht und danach die der Liste neu hinzugefügten. Es handelt sich bei der Liste um eine Schlange, also wird am Anfang ausgelesen und am Ende angefügt. Zusammenfassend kann man sagen: Beim Tiefendurchlauf wird in der Hauptreihenfolge gesucht, also jeweils der in der Anwärterliste neuste Knoten bearbeitet. Beim Breitendurchlauf wird in Ebenen abgesucht, also jeweils der in der Anwärterliste älteste Knoten inspiziert. Dies ist auch der einzige Unterschied, da beide Verfahren den gleichen Suchbaum erzeugen.

(Quelle: Informatik für die Sek. Stufe 2)

3. Schlußteil / Resümee

Während der Bearbeitung dieser Arbeit ist mir klar geworden, dass Graphen nicht nur in der reinen theoretischen Informatik ihre Verwendung haben, sondern dass sie auch vor allem in der praktischen Anwendung der Informatik eine wichtige Rolle spielen. Auch fachfremde Personen benutzten Graphen und Graphenalgorithmen für die Lösung von verschiedensten Aufgaben in den unterschiedlichsten Anwendungen, wie z.B. in Routen-Programmen.
Außerdem wurde deutlich, dass es keine einheitlichen Normen gibt, um Graphen darzustellen.
Sie können sowohl mit Matrizen in Form von statischen Arrays (z.B. die Adjazenzmatrix) als auch mit Listen, die auch wiederum unterschiedlich strukturiert sein können, objektorientiert dargestellt werden. Auch die verschiedenen Bezeichnungen innerhalb der Graphentheorie weichen von Literatur zu Literatur voneinander ab. Des weiteren wurde deutlich, dass man für Graphen viele Algorithmen entwickeln kann, wie z.B. die Tiefen- und Breitensuche, die auch wiederum verschiedentlich realisiert werden können. Man kann z.B. den Graphendurchlauf rekursiv und nicht-rekursiv programmieren. Natürlich müssen sich die Algorithmen an die Beschaffenheit der Graphen anpassen.

Literaturverzeichnis

1. Läuchli, Peter
   „Algorithmische Graphentheorie"
   Birkhäuser Verlag

2. Schaerer, Daniel
   „Listen, Bäume und Graphen als Objekte"
   Springer-Verlag

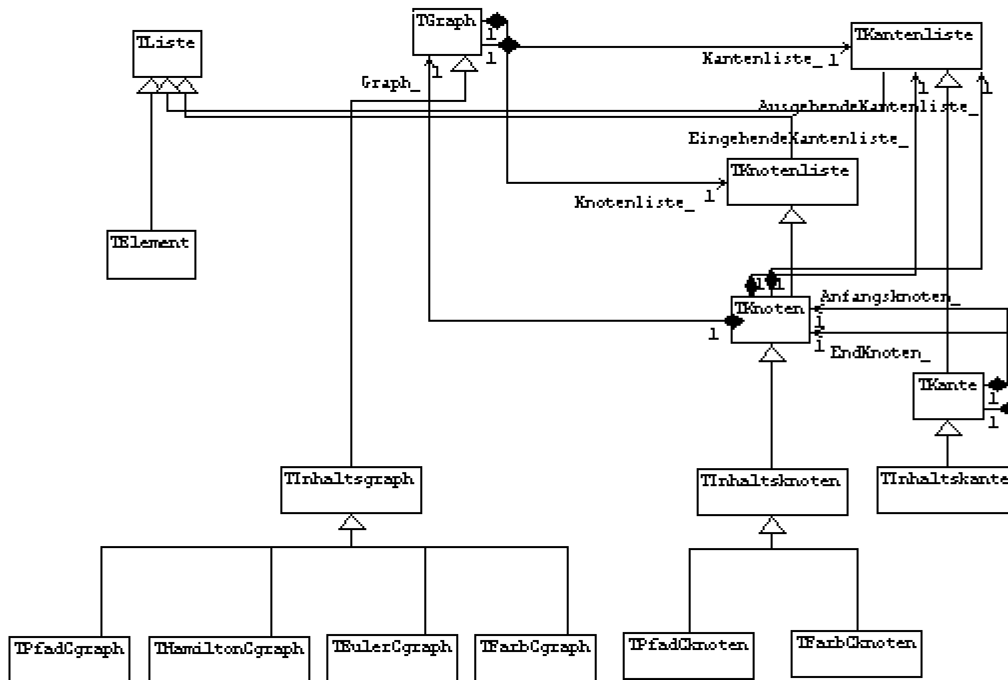
3. Baumann, Rüdiger
   „Informatik für die Sek.Stufe 2"

Internet-Seiten:
1. Informatik-Treff
2. http://www.bezreg-duesseldorf.nrw.de/schule/informatik/index.htm

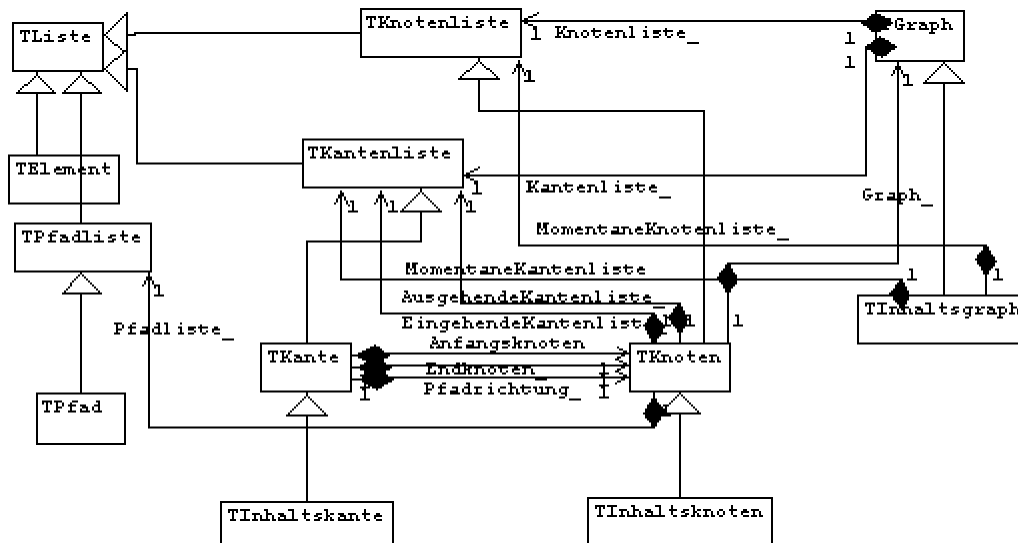
UML-Diagramme zu den Projekten CAK,DWK und EWK

Aus Übersichtlichkeitsgründen beschränkt sich die Darstellung auf die grundlegenden Objektrelationen.

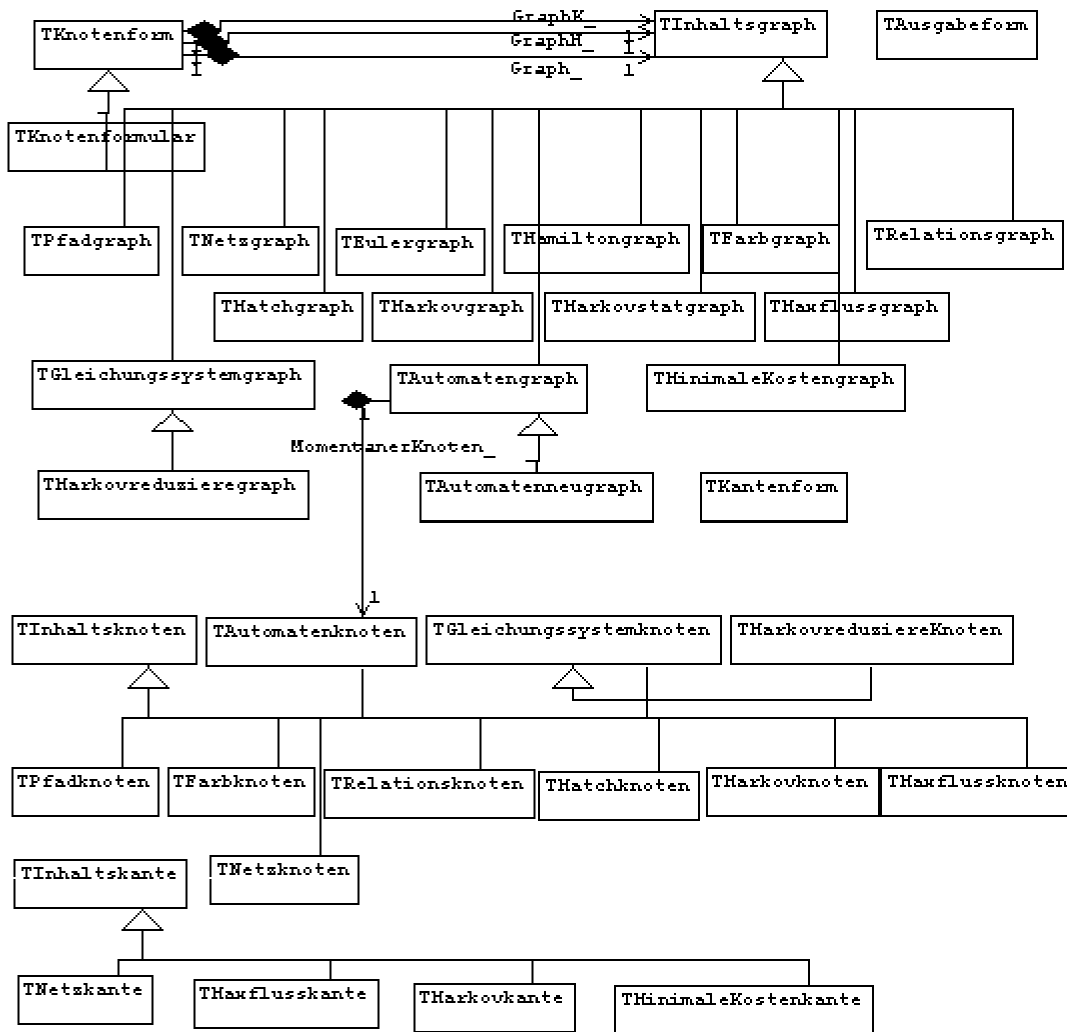
UML-Diagramm Objektrelationen Projekt CAK:



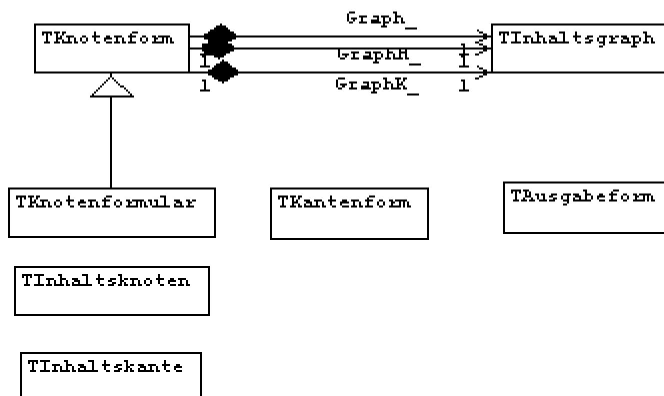
UML-Diagramm Objektrelationen DWK und EWK,gemeinsame Objekte,1.Teil:



UML-Diagramm Objektrelationen DWK,2.Teil:



UML-Diagramm Objektrelationen EWK,2.Teil:



Update

Das Verzeichnis Update enthält eine verbesserte Version des Programms Knotengraph der Konzeption DWK.

Folgende Verbesserungen sind enthalten:

Menü Eigenschaften/Kreise bestimmt jetzt auch zusätzlich Kantenzüge (Walks) fester Länge.

Menü Eigenschaften/Brücken bestimmt jetzt auch zusätzlich Artikulationspunkte.

Menü Bild/Gerichtete Kanten ersetzt in einem Graphen alle ungerichteten Kanten (außer Schlingen) durch zwei entgegengesetzt gerichtete Kanten zwischen den entsprechenden Knoten.Ungerichtete Schlingen werden nur mit einer Richtung versehen.

Menü Bild/Knoten numerieren numeriert alle Knoten in der Reihenfolge ihres Einfügens i in den Graph mittels der neuen Bezeichnung v_i durch.

Menü Bild/Kanten numerieren numeriert alle Kanten in der Reihenfolge ihres Einfügens i in den Graph mittels der neuen Bezeichnung e_i durch.

Menü Anwendungen/Verbindungszahl nach Menger bestimmt jetzt auch zusätzlich die Maximalzahl Kanten- oder Knotenverschiedener Wege von einem Quellen- zu einem Senkenknoten im gerichteten Graph gemäß dem Satz von Menger.

Der Algorithmus arbeitet nach auf der Grundlage des Bestimmung eines 1/0-Maximalflusses (mit den Schranken 1) nach dem Algorithmus von Ford-Fulkerson (unter Verwendung des Objekts TMaxflussgraph sowie dessen Methoden)und nutzt die weitgehende Äquivalenz der Sätze von Menger und Ford-Fulkerson aus.So entstehen sofort Kantenverschiedene Wege.Knotenverschiedene Wege ergeben sich durch Aufsplitten aller Knoten des Graphen in den ursprünglichen Knoten und einen mit * gekennzeichneten Knoten.Der mit * bezeichnete Knoten erhält die ausgehende Kantenliste des ursprünglichen Knoten,der die eingehende Kantenliste behält.Zwischen beiden Knoten wird eine gerichtete Kante vom Wert 1 eingefügt.Faßt man beide Knoten bei der Pfadausgabe im Ausgabefenster wieder als einen Knoten auf,erhält man alle Knotenverschiedenen Wege.

Ungerichtete Graphen können mit dem Menü Bild/Gerichtete Graphen in gerichtete Graphen verwandelt werden,wobei eventuell Quellen- und Senkenknoten noch manuell durch Löschung geeigneter Kanten(-richtungen) erzeugt werden müssen.(Auch im gerichteten Graph ohne Quelle oder Senke können oft durch Löschen entsprechender Kanten Quellen- und Senkenknoten hergestellt werden,ohne den Fluss zu ändern:Lit 55,S. 221)

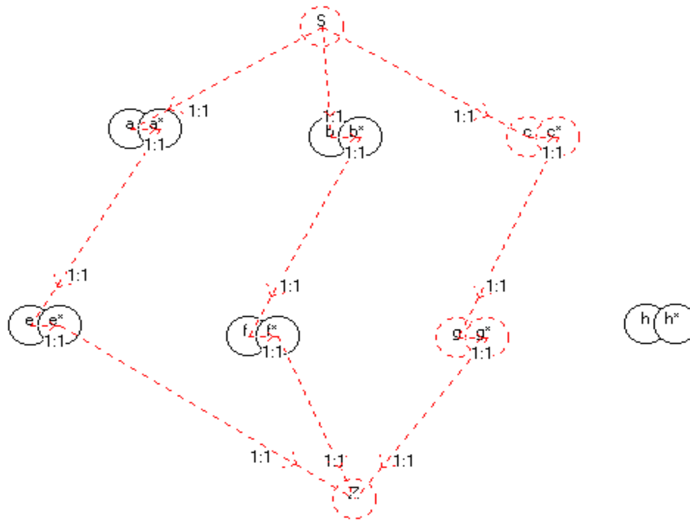
Der in den vorigen Absätzen beschriebene Algorithmus zur Bestimmung der Verbindungszahl nach Menger ist eine Möglichkeit,auf der Grundlage des Maximalflussalgoritmus didaktisch-methodisch den Satz von Menger im Unterricht zu besprechen.Der minimale Schnitt im Satz von Ford-Fulkerson entspricht dabei der trennenden Kanten-bzw. Knotenmengen im Satz von Menger und kann auch so erkannt werden.(Fluss gleich Schranke:vgl Kapitel CIX,Aufgabe C IX.3,S. 294)

Dabei kann man sich auch darauf beschränken manuell 1/0-Graphen mit den Kantenschranken 1 zu erzeugen (sowie evtl. Knoten zu splitten) und ohne einen neuen Algorithmus zu programmieren und mit dem Menü Anwendungen/Maximaler Netzfluss die Verbindungszahl bestimmen.

Als Spezialfall des Satzes von Menger erweist sich der Satz von König (Kapitel C X,Aufgabe CX.4,S. 303,Satz CX.3)

Legt man den Graph G050.gra bzw. G053.gra (S.297 ff.,S304 ff.) zugrunde und wendet darauf den Algorithmus Anwendungen/Verbindungszahl nach Menger an

mit der Option der Maximalzahl Knotenverschiedener Wege,ergibt sich folgendes Ergebnis:



Graph G053.gra nach Anwendung des Algorithmus Verbindungszahl nach Menger

Man erkennt,dass die Knoten a,b und c gerade eine minimale trennende Knotenmenge bilden.Die maximale Anzahl der Knotenverschiedenen Wege ist ebenfalls 3 und entspricht der Kantenzahl des maximalen Matchings: a->e,b->f,c->g (vgl. Aufgabe CX.4,S. 303).

Damit läßt sich der Satz von König,wenn auch nicht exakt beweisen,so doch mit Schulmittel didaktisch-methodisch plausibel machen.

Somit läßt sich eine Unterrichtsreihe konzipieren,in der die Themen Maximaler Netzfluss,Maximales Matching,Verbindungszahl und Trennungszahl nach Menger,der Satz von Ford-Fulkerson,die Sätze von Menger und der Satz von König in gegenseitiger Beziehung zueinander didaktisch betrachtet und begründet werden können.Die Unterrichtsreihe kann mit den Themen Minimale Kosten,Transportprobleme,Optimales Matching und Chinesisches Briefträgerproblem ergänzt und fortgesetzt werden (vgl. Kapitel CXIII,CXIV).

Außerdem läßt sich ein didaktischer Zusammenhang zum Algorithmus Eigenschaften/Brücken (und Artikulationspunkte) herstellen,da die Minimalzahl der Knotenverbindungszahlen zwischen verschiedenen Knoten eines Graphen der Zusammenhangszahl des Graphen entspricht,d.h. bei einer minimalen Verbindungszahl von 2 gibt es keine Artikulationspunkte.Es gilt:minimale Knotenverbindungszahl <= minimale Kantenverbindungszahl<=kleinster (ungerichteter)Knotengrad (Knotengrad mittels Menü Eigenschaften/Knoten,Lit.55,S 220ff,als Beispiele die Graphen Menger im Verzeichnis Graphen,Lit 55,S 211)

Die Objekte TInhaltsgraph,TInhaltsknoten und TInhaltskante besitzen jetzt auch die Datenfelder Zeichenflaeche_:TCanvas,Ausgabelabel1:TLabel; Ausgabelabel2_:TLabel,TInhaltsgraph und darüber hinaus das Datenfeld Strliste: TStringlist sowie die entsprechenden Propertys.Dadurch ist es möglich die Zeichenflaeche Paintbox,die Label Ausgabe1 und Ausgabe2 und eine Stringliste vom Typ TStringlist direkt in den genannten Objekten zu speichern,wodurch sich die vereinfachten (Parameterreduzierten) Methoden ZeichnedenGraph und DenGraphzeichnen(F:TColor;T:TPenstyle) in TInhaltsgraph sowie ZeigePfadliste(Zeichnen:Boolean;LetzterPfad:Boolean) in TInhaltsknoten zusätzlich implementieren lassen,die die Methoden ZeichneGraph und GraphZeichnen sowie AnzeigePfadliste mit den entsprechenden Parametern ersetzen.

Neu implementiert wurde auch die Methode KantenlistealsGraph die zu einer Kantenliste einen Graphen vom Typ TInhaltsgraph zurückgibt,der als Kantenliste die vorgegebene Kantenliste und als Knotenliste deren Knoten enthält.Unter Zuhilfenahme der oben genannten Methoden lassen sich so Pfade (auch im Demomodus mit Umfärbungen) leichter zeichnerisch darstellen.

Der Demomodus der Algorithmen des Menüs Pfade wurde durch Anwendung der im letzten Abschnitt genannten Methoden verbessert:Die Anzeige der Kanten entsprechender Teilgraphen,die den Lösungsalgorithmus kennzeichnen,erfolgt grün gestrichelt.

_