F VI Anhang auf CD

Quelltextlisting der verschiedenen Programme,Lösungen der Klausuraufgaben,Beschreibung der Objektmethoden

Inhaltsverzeichnis

1)Quelltext des Beispielprogramms zur Einführung in die objekt-orientierte Programmierung „ZeichneGraph“ (Einführungsprojekt)

Unit UGraph:

```
unit UGraph;

interface

uses Graphics,Dialogs,Sysutils;

const Max=100;

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);
  function Anfangsknoten:TKnoten;
  function Endknoten:TKnoten;
  procedure Freeall;
  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 Wertschreiben;
  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;

implementation

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;

constructor TKnoten.Create(Xk,Yk:Integer;Rd:Integer);
begin
  inherited Create(Xk,Yk);
  Radius_:=Rd;
end;

function TKnoten.Radius:Integer;
begin
  Radius:=Radius_;
```

```
end;

procedure TKnoten.NeuerRadius(Rd:Integer);
begin
  Radius_:=Rd;
end;

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;

{Notwendige Zusatzmethode,falls die Methoden TFigur.Zeichnen und
TFigur.Loeschen
sowie die Nachfolgermethoden nicht virtual sind.}
{
procedure TKnoten.NeuePosition(Xneu,Yneu:Integer);
begin
  Loeschen;
  NeuesX(XNeu);
  NeuesY(Yneu);
  Zeichnen;
end;
}

constructor
TInhaltsknoten.Create(Xk,Yk:Integer;Rd:Integer;Ih:string);
begin
  inherited Create(Xk,Yk,Rd);
  Inhalt_:=Ih;
end;


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

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

{Notwendige Zusatzmethode,falls die Methoden TFigur.Zeichnen und
TFigur.Loeschen
sowie die Nachfolgermethoden nicht virtual sind.}
{
procedure TInhaltsknoten.NeuePosition(Xneu,Yneu:Integer);
begin
  Loeschen;
  NeuesX(XNeu);
  NeuesY(Yneu);
  Zeichnen;
end;
}

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;

function TKante.Anfangsknoten:TKnoten;
begin
  Anfangsknoten:=Anfangsknoten_;
end;


function TKante.Endknoten:TKnoten;
begin
  Endknoten:=Endknoten_;
end;

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;
```

```
constructor TInhaltskante.Create(AKno,EKno:TKnoten;Ih:string);
begin
  inherited Create(AKno,EKno);
  Inhalt_:=Ih;
 end;

function TInhaltskante.Wertlesen:string;
begin
  Wertlesen:=Inhalt_;
end;

procedure TInhaltskante.Wertschreiben(S:string);
begin
  Inhalt_:=S;
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;

Constructor TGraph.Create;
begin
  inherited Create;
```

```
  Knotenindex_:=0;
  Kantenindex_:=0;
end;

procedure TGraph.Freeall;
var Index:Integer;
begin
  for Index:=1 to Knotenindex_ do
  begin
    Knotenliste_[Index].Free;
    Knotenliste_[Index]:=nil;
  end;
  for Index:=1 to Kantenindex_ do
  begin
    Kantenliste_[Index].Free;
    Kantenliste_[Index]:=nil;
  end;
  inherited Free;
end;


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;

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;

{Zusatz-Methode,um zu demonstrieren,daß der Datentyp Array nicht
optimal zur
Speicherung geeignet ist.Der Datentyp Liste eignet sich besser.}

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;

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;

end.

Unit UForm:

unit UForm;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
Controls,
  Forms, Dialogs, Menus,
  UFigur;

type
  TKnotenformular = class(TForm)
    MainMenu: TMainMenu;
```

```
    Ende: TMenuItem;
    Punktezeichnen: TMenuItem;
    Knotenzeichnen: TMenuItem;
    InhaltsKnotenzeichnen: TMenuItem;
    Kantezeichnen: TMenuItem;
    Graphzeichen: TMenuItem;
    procedure EndeClick(Sender: TObject);
    procedure PunktezeichnenClick(Sender: TObject);
    procedure KnotenzeichnenClick(Sender: TObject);
    procedure InhaltsKnotenzeichnenClick(Sender: TObject);
    procedure KantezeichnenClick(Sender: TObject);
    procedure GraphzeichenClick(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);
    procedure FormMouseMove(Sender: TObject; Shift: TShiftState;
     X,Y: Integer);

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

var
  {Alle Anwendungen:}
  Knotenformular: TKnotenformular;
  Objekt:TObject;
  {Punkt zeichnen:}
  Punkt:TPunkt;
  {Knoten zeichnen:}
  Knoten:TKnoten;
  {Inhaltsknoten zeichnen:}
  Inhaltsknoten:TInhaltsKnoten;
  {Kante zeichnen:}
  Kante:TKante;
  {Kante zeichnen und Graph zeichnen:}
  ZweiterKnoten:Boolean;
  {Graph (mit Inhaltskanten) zeichnen:}
  Graph:TGraph;
 Inhaltsknoten1,Inhaltsknoten2,IKnoten,IKnoten1,IKnoten2:TInhaltsknoten;
  Inhaltskante:TInhaltskante;
  Knoten1,Knoten2:TKnoten;

implementation
```

```
{$R *.DFM}
procedure TKnotenformular.Init;
begin
  Punkt.Loeschen;
  Knoten.Loeschen;
  Inhaltsknoten.Loeschen;
  if Kante.Anfangsknoten<>nil then Kante.AnfangsKnoten.Loeschen;
  if Kante.Endknoten<>nil then Kante.Endknoten.Loeschen;
  Kante.Loeschen;
  Kante.Freeall;
  Kante:=TKante.create(nil,nil);
  Graph.Loeschen;
end;

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

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

procedure TKnotenformular.KnotenzeichnenClick(Sender: TObject);
begin
  Init;
  Knoten.Zeichnen;
  Objekt:=Knoten;
end;

procedure TKnotenformular.InhaltsKnotenzeichnenClick(Sender:
TObject);
begin
  InhaltsKnoten.Wert:=InputBox(‘Eingabe’, ‘Eingabe:’,’0');
  Inhaltsknoten.zeichnen;
  Objekt:=Inhaltsknoten;
end;

procedure TKnotenformular.KantezeichnenClick(Sender: TObject);
begin
  Init;
  Objekt:=Kante;
end;

procedure TKnotenformular.GraphzeichenClick(Sender: TObject);
begin
   Init;
```

```
   Objekt:=Graph;
end;


procedure TKnotenformular.FormCreate(Sender: TObject);
begin
  {Alle Anwendungen:}
  Oberflaeche:=KnotenFormular.Canvas;
  Objekt:=nil;
  {Punkt zeichen:}
  Punkt:=TPunkt.Create(320,240);
  {Knoten zeichnen:}
  Knoten:=TKnoten.Create(320,240,15);
  {Inhaltsknoten zeichne:}
  Inhaltsknoten:=TInhaltsKnoten.Create(320,240,15,'0');
  {Kante zeichnen:}
  Kante:=TKante.Create(nil,nil);
  {Graph zeichnen und Kante zeichnen:}
  ZweiterKnoten:=false;
  {Graph zeichnen:}
  Graph:=TGraph.Create;
end;

procedure TKnotenformular.FormDestroy(Sender: TObject);
begin
  {Punkt zeichnen:}
  Punkt.Free;
  Punkt:=nil;
  {Knoten zeichnen:}
  Knoten.Free;
  Knoten:=nil;
  {Inhaltsknoten zeichnen:}
  Inhaltsknoten.Free;
  Inhaltsknoten:=nil;
  {Kante zeichnen}
  Kante.Freeall;
  Kante:=nil;
  {Graph zeichnen:}
  Graph.Freeall;
  Graph:=nil;
end;


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;
  if Objekt=Kante then Kante.Zeichnen;
  {Graph zeichnen:}
  if Objekt=Graph then Graph.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);
  {Knoten zeichnen:}
  if Objekt=Knoten then Knoten.NeuePosition(X,Y);
  {Inhaltsknoten zeichnen:}
  if Objekt=Inhaltsknoten then Inhaltsknoten.NeuePosition(X,Y);
  {Kante zeichnen:}
  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.Freeall;
      Kante:=TKante.Create(Inhaltsknoten1,Inhaltsknoten2);
      Objekt:=Kante;
      Kante.Zeichnen;
    end;
  end;
 {Graph zeichnen:}
  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
      begin
        Showmessage('Kein Knoten!');
        ZweiterKnoten:=false;
      end;
   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;
end;

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;

end.
```

Projekt-Quelltext:

```
program Project1;

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

{$R *.RES}

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

2)Quelltext des Programms „Objekt-Liste“

Unit UList:

```
unit UList;
{$F+}

interface

type

  TList = class(TObject)
  private
    Weiter_:TList;
  end;

  TElement=class(TList)
   private
     OB_:TObject;
   public
     constructor Create(Ob:TObject);
     function Objekt:TObject;
   end;

  TVorgang=Procedure (X:TObject);

  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;
```

```
 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;

implementation

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
    if not Leer
    then
      Erstes := ErstesElement.Objekt;
end;

function TListe.Letztes:TObject;
begin
    if not Leer
    then
      Letztes:= LetztesElement.Objekt;
end;
```

```
function TListe.Anzahl:Integer;
begin
  if Leer
  then
    Anzahl:=0
  else
    Anzahl:= TListe(Weiter_).Anzahl+1;
end;

function TListe.Element(Index:Integer):TElement;
begin
  if (Index<0)or(Index>Anzahl-1)
  then
    Element:=nil
  else
    if Leer
    then
      Element:=nil
    else
      if Index=0
      then
        Element:=ErstesElement
      else
        Element:=TListe(Weiter_).Element(Index-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;


function TListe.Position(Ob:TObject):Integer;
var Index:Integer;
    L:TListe;
    X:TElement;
begin
  X:=TElement.Create(Ob);
  Index:=0;
  L:=self;
  while (L.Erstes<>X) and (not L.Leer) do
  begin
    Index:=Index+1;
    L:=TListe(L.Weiter_);
  end;
  if L.Leer
  then
    Position:=-1
  else
    Position:=Index;
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.AnPositioneinfuegen(Ob:TObject;Index:Integer);
var Y:TListe;
```

```
    X:TElement;
begin
  X:=TElement.Create(Ob);
  Y:=Liste(Index);
  if Y<>nil
  then
  begin
    X.Weiter_:=Y.Weiter_;
    Y.Weiter_:=X;
   end
   else
   if Index=0
   then
   begin
     Y:=self;
     X.Weiter_:=nil;
     Y.Weiter_:=X;
   end;
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.Loeschen(Ob:TObject);
var Y,Z:TListe;
begin
  if not Leer
  then
    if ErstesElement.Objekt=Ob
    then
    begin
      Y:=self;
      Z:=TListe(Y.Weiter_);
      Y.Weiter_:=TListe(Y.Weiter_).Weiter_;
      Z.Weiter_:=nil;
      Z.Free;
      Z:=nil;
    end
    else
      TListe(Weiter_).Loeschen(Ob);
 end;

procedure TListe.AnPositionLoeschen(Index:Integer);
var Y,Z:TListe;
begin
  Y:=Liste(Index);
  if  (Y<>nil) and (not Y.Leer)
  then
  begin
    Z:=TListe(Y.Weiter_);
    Y.Weiter_:=Y.Weiter_.Weiter_;
    Z.Weiter_:=nil;
    Z.Free;
  end;
end;

procedure TListe.FueralleElemente(Vorgang:TVorgang);
begin
  if not Leer
  then
  begin
    Vorgang(ErstesElement.Objekt);
    TListe(Weiter_).FueralleElemente(Vorgang);
  end;
end;

procedure TListe.FueralleElementezurueck(Vorgang:TVorgang);
begin
  if not Leer
  then
  begin
    Weiter.FueralleElementezurueck(Vorgang);
    Vorgang(ErstesElement.Objekt);
```

```
    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;

end.
```

Projekt-Quelltext/Hauptprogramm:

```
program ListTest;
{$F+}

uses
  Ulist in 'ULIST.PAS',
  Wincrt;

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;

var
  Keller:TKeller;
  Schlange:TSchlange;
  SListe: TSListe;
  SObject:TSObject;
  Eingabe:string;
  K :Integer;


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;

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;
  Write(‘Erstes/Letztes Element: ‘);
  if not SListe.Leer
  then
  begin
    TSObject(SListe.Erstes).Ausgabe;
    TSObject(SListe.Letztes).Ausgabe;
  end;
  Writeln;
  Writeln;
  Write(‘An welcher Position soll eingefügt werden? ‘);
  Readln(K);
  Writeln;
  Write(‘String: ‘);
  Readln(Eingabe);
  SObject:=TSObject.Create(Eingabe);
  SListe.AnPositioneinfuegen(SObject,K);
  Writeln;
  Write(‘Listenausgabe: ‘);
  SListe.Ausgabevorwaerts;
  Writeln;
  Writeln;
```

```
Writeln(‘Letztes eingefügtes Element wieder löschen:’);
Writeln;
SListe.Loeschen(SObject);
Write(‘Listenausgabe: ‘);
SListe.Ausgabevorwaerts;
Writeln;
Writeln;
Writeln(‘Ausgabe des k.ten Elements:’);
Writeln;
Write(‘Welches k? ‘);
Readln(K);
Writeln;
Write(K,’.tes Element:’);
if (K>=0)and (K<SListe.Anzahl)
then
  TSObject(SListe.Objekt(K)).Ausgabe;
Writeln;
Writeln;
Writeln(‘Zufälliges Löschen: ‘);
Writeln;
while SListe.Anzahl > 0 do
begin
  Randomize;
  K :=Random(SListe.Anzahl);
  Write(‘Zu löschendes ‘, K,’.Element= ‘);
  TSObject(SListe.Objekt(K)).Ausgabe;
  Writeln;
  Writeln;
  Writeln;
  Readln;
  SListe.AnPositionLoeschen(K);
  Writeln(‘Noch zu löschende Elemente:’);
  SListe.Ausgabevorwaerts;
  WriteLn;
  Writeln;
end;
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.
```

3)Quelltexte und Lösungen der Programme „Figuren“ und „Ziffernliste“ als Lösung der entsprechenden Klausuraufgaben

a)Klausur Figuren:

Quelltext:

Unit UGraph:

```
unit UGraph;


interface

uses Graphics,Wintypes;

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;


  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;

  TQuadrat=class(TRechteck)
    constructor Create(Pt:TPoint;Seite:Integer);
  end;

  TDreieck=class(TQuadrat)
    procedure Zeichnen;override;
    procedure Loeschen;override;
  end;


var Oberflaeche:TCanvas;

implementation

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(Pt:TPoint);
begin
  inherited Create;
  P_:=Pt;
end;

function TPunkt.Punkt:TPoint;
begin
  Punkt:=P_;
end;

procedure TPunkt.NeuerPunkt(Pt:TPoint);
begin
  P_:=Pt;
end;

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

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

procedure TPunkt.NeuePosition(Pt:TPoint);
begin
  Loeschen;
  NeuerPunkt(Pt);
  Zeichnen;
end;

constructor TStrecke.Create(Pt:TPoint;NeueLaenge:Integer);
begin
```

```
  inherited create(Pt);
  Laenge_:=NeueLaenge;
end;

function TStrecke.Laenge:Integer;
begin
  Laenge:=Laenge_;
end;

procedure TStrecke.NeueLaenge(L:Integer);
begin
  Laenge_:=L;
end;

procedure TStrecke.Zeichnen;
var HilfFarbe:TColor;
begin
  Hilffarbe:=Farbe;
  NeueFarbe(clwhite);
  Oberflaeche.Moveto(Punkt.X,Punkt.Y);
  NeueFarbe(clblack);
  Oberflaeche.Lineto(Punkt.X+Laenge_,Punkt.Y);
  NeueFarbe(Hilffarbe);
end;

procedure TStrecke.Loeschen;
var HilfFarbe:TColor;
begin
  Hilffarbe:=Farbe;
  NeueFarbe(clwhite);
  Oberflaeche.Moveto(Punkt.X,Punkt.Y);
  Oberflaeche.Lineto(Punkt.X+Laenge_,Punkt.Y);
  NeueFarbe(Hilffarbe);
end;

constructor
TRechteck.Create(Pt:TPoint;NeueLaenge,NeueBreite:Integer);
begin
  inherited create(Pt,NeueLaenge);
  Breite_:=NeueBreite;
end;

function TRechteck.Breite:Integer;
begin
  Breite:=Breite_;
end;


procedure TRechteck.NeueBreite(B:Integer);
begin
```

```
  Breite_:=B;
end;

procedure TRechteck.Zeichnen;
var HilfFarbe:TColor;
begin
  Hilffarbe:=Farbe;
  NeueFarbe(clwhite);
  Oberflaeche.Moveto(Punkt.X,Punkt.Y);
  NeueFarbe(clblack);
  Oberflaeche.Moveto(Punkt.X,Punkt.Y);
  Oberflaeche.Lineto(Punkt.X+Laenge,Punkt.Y);
  Oberflaeche.Lineto(Punkt.X+Laenge,Punkt.Y+Breite);
  Oberflaeche.Lineto(Punkt.X,Punkt.Y+Breite);
  Oberflaeche.Lineto(Punkt.X,Punkt.Y);
  NeueFarbe(Hilffarbe);
end;

procedure TRechteck.Loeschen;
var HilfFarbe:TColor;
begin
  Hilffarbe:=Farbe;
  NeueFarbe(clwhite);
  Oberflaeche.Moveto(Punkt.X,Punkt.Y);
  Oberflaeche.Moveto(Punkt.X,Punkt.Y);
  Oberflaeche.Lineto(Punkt.X+Laenge,Punkt.Y);
  Oberflaeche.Lineto(Punkt.X+Laenge,Punkt.Y+Breite);
  Oberflaeche.Lineto(Punkt.X,Punkt.Y+Breite);
  Oberflaeche.Lineto(Punkt.X,Punkt.Y);
  NeueFarbe(Hilffarbe);
end;

constructor TQuadrat.Create(Pt:TPoint;Seite:Integer);
begin
  inherited Create(Pt,Seite,Seite);
end;

procedure TDreieck.Zeichnen;
var HilfFarbe:TColor;
begin
  Hilffarbe:=Farbe;
  NeueFarbe(clwhite);
  Oberflaeche.Moveto(Punkt.X,Punkt.Y);
  NeueFarbe(clblack);
  Oberflaeche.Lineto(Punkt.X+Laenge,Punkt.Y);
  Oberflaeche.Lineto(Punkt.X+Laenge,Punkt.Y+Laenge);
  Oberflaeche.Lineto(Punkt.X,Punkt.Y);
  NeueFarbe(Hilffarbe);
end;
```

```
procedure TDreieck.Loeschen;
var HilfFarbe:TColor;
begin
  Hilffarbe:=Farbe;
  NeueFarbe(clwhite);
  Oberflaeche.Moveto(Punkt.X,Punkt.Y);
  Oberflaeche.Lineto(Punkt.X+Laenge,Punkt.Y);
  Oberflaeche.Lineto(Punkt.X+Laenge,Punkt.Y+Laenge);
  Oberflaeche.Lineto(Punkt.X,Punkt.Y);
  NeueFarbe(Hilffarbe);
end;

end.
```

Unit UBeisp:

```
unit UBeisp;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
Controls,
  Forms, Dialogs, Menus, StdCtrls,UFigur;

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

{$R *.DFM}

procedure TFormneu.FormActivate(Sender: TObject);
begin
  Oberflaeche:=Canvas;
  Dreieck:=TDreieck.Create(Point(50,60),70);
  Objekt:=nil;
end;

procedure TFormneu.FormDestroy(Sender: TObject);
begin
  Dreieck.Free;
end;

procedure TFormneu.FormPaint(Sender: TObject);
begin
  if Objekt=Dreieck then Dreieck.Zeichnen;
end;

procedure TFormneu.FormMouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
begin
  If Button=mbleft then Dreieck.NeuePosition(Point(X,Y));
end;


procedure TFormneu.DreieckzeichnenClick(Sender: TObject);
begin
  Dreieck.Zeichnen;
  Objekt:=Dreieck;
end;

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

end.
```

Projektquelltext:

```
program Klausur;

uses
  Forms,
  Ufigur in ‘UFIGUR.PAS’,
  Ubeisp in ‘UBEISP.PAS’ {Formneu};
```

```
{$R *.RES}

begin
  Application.CreateForm(TFormneu, Formneu);
  Application.Run;
end.
```

b)Klausur Ziffernliste:

Quelltext:

Unit UList:

```
unit Ulist;
{$F+}

interface

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 LetztesElement:TElement;
    function ErstesElement:TElement;
    function Erstes: TObject;
    function Letztes: TObject;
    function Anzahl:Integer;
    procedure AmAnfanganfuegen(Ob:TObject);
    procedure AmEndeAnfuegen(Ob:TObject);
    procedure AmAnfangLoeschen(var Ob:TObject);
    procedure AmEndeLoeschen(var Ob:TObject);
  end;

implementation
```

```
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 Weiter_.Free;
  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.ErstesElement:TElement;
begin
  if not Leer
  then
    ErstesElement := TElement(Weiter_)
end;

function TListe.Erstes:TObject;
begin
    if not Leer
    then
      Erstes := ErstesElement.Objekt;
end;
```

```
function TListe.Letztes:TObject;
begin
    if not Leer
    then
      Letztes:= LetztesElement.Objekt;
end;

function TListe.Anzahl:Integer;
begin
  if Leer
  then
    Anzahl:=0
  else
    Anzahl:= TListe(Weiter_).Anzahl+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;

function TListe.LetztesElement:TElement;
begin
  if Leer
  then
    LetztesElement:=TElement(self)
  else
    LetztesElement:= TListe(Weiter_).LetztesElement;
end;


end.
```

Projekt-Quelltext/Hauptprogramm:

```
program Ziffern;
{$F+}
```

```
uses
  UList in ‘ULIST.PAS’,
  Wincrt;

type

 TZiffer=0..1;

  TZiffernObject = class(TObject)
  private
    Ziffer_:TZiffer;
  public
    constructor Create(Z:TZiffer);
    function Ziffer:TZiffer;
    procedure NeueZiffer(Z:TZiffer);
  end;

  TZiffernListe=class(TListe)
    procedure Gib_zahl_ein;
    procedure Gib_zahl_aus;
    function Paritaet:Boolean;
    procedure Addition(Zahl:TZiffernliste);
    procedure GleicheLaenge(Zahl:TZiffernliste);
  end;

var
  Zahl1,Zahl2:TZiffernliste;

constructor TZiffernObject.Create(Z:TZiffer);
begin
  inherited Create;
  Ziffer_:= Z;
end;

function TZiffernObject.Ziffer:TZiffer;
begin
  Ziffer:=Ziffer_;
end;

procedure TZiffernObject.NeueZiffer(Z:TZiffer);
begin
  Ziffer_:=Z;
end;


procedure TZiffernliste.Gib_Zahl_ein;
var Eingabe:Integer;
begin
  WriteLn(‘Ziffern 0 oder 1 einzeln eingeben (nach jeder Ziffer
Return drücken.
```

```
  Eingabeende mit 2):');
  Writeln;
  Readln(Eingabe);
  while  Eingabe in [0,1]  do
  begin
    AmAnfangAnfuegen(TZiffernObject.Create(Eingabe));
    Readln(Eingabe);
  end;
  Writeln('Eingabe fertig');
end;

procedure TZiffernliste.Gib_Zahl_aus;
begin
  if not Leer
  then
    TZiffernliste(Weiter).Gib_Zahl_aus;
  if not Leer then Write(TZiffernObject(Erstes).Ziffer);
end;

function TZiffernliste.Paritaet:Boolean;
var Summe:Integer;
    Y:TZiffernliste;
begin
  Summe:=0;
  Y:=self;
  While not Y.Leer do
  begin
    Summe:=Summe+TZiffernObject(Y.Erstes).Ziffer;
    Y:=TZiffernListe(Y.Weiter);
  end;
  if odd(Summe)
  then
    Paritaet:=false
  else
    Paritaet:=true;
end;

procedure TZiffernliste.Addition(Zahl:TZiffernliste);
var Neuziffer,Uebertrag:TZiffer;
    Z:TZiffernliste;

  procedure Stelle(Ziffer1,Ziffer2:TZiffer;var
Neuziffer,Uebertrag:TZiffer);
  begin
     Neuziffer:=(Ziffer1+Ziffer2+Uebertrag) mod 2;

     Uebertrag:=(Ziffer1+Ziffer2+Uebertrag) div 2;
   end;

begin
```

```
  Z:=self;
  Uebertrag:=0;
  While not Z.Leer do
  begin
    Stelle(TZiffernObject(Z.Erstes).Ziffer,
        TZiffernObject(Zahl.Erstes).Ziffer,
        Neuziffer,Uebertrag);
     TZiffernObject(Z.Erstes).NeueZiffer(Neuziffer);
      Z:=TZiffernliste(Z.Weiter);
      Zahl:=TZiffernliste(Zahl.Weiter);
   end;
   if Uebertrag=1
   then
     Z.AmAnfangAnfuegen(TZiffernObject.Create(Uebertrag));
end;

procedure TZiffernliste.GleicheLaenge(Zahl:Tziffernliste);
var Index,Anzahl1,Anzahl2:Integer;
    Z:TZiffernliste;
begin
  Anzahl1:=Anzahl;
  Anzahl2:=Zahl.Anzahl;
  if Anzahl1>Anzahl2
  then
    Z:=Zahl
  else
    Z:=Self;
  for index:=1 to abs(Anzahl1-Anzahl2) do
    Z.AmEndeAnfuegen(TZiffernObject.Create(0));
end;

begin
  InitWincrt;
  Writeln('Zahlen erzeugen:');
  Zahl1:=TZiffernliste.Create;
  Zahl2:=TZiffernliste.Create;
  Writeln('Zahl1 eingeben:');
  Zahl1.Gib_Zahl_ein;
  Writeln('Zahl2 eingeben:');
  Zahl2.Gib_Zahl_ein;
  Write('Zahl1:');
  Zahl1.Gib_Zahl_aus;
  Writeln;
  Write('Parität:');
  Writeln(Zahl1.Paritaet);
  Write('Zahl2:');
  Zahl2.Gib_Zahl_aus;
  Writeln;
  Write('Paritaet:');
  Writeln(Zahl2.Paritaet);
```

```
Write(‘Summe:’);
  Zahl1.GleicheLaenge(Zahl2);
  Zahl1.Addition(Zahl2);
  Zahl1.Gib_Zahl_aus;
  Writeln;
  Readln;
  Clrscr;
  Writeln(‘Ende’);
  Readln;
  DoneWincrt;
end.
```

Lösungen: (Teilaufgabe c bis i)

```
var
  Zahl1,Zahl2:TZiffernliste;

constructor TZiffernObject.Create(Z:TZiffer);
begin
  inherited Create;
  Ziffer_:= Z;
end;

function TZiffernObject.Ziffer:TZiffer;
begin
  Ziffer:=Ziffer_;
end;

procedure TZiffernObject.NeueZiffer(Z:TZiffer);
begin
  Ziffer_:=Z;
end;

procedure TZiffernliste.Gib_Zahl_ein;
var Eingabe:Integer;
begin
  WriteLn(‘Ziffern 0 oder 1 eingeben:’);
  Writeln;
  Readln(Eingabe);
  while  Eingabe in [0,1]  do
  begin
    AmAnfangAnfuegen(TZiffernObject.Create(Eingabe));
    Readln(Eingabe);
  end;
  Writeln(‘Eingabe fertig’);
end;
```

```
procedure TZiffernliste.Gib_Zahl_aus;
begin
  if not Leer
  then
    TZiffernliste(Weiter).Gib_Zahl_aus;
  if not Leer then Write(TZiffernObject(Erstes).Ziffer);
end;

function TZiffernliste.Paritaet:Boolean;
var Summe:Integer;
    Y:TZiffernliste;
begin
  Summe:=0;
  Y:=self;
  While not Y.Leer do
  begin
    Summe:=Summe+TZiffernObject(Y.Erstes).Ziffer;
    Y:=TZiffernListe(Y.Weiter);
  end;
  if odd(Summe)
  then
    Paritaet:=false
  else
    Paritaet:=true;
end;


procedure TZiffernliste.Addition(Zahl:TZiffernliste);
var Neuziffer,Uebertrag:TZiffer,Z:TZiffernliste;

  procedure Stelle(Ziffer1,Ziffer2:TZiffer;var
Neuziffer,Uebertrag:TZiffer);
  begin
     Neuziffer:=(Ziffer1+Ziffer2+Uebertrag) mod 2;
     Uebertrag:=(Ziffer1+Ziffer2+Uebertrag) div 2;
   end;
begin
  Z:=self;
  Uebertrag:=0;
  While not Z.Leer do
  begin
    Stelle(TZiffernObject(Z.Erstes).Ziffer,
        TZiffernObject(Zahl.Erstes).Ziffer,
        Neuziffer,Uebertrag);
     TZiffernObject(Z.Erstes).NeueZiffer(Neuziffer);
      Z:=TZiffernliste(Z.Weiter);
      Zahl:=TZiffernliste(Zahl.Weiter);
   end;
   if Uebertrag=1
   then
```

```
     Z.AmAnfangAnfuegen(TZiffernObject.Create(Uebertrag));
end;

procedure TZiffernliste.GleicheLaenge(Zahl:Tziffernliste);
var Index,Anzahl1,Anzahl2:Integer;
    Z:TZiffernliste;
begin
  Anzahl1:=Anzahl;
  Anzahl2:=Zahl.Anzahl;
  if Anzahl1>Anzahl2
  then
    Z:=Zahl
  else
    Z:=Self;
  for index:=1 to abs(Anzahl1-Anzahl2) do
    Z.AmEndeAnfuegen(TZiffernObject.Create(0));
end;

begin
  InitWincrt;
  Writeln('Zahlen erzeugen:');
  Zahl1:=TZiffernliste.Create;
  Zahl2:=TZiffernliste.Create;
  Writeln('Zahl1 eingeben:');
  Zahl1.Gib_Zahl_ein;
  Writeln('Zahl2 eingeben:');
  Zahl2.Gib_Zahl_ein;
  Write('Zahl1:');
  Zahl1.Gib_Zahl_aus;
  Writeln;
  Write('Parität:');
  Writeln(Zahl1.Paritaet);
  Write('Zahl2:');
  Zahl2.Gib_Zahl_aus;
  Writeln;
  Write('Paritaet:');
  Writeln(Zahl2.Paritaet);
  Write('Summe:');
  Zahl1.GleicheLaenge(Zahl2);
  Zahl1.Addition(Zahl2);
  Zahl1.Gib_Zahl_aus;
  Writeln;
  Readln;
  Clrscr;
  Writeln('Ende');
  Readln;
  DoneWincrt;
end
```

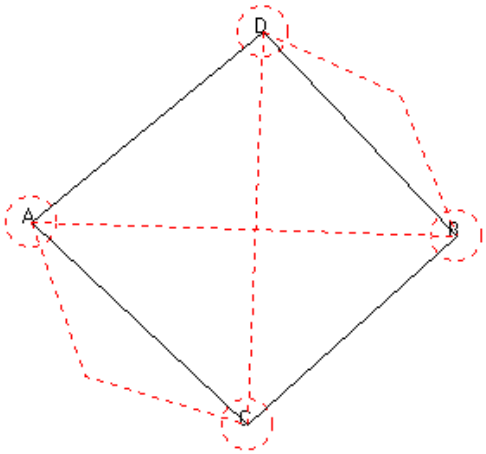
4)Lösungen der übrigen Klausuren,Algorithmus nach Ford zur Abstandsbestimmung

I)Lösung Klausur Eulerproblem:

Lösung:

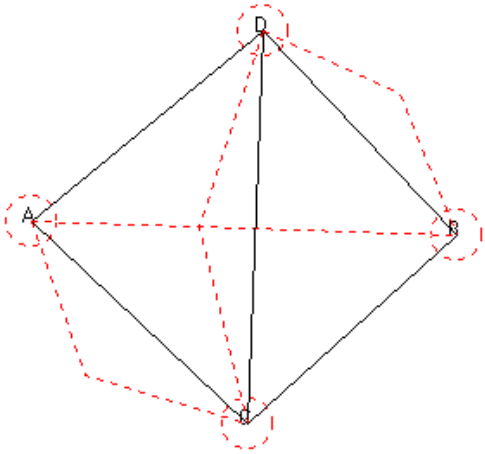
a)
Graph 1,Graph 2,Graph 4:keine Eulerlinien
Graph 3: A C A D B C D B A

b)Graph 3:



Alle Kanten befinden sich in disjunkten Kreisen und werden gelöscht.

Graph 4:



Eine Kante z.B. CD bleibt übrig.

Beweis des Satzes:siehe Kapitel C V/Satz C V 1
c)Quelltext der Methoden:siehe Methoden der Unit UGrapC und Unit UInhgrphC
d)

```
function TKnoten.KnotenGrad:Integer;
begin
    KnotenGrad:=Kantenzahl;
end;

function TGraph.IstEulergraph:Boolean;
var IstEuler:Boolean;
    Index,Kantenzahl:Integer;
begin
  IstEuler:=true;
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kantenzahl:=Knotenliste.Knoten(Index).Knotengrad;
      if  odd(Kantenzahl) then IstEuler:=false;
    end;
  IstEulergraph:=IstEuler;
end;
```

e)

```
TEulerCgraph   = class(TInhaltsgraph)
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    procedure
Euler(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;var
Kliste:TKantenliste;
      EineLoesung:Boolean;var Gefunden:Boolean);
    procedure
Eulerlinie(Anfangsknoten,Endknoten:TInhaltsknoten);
    procedure Menu;
  end;
```

f)(Alternativ Methode Eulerfix von S.242)

```
procedure
TEulerCgraph.Euler(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;
  Kliste:TKantenliste);
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 Ka.Besucht) and
(Stufe<=Self.AnzahlKantenmitSchlingen)) then
        begin
          Ka.Pfadrichtung:=Zkno;
          Ka.Besucht:=true;
          Kliste.AmEndeanfuegen(Ka);
          Stufe:=Stufe+1;
          if (Zkno=Zielknoten)and
(Stufe=Self.AnzahlKantenmitSchlingen+1)  then
            TInhaltskante(Kliste).Listenausgabe;
          else
            Euler(Stufe,Zkno,Zielknoten,Kliste);
          Stufe:=Stufe-1;
          Ka.Besucht:=false;
          if not Kliste.Leer then
            Kliste.AmEndeloeschen(TObject(Ka));;
        end;
     end;
end;
```

g)siehe das besprochene Schema eines Backtrackingalgorithmus

II)Lösungen der Abiturklausur:

a)Graph1: kein offene Eulerlinie
  Graph2: ACADABCDB
  Graph3: BACADACBDC
  Graph4: keine offene Eulerlinie,aber geschlossene Eulerlinie

Graph 4 enthält zusätzlich die Kante BC.Diese Kante ist in der geschlossener Eulerlinie von Graph 4 enthalten.Wird diese Kante aus dem Pfad entfernt,entsteht eine offene Eulerlinie zwischen B und C.
Fazit: Ein Graph hat genau dann eine offene Eulerlinie,falls der um eine Kante erweiterte Graph eine geschlossene Eulerlinie hat.

Tabelle Knotengrad:

Graph	A	B	C	D
1	5	3	3	
2	5	3	4	4
3	6	3	5	4
4	6	4	6	4

Ein zusammenhängender Graph hat eine offene Eulerlinie genau

dann,wenn genau zwei Knoten ungeraden Grad und alle anderen Knoten geraden Grad haben.

Beweis des Satzes:
I)Wenn G eine offene Eulerlinie hat, entsteht durch Hinzufügen einer Kante zwischen den Anfangs-und Endknoten des Pfades ein Graph mit geschlossener Eulerlinie.Alle Knoten haben geraden Knotengrad.Durch Entfernen der Kante haben der Anfangs-und Endknoten (und nur diese) ungeraden Knotengrad.

II)Haben genau zwei Knoten ungeraden Grad in G, entsteht durch Hinzufügen einer Kante zwischen diesen Knoten ein Graph mit geschlossener Eulerlinie,da dann alle Knotengrade gerade sind.Durch Entfernen dieser Kante aus der geschlossenen Eulerlinie entsteht eine offene Eulerlinie.

b) Die Methode bestimmt an Hand des Knotengradkriteriums,ob der Graph eine offene Eulerlinie hat oder nicht und gibt einen entsprechenden Boolschen Wert zurück.Außerdem werden,falls eine Eulerlinie existiert,Anfangs-und Endknoten des Pfades als Referenzparameter Kno1 und Kno2 zurückgegeben.Falls keine Eulerlinie existiert,ist der Wert von Kno1 und Kno2 nil.Der Bezeichner der Methode könnte GraphhatoffeneEulerlinie sein.

c)(Alternativ Methode Eulerfix von S.242)

```
procedure
TEulergraph.Euler(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;
Kliste:TKantenliste);
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<=AnzahlKanten)) then
        begin
          Ka.Pfadrichtung:=Zkno;
          Ka.Besucht:=true;
          Kliste.AmEndeanfuegen(Ka);
          Stufe:=Stufe+1;
          if (Zkno=Zielknoten)and (Stufe=AnzahlKanten+1)  then
            TInhaltskante(Kliste).Listenausgabe
          else
            Euler(Stufe,Zkno,Zielknoten,Kliste,EineLoesung);
          Stufe:=Stufe-1;
          Ka.Besucht:=false;
          if not Kliste.Leer then
```

```
            Kliste.AmEndeloeschen(TObject(Ka));;
        end;
     end;
end;
```

Eine Backtrackingmethode eignet sich zur Lösung eines Problems,das nach Stufen gegliedert werden kann und arbeitet rekursiv, wobei ein Stufenparameter bei jedem neuen Methoden-aufruf um eins erhöht wird.In jeder Rekursionsebene wird untersucht,ob eine Erweiterung der Lösung zur nächsten Stufe hin möglich ist.Wenn nicht,wird zur vorigen Rekursionsebene zurückgekehrt,der Stufenparameter um eins vermindert und dort erneut nach einer alternativen Erweiterung zur nächsten Stufe hin gesucht.Falls der Stufenparameter die Endzahl der Stufen des Problems erreicht hat,ist eine Lösung gefunden und kann ausgege-ben werden.Falls der Stufenparameter wieder den Startwert an-nimmt und in der Anfangsrekursionsebene alle Möglichkeiten,eine nächste Stufe zu erreichen, erschöpft sind,ist das Backtrackingverfahren beendet.Die Methode arbeitet nach dem Prinzip „trial and error“.

d)

```
procedure TEulergraph.BestimmeEulerlinie;
var MomentaneKantenliste:TKantenliste;
     Kno1,Kno2,K1,K2:TInhaltsknoten;
     W:string;
begin
  If GraphhatoffeneEulerlinie(Kno1,Kno2) then
  begin
    MomentaneKantenliste:=TKantenliste.create;
    LoescheKantenbesucht;
    Kno1:=TInhaltsknoten.Create;
    repeat
      Readln(W);
      Kno1.Wert:=W;
      K1:=Graphknoten(Kno1);
      K2:=K1;
    until K1<>nil;
    Kno1.Free;
    Kno1:=nil;
    Write('Eingabe des Endknoten: ');
    Kno2:=TInhaltsknoten.create;
    repeat
      Readln(W);
      Kno2.Wert:=W;
      K2:=Graphknoten(Kno2);
    until K2<>nil;
    Kno2.Free;
    Kno2:=nil;
    Writeln('Offene Eulerlinien:');
```

```
    Writeln;
    Euler(1,Kno1,Kno2,MomentaneKantenliste);
    MomentaneKantenliste.Free;
    MomentaneKantenliste:=nil;
  end
  else
     Writeln('Es gibt keine offene Eulerlinie!')
end;
```

III)Lösungen Umfüllaufgabe:

Lösung:

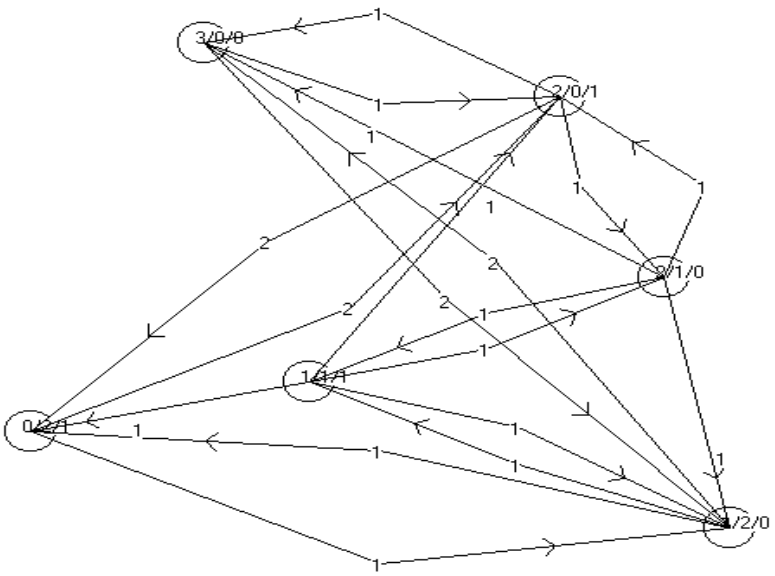
a)z.B.:

```
 3/0/0 2/0/1 2/1/0 1/1/1 1/2/0 0/2/1 Summe: 5
 3/0/0 2/0/1 0/2/1 Summe: 3
 3/0/0 1/2/0 1/1/1 2/1/0 2/0/1 0/2/1 Summe: 7
```

b)

Zustand:	Krug A:	Krug B:	Krug C:	Übergang zu:
1)	3	0	0	2),4)
2)	2	0	1	1),3),6)
3)	2	1	0	1),2),4),5)
4)	1	2	0	1),5),6)
5)	1	1	1	2),3),5),6)
6)	0	2	1	2),4)

c) Graph:



d)Tiefe Baumpfade (Postorder/Preorder):

(Kantenfolge der ausgehenden Kanten gemäß der aufsteigenden Nummerierung der Zielknoten/Zustände in der Tabelle von Aufgabe b)

3/0/0 2/0/1 2/1/0 1/2/0 1/1/1 0/2/1 Summe: 5 Produkt: 1
3/0/0 2/0/1 2/1/0 1/2/0 1/1/1 Summe: 4 Produkt: 1
3/0/0 2/0/1 2/1/0 1/2/0 Summe: 3 Produkt: 1
3/0/0 2/0/1 2/1/0 Summe: 2 Produkt: 1
3/0/0 2/0/1 Summe: 1 Produkt: 1
3/0/0 Summe: 0 Produkt: 0

3/0/0 Summe: 0 Produkt: 0
3/0/0 2/0/1 Summe: 1 Produkt: 1
3/0/0 2/0/1 2/1/0 Summe: 2 Produkt: 1
3/0/0 2/0/1 2/1/0 1/2/0 Summe: 3 Produkt: 1
3/0/0 2/0/1 2/1/0 1/2/0 1/1/1 Summe: 4 Produkt: 1
3/0/0 2/0/1 2/1/0 1/2/0 1/1/1 0/2/1 Summe: 5 Produkt: 1

Perorder: Knoten,dann Kinderknoten (weiter vom Startknoten als der Knoten entfernt)
(gerichteter Binärbaum:Knoten,linker Sohn,rechter Sohn)

Postorder: Kinderknoten (weiter vom Startknoten als der Knoten entfernt), dann Knoten
(gerichteter Binärbaum:linker Sohn,rechter Sohn,Knoten)

Inorder nur bei gerichtetem Binärbaum:Linker Sohn,Knoten,rechter Sohn

Da der Endknoten (0/2/1) als Zielknoten im TiefenBaumdurchlauf vorhanden sein muss,falls es überhaupt eine Lösung gibt,ist damit eine mögliche (nicht unbedingt minimale) Umschüttfolge gefunden.

e)

```
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);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
    Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);{für
```

```
Preorder}
      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));
     {Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
       für Postorder}
      MomentaneKantenliste.AmEndeloeschen(Ob);
      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
    GehezuNachbarknoten(self,AusgehendeKantenliste.Kante(Index));
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
  P:=TGraph.Create;
  P.Knotenliste.AmEndeanfuegen(self);
  Pfadliste.AmAnfanganfuegen(P);{Preorder}
  {Pfadliste.AmEndeanfuegen(P); bei Postorder}
end;

f)
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;
```

Minimale Lösung:

(Abstand von zwei Knoten)

3/0/0 2/0/1 0/2/1 Summe: 3

oder

3/0/0 1/2/0 0/2/1 Summe: 3

Alternativ (ohne Programmierungskentnisse):
e)Siehe den Algorithmus Kürzeste Wege zwischen Knoten nach Dijkstra in Kapitel C III.

1. Durchgang:

(3/0/0)->(1/2/0) Summe :2
(3/0/0)->(2/0/1) Summe :1

Der letzte Pfad wird gelöscht und (2/0/1) zum Ausgangsknoten neuer Pfade gemacht.

2. Durchgang:

(3/0/0)->(2/0/1)->(0/2/1) Summe: 3
(3/0/0)->(2/0/1)->(2/1/0) Summe: 2
(3/0/0)->(1/2/0) Summe : 2

Der letzte Pfad wird gelöscht und (1/2/0) zum Ausgangsknoten neuer Pfade gemacht.

3. Durchgang:

(3/0/0)->(2/0/1)->(0/2/1) Summe: 3
(3/0/0)->(1/2/0)->(0/2/1) Summe: 3
(3/0/0)->(1/2/0)->(1/1/1) Summe: 3
(3/0/0)->(2/0/1)->(2/1/0) Summe: 2

Der letzte Pfad wird gelöscht und (2/1/0) zum Ausgangsknoten neuer Pfade gemacht.

(3/0/0)->(2/0/1)->(0/2/1) Summe: 3 Zielknoten erreicht
(3/0/0)->(1/2/0)->(0/2/1) Summe: 3 Zielknoten erreicht
(3/0/0)->(1/2/0)->(1/1/1) Summe: 3
(3/0/0)->(2/0/1)->(2/1/0)->(1/1/1) Summe:3

Damit ergibt sich als Lösung:

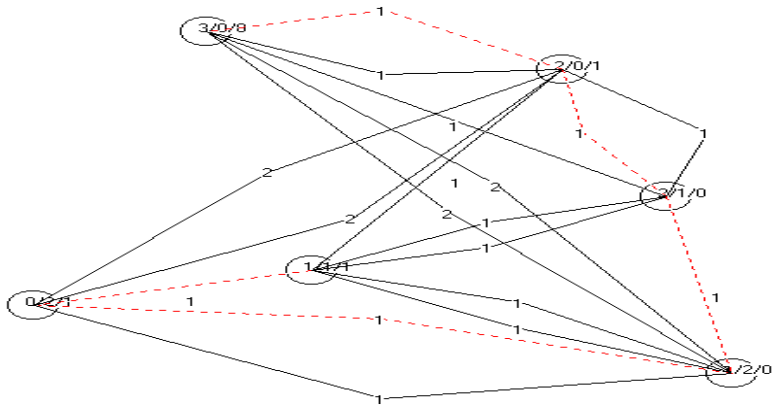
(3/0/0)->(2/0/1)->(0/2/1) Summe: 3
(3/0/0)->(1/2/0)->(0/2/1) Summe: 3

f)Gerüst:

Eine mögliche Auswahl der Kanten:

(3/0/0)->(2/0/1)->(2/1/0)->(1/2/0)->(0/2/1)->(1/1/1)

Alle Kanten haben die minimale Bewertung 1.



Die Gerüstlösung bestimmt eine offene Umfüllkette,die jeden Umfüllzustand genau einmal enthält und minimal bezüglich der Gesamt-Umfüllmenge ist.

g)Hamiltonkreise:

3/0/0 2/0/1 2/1/0 1/1/1 0/2/1 1/2/0 3/0/0 Summe: 7 Produkt: 2
3/0/0 2/0/1 0/2/1 1/2/0 1/1/1 2/1/0 3/0/0 Summe: 7 Produkt: 2
3/0/0 1/2/0 1/1/1 0/2/1 2/0/1 2/1/0 3/0/0 Summe: 8 Produkt: 4

Traveling Salesmann Lösung: 3/0/0 2/0/1 0/2/1 1/2/0 1/1/1 2/1/0 3/0/0 Summe: 7 Produkt: 2

Die Existenz eines Hamilton-Kreises ergibt eine Umfüllkette,die vom Anfangszustand zum gewünschten Zielzustand und von dort wieder zum Anfangszustand zurückführt,wobei alle möglichen Umfüllzustände genau einmal durchlaufen werden.Die Traveling-Salseman-Lösung ist dabei die Lösung mit der minimalsten Umfüllmenge.

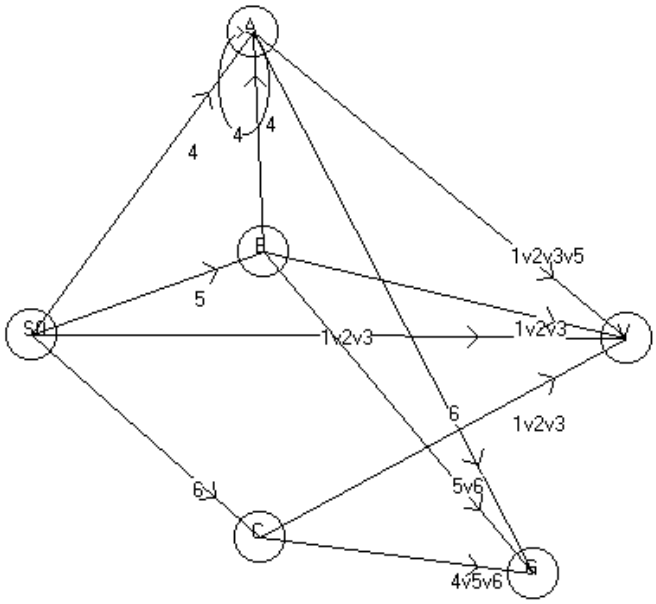
iV)Lösungen Würfelspielaufgabe:

a)Definition endlicher,determinierter Automat:
A=(s,S,S0,F,d)
s:endliche Folge von Zuständen, S:endliche Menge von Eingabezeichen,S0: Anfangszustand,F:Endzustandsmenge, d:Übergangsfunktion d:sxS—>s
Durch das Einlesen der Zeichen des Eingabealphabets wechselt der Automat jeweils in den nächsten Zustand.Die Übergangsfunktion legt fest,welcher Folgezustand abhängig vom momentanen Zustand und vom Eingabezeichen angenommen wird.
Bei einem reduzierten Automaten werden alle die Zustände zu einem gemeinsamen Zustand zusammengefasst,die durch die Eingabe derselben Wörter (Folge von Eingabezeichen) in einen der Endzustände führen.

Die Sprache des Automaten ist die Menge von Worten,die als Folge von Eingabezeichen den Automaten vom Anfangszustand in einen der Endzustände überführen.

b)Gewinn: 4,4,4,6 6,5 5,4,4,6
Verlust:4,4,5 2 5,3

Die Zahlenbeispiel sind auch Wörter der vom Automaten erkannten Sprache.



Zustand V und G sollten durch einen Doppelkreis als Endzustände gekennzeichnet werden.(Außerdem sollte noch ein Fehlerzustand hinzugefügt werden.)

c)Reduzierung gemäß der Pfadregeln der Stochastik:

Schlingenreduzierung gemäß der Regel: $w_a = w_a /(1-w_s)$

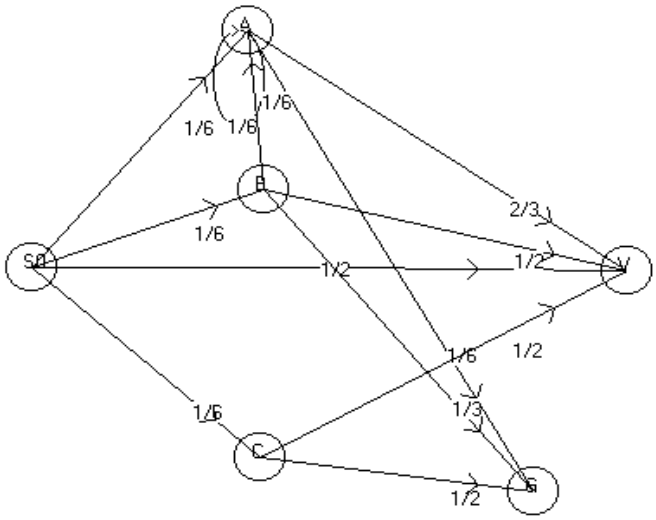
w_a Wahrscheinlichkeit der ausgehenden Kanten

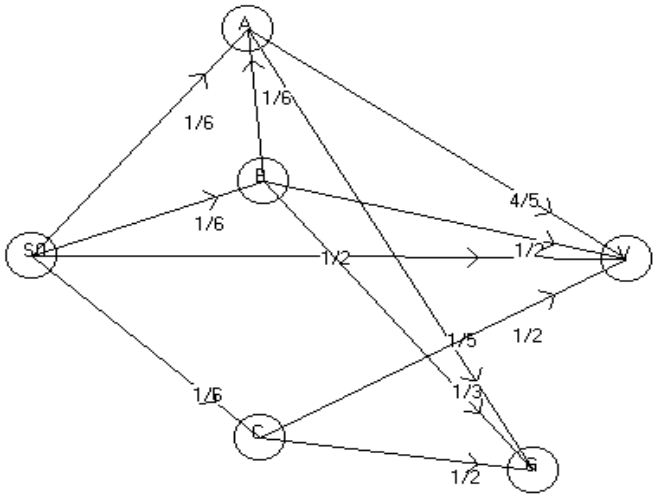
w_s Wahrscheinlichkeit der Schlinge

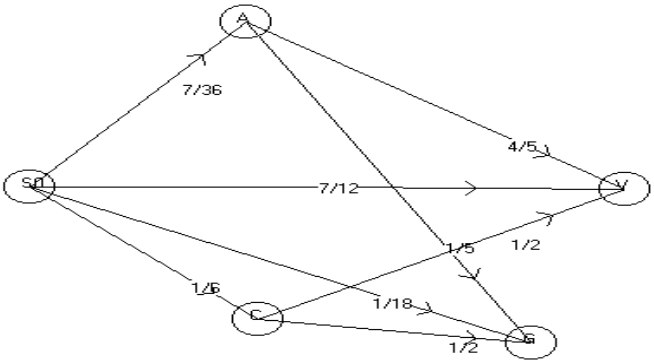
Reduzierung von Parallelkanten gemäß: $w=w_a + w_b$

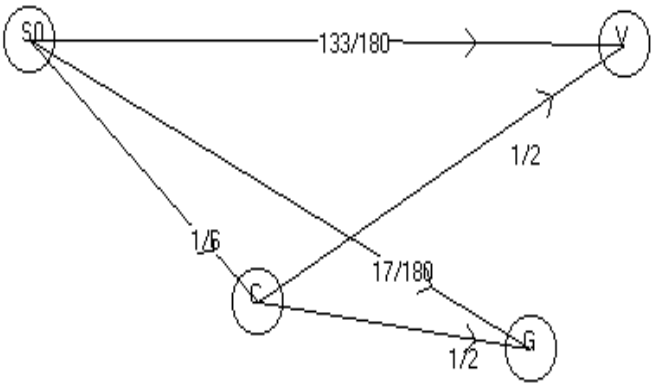
w_a und w_b Wahrscheinlichkeiten der Parallelkanten

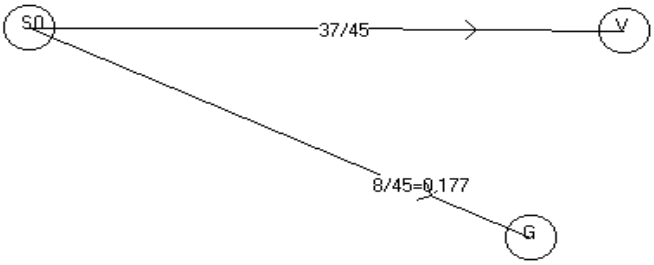
(vgl. dazu Kapitel C XII)


A
B
C
S
V
1/6
1/6
1/6
1/6
2/3
1/2
1/2
1/6
1/2
1/6
1/3
1/2


A
B
C
S
V
1/6
1/6
1/6
4/5
1/2
1/2
1/5
1/2
1/6
1/3
1/2

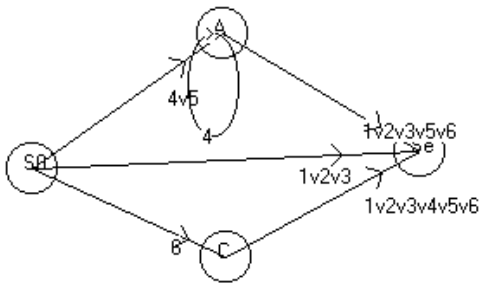






Gewinnwahrscheinlichkeit: 37/45
Verlustwahrscheinlichkeit: 8/45

d)
Reduzierter Automat (die Zustände B und C sowie V und G können zusammengefasst werden.):



(Se sollte durch einen Doppelkreis gekennzeichnet

werden.Außerdem sollte noch ein Fehlerzustand hinzugefügt werden.)

Grammatik:

1) S—>A1|A2|A3|A5|A6|C1|C2|C3|C4|C5|C6|1|2|3
2) A—>4|5|A4
3) C—>6

Der Algorithmus von Ford zur Entfernungsbestimmung

Der Algoritmus von Dijkstra gestattet nur die Entfernungsbestimmung zwischen zwei Knoten eines Graphen bei nichtnegativer Kantenbewertung.In den Kapiteln CXIII und CIV wird eine Entfernungsbestimmung bezüglich modifizierter Kosten,die auch negativ sein können,benötigt.Der folgende modifizierte Algorithmus zur Rangbestimmung von Ford (Lit 45,S.106) gestattet auch die Bestimmungen von kürzesten Pfaden bei negativ bewerteten gerichteten Graphen,sofern keine negativen Kreise vorliegen.Bei nichtnegativer Kantenbewertung ist er auch für ungerichtete Graphen anwendbar.Die Methode von TGraph befindet sich in der Unit UGraph und kann im Menü Abstand von zwei Knoten des Progamms Knotengraph (Konzeption DWK) als Möglichkeit neben dem Dijkstra-Algorithmus gewählt sowie als zur Verfügung stehende Methode in der Konzeption EWK benutzt werden.

Das Verfahren sollte im Unterricht als Vergleichsalgorithmus zum Verfahren von Dijkstra besprochen werden,wenn der Algoritmus von Busacker und Gowen angewendet werden soll.Die Grundidee dieses Algorithmus ist möglicherweise von Schülern noch einfacher als die der Dijkstra-Methode zu überblicken.

Quelltext:

```
function TGraph.BestimmeminimalenPfad(Kno1,Kno2:TKnoten;Wert:TWert):TPfad;
label Endproc;
var MomentaneKantenliste,KaWeg:TKantenliste;
    Index,Index1:Integer;
    Di,Dj,D:Extended;
    Kno,ZKno:TKnoten;
    Ka:TKante;
    Negativ:boolean;

function BesuchtMarkierung:Boolean;                                    (2)
var Zaehl:Integer;
begin
  BesuchtMarkierung:=false;
  if not Knotenliste.leer then
  for Zaehl:=0 to Knotenliste.Anzahl-1 do
  if Knotenliste.Knoten(Zaehl).Besucht
  then
    BesuchtMarkierung:=true;
end;

begin
  Pfadlistenloeschen;
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      Knotenliste.Knoten(Index).Pfadliste.AmEndeAnfuegen(TGraph.Create);
  Negativ:=false;
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
      if StringtoReal(Kantenliste.Kante(Index).Wert)<0 then Negativ:=true;
  if Negativ and (AnzahlungerichteteKanten>0)
  then
  begin
    ShowMessage('Der Graph enthält ungerichtete und negativ bewertete Kanten!Fehler!');
    goto Endproc;
  end;
  LoescheKnotenbesucht;
  if not Kno1.AusgehendeKantenliste.Leer
  then
  for Index:=0 to Kno1.AusgehendeKantenliste.Anzahl-1 do               (1)
  begin
    Ka:=Kno1.AusgehendeKantenliste.Kante(Index);
    ZKno:=Ka.Zielknoten(Kno1);
    if (not Ka.KanteistSchlinge)
    then
    begin
      Ka.Pfadrichtung:=ZKno;
      MomentaneKantenliste:=TKantenliste.Create;
```

```
      MomentaneKantenliste.AmEndeAnfuegen(Ka);Di:=ZKno.Pfadliste.Pfad(0).Pfadsumme(Wert);
      if (not ZKno.Besucht) or (Di>MomentaneKantenliste.WertsummederElemente(Wert) then
      begin ZKno.LoeschePfad;ZKno.Pfadliste.AmEndeAnfuegen(MomentaneKantenliste.Graph);end;
      ZKno.Besucht:=true;
    end;
  end;
  while Besuchtmarkierung do                                          (2)
  begin
    if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do                           (3)
    begin
      Application.ProcessMessages;
      if Abbruch then goto Endproc;
      Kno:=Knotenliste.Knoten(Index);                                 (3)
      if Kno.Besucht then                                             (3)
      begin
        Di:=Kno.Pfadliste.Pfad(0).Pfadsumme(Wert);
        if not Kno.AusgehendeKantenliste.leer then
        for Index1:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do        (3)
        begin
          MomentaneKantenliste:=TGraph(Kno.Pfadliste.Pfad(0)).Kantenliste.Kopie;
          if MomentaneKantenliste.Anzahl>AnzahlKanten then
          begin
            ShowMessage('Pfad länger als Kantenzahl!Negativer Kreis!');
            Kno2.LoeschePfad;
            goto Endproc;
          end;
          Ka:=Kno.AusgehendeKantenliste.Kante(Index1);                (3)
          ZKno:=Ka.Zielknoten(Kno);                                   (3)
          if (not Ka.KanteistSchlinge) and (ZKno<>Kno1)
          then
          begin
            KaWeg:=TKantenliste.Create;
            KaWeg.AmEndeAnfuegen(Ka);
            D:=KaWeg.WertsummederElemente(Wert);                      (4)
            KaWeg.Free;
            Ka.Pfadrichtung:=ZKno;
            Dj:=ZKno.Pfadliste.Pfad(0).Pfadsumme(Wert);               (4)
            if (Dj>Di+D) or (TGraph(ZKno.Pfadliste.Pfad(0)).Leer) then (4)
            begin
              ZKno.Besucht:=true;Ka.Pfadrichtung:=ZKno;                                    (5)
              MomentaneKantenliste.AmEndeAnfuegen(Ka);                (4)
              ZKno.LoeschePfad;
              ZKno.Pfadliste.AmEndeAnfuegen(MomentaneKantenliste.Graph.Kopie); (4)
            end
          end;
        end;
      Kno.Besucht:=false;                                             (6)
      end;
    end;
  end;
  Endproc:
  if not Kno2.Pfadliste.leer
  then
    BestimmeminimalenPfad:=Kno2.Pfadliste.Pfad(0)                     (7)
  else
    BestimmeminimalenPfad:=TPfad(TGraph.create);
MomentaneKantenliste.Free;
MomentaneKantenliste:=nil;
end;
```

Erläuterung des Quelltextes:

Zunächst werden bei 1) alle Kanten zu den Nachbarknoten des Startknoten Kno1 als Pfade in den Pfadlisten der Nachbarknoten gespeichert und die Nachbarknoten als besucht markiert.Bei 2) wird überprüft,ob noch Knoten mit einer Besucht-Markierung im Graphen vorhanden sind.Solange dies der Fall ist,werden zu den aktuell als besucht markierten Knoten Kno die Kanten Ka zu den Nachbarknoten gesucht (3).Wenn entweder kein Pfad in den Pfad-

listen der Zielknoten ZKno dieser Kanten (d.h. sie wurden noch nicht aufgesucht) oder aber ein Pfad,dessen Länge gemäß der Bewertung Wert größer als der Pfad,der über den Knoten Kno und die Kante Ka führt,gespeichert ist,wird der Pfad über den Knoten Kno zuzüglich der Kante Ka in der Pfadliste des Zielknotens ZKno gespeichert (4).Der Zielknoten wird als besucht markiert (5) und nachdem alle Zielknoten der Kanten des Knoten Kno auf diese Weise untersucht wurden,wird die Besucht-Markierung des Knotens Kno gelöscht (6).
Nachdem als Abbruchbedingung bei allen Knoten des Graphen die Besucht-Markierung gelöscht ist,befindet sich der kürzeste Pfad von Kno1 zum Ziel dem Knoten Kno2 in dessen Pfadliste (7).(Auch alle anderen Knoten des Graphen enthalten in ihren Pfadlisten den minimalen Pfad von Kno1 zu Ihnen.Dies kann in weiteren von einem Anwender (z.B. Schüler) erstellten Algorithmen weiter ausgenutzt werden.Deshalb werden die Pfadlisten am Ende des Algorithmus nicht gelöscht.)

5)Kurzbeschreibungen und Quelltext von ausgewählten Methoden des Unterrichtsprojekts CAK

Quellcode mit Kurzerläuterungen:

Zunächst werden die Methoden der Unit UListC aufgeführt,die fertig zur Verfügung gestellt wird,da die in ihr enthaltenden Methoden (zum größten Teil) aus dem Projekt Liste als Objekt bekannt sind.Auf die Elemente der Liste kann wie gewohnt mittels Element(Index) zugegriffen werden.

Unit UListC:

Die Unit enthält folgenden Interfaceteil:

```
type

  TWert=function(Ob:TObject):Extended;
  TElement=class;

  TListe=class(Tlist)
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    function Element(Index:Integer):TElement;
    property Items[Index:Integer]:TElement read Element;
    procedure AmEndeanfuegen(Ob:TObject);
    procedure AmAnfanganfuegen(Ob:TObject);
    procedure AmAnfangloeschen(var Ob:TObject);
    procedure AmEndeloeschen(var Ob:TObject);
    procedure LoescheElement(Ob:TObject);
    procedure Loeschen;
    function Anzahl:Integer;
    function WertSummederElemente(Wert:TWert):Extended;
    function WertproduktderElemente(Wert:TWert):Extended;
    function Leer:Boolean;
  end;

  TElement=class(TListe)
  public
    constructor Create;
    procedure Free;
  end;

{Functionen zur Datenkonvertierung)

  function StringtoReal(S:string):Extended;
  function RealtoString(R:Extended):string;
  function IntegertoString(I:Integer):string;
```

```
  function StringtoInteger(S:string):Integer;
```

Unit UGraphC:

Constructor von TKantenliste:

```
constructor TKantenliste.Create;
begin
  inherited Create;
end;
```

Löscht eine Instanz von TKantenliste aus dem Speicher:

```
procedure TKantenliste.Free;
begin
  inherited Free;
end;
```

Löscht eine Instanz von TKantenliste einschließlich der Kanten aus dem Speicher:

```
procedure TKantenliste.Freeall;
var Index:Integer;
    Ka:TKante;
begin
  if not Leer then
    for Index:=0 to self.Count-1 do
    begin
      Ka:=Kante(Index);
      Ka.Free;
      Ka:=nil;
    end;
  inherited Free;
end;
```

Kante der Liste mit der Nummer Index:

```
function TKantenliste.Kante(Index:Integer):TKante;
begin
  if (Index<=Self.Anzahl-1)and(Index>=0)
  then
    result:=TKante(Tlist(self).Items[Index])
   else
    Writeln('Fehler! Listenindex außerhalb des zulässigen Be-
reichs');
end;
```

Property-Methode zwecks Zugriff auf den Anfangsknoten der Kante:

```
procedure TKante.SetzeAnfangsknoten(Kno:TKnoten);
begin
  Anfangsknoten_:=Kno;
end;
```

Property-Methode zwecks Zugriff auf den Anfangsknoten der Kante:

```
function TKante.WelcherAnfangsknoten:TKnoten;
begin
  WelcherAnfangsknoten:=Anfangsknoten_;
end;
```

Property-Methode zwecks Zugriff auf den Endknoten der Kante:

```
procedure  TKante.SetzeEndknoten(Kno:TKnoten);
begin
  Endknoten_:=Kno;
end;
```

Poperty-Methode zwecks Zugriff auf den Endknoten der Kante:

```
function TKante.WelcherEndknoten:TKnoten;
begin
  WelcherEndknoten:=Endknoten_;
end;
```

Property-Methode zwecks Zugriff auf die Pfadrichtung der Kante:
(Die Pfadrichtung ist die Durchlaufrichtung,angegeben durch einen der Randknoten der Kante.)

```
procedure  TKante.SetzePfadrichtung(Kno:TKnoten);
begin
  Pfadrichtung_:=Kno;
end;
```

Property_Methode zwecks Zugriff auf die Pfadrichtung der Kante:
(Die Pfadrichtung ist die Durchlaufrichtung,angegeben durch einen der Rndknoten der Kante.)

```
function TKante.WelchePfadrichtung:TKnoten;
begin
  WelchePfadrichtung:=Pfadrichtung_;
end;
```

Property-Methode zwecks Zugriff auf die Gerichtet-Eigenschaft der Kante:

```
procedure TKante.Setzegerichtet(G:Boolean);
begin
```

```
  Gerichtet_:=G;
end;
```

Property-Methode zwecks Zugriff auf die Gerichtet-Eigenschaft der Kante:

```
function TKante.Istgerichtet:Boolean;
begin
  Istgerichtet:=Gerichtet_;
end;
```

Property-Methode zwecks Zugriff auf die Besucht-Markierung der Kante:

```
procedure TKante.SetzeBesucht(B:Boolean);
begin
  Besucht_:=B;
end;
```

Propertymethode zwecks Zugriff auf die Besucht-Markierung der Kante:

```
function TKante.Istbesucht:Boolean;
begin
  Istbesucht:=Besucht_;
end;
```

Constructor für TKante:

```
constructor TKante.Create;
begin
  inherited Create;
  Anfangsknoten_:=nil;
  EndKnoten_:=nil;
  Pfadrichtung_:=nil;
  Gerichtet_:=true;
  Besucht_:=false;
end;
```

Löscht eine Instanz von TKante aus dem Speicher:

```
procedure TKante.Free;
begin
  inherited Free;
end;
```

Löscht eine Instanz von TKante einschließlich der Randknoten aus dem Speicher:

```
procedure TKante.Freeall;
begin
  Anfangsknoten_.Free;
  Anfangsknoten_:=nil;
  Endknoten_.Free;
  Endknoten_:=nil;
  inherited Free;
end;
```

Gibt als Resultat den Zielknoten als zum Knoten Kno entgegenge-
setztem Randknoten der Kante (oder den Endknoten) zurück:

```
function TKante.Zielknoten(Kno:TKnoten):TKnoten;
begin
  Zielknoten:=Endknoten;
  if Kno=Endknoten then Zielknoten:=Anfangsknoten;
end;
```

Gibt als Resultat den Knoten Kno als Randknoten der Kante oder
den Anfangsknoten der Kante zurück:

```
function TKante.Quellknoten(Kno:TKnoten):TKnoten;
begin
  Quellknoten:=Anfangsknoten;
  if Kno=Endknoten then Quellknoten:=Endknoten;
end;
```

Gibt als Resultat zurück,ob die Kante eine Schlinge ist:

```
function TKante.KanteistSchlinge:Boolean;
begin
  KanteistSchlinge:=(EndKnoten=Anfangsknoten);
end;
```

Constructor für TKnotenliste:

```
constructor TKnotenliste.Create;
begin
  inherited Create;
end;
```

Löscht eine Instanz von TKnotenliste aus dem Speicher:

```
procedure TKnotenliste.Free;
begin
  inherited Free;
end;
```

Löscht eine Instanz von TKnotenliste einschließlich ihrer Knoten
aus dem Speicher:

```
procedure TKnotenliste.Freeall;
var Index:Integer;
    Kno:TKnoten;
begin
  if not Leer then
  for Index:=0 to Anzahl-1 do
  begin
    Kno:=self.Knoten(Index);
    Kno.Free;
    Kno:=nil;
  end;
  inherited Free
end;
```

Gibt den Knoten der Knotenliste mit der Nummer Index zurück:

```
function TKnotenliste.Knoten(Index:Integer):TKnoten;
begin
  if (Index<=Self.Anzahl-1)and (Index>=0)
  then
    result:=TKnoten(Tlist(self).Items[Index])
   else
    Writeln(‘Fehler! Listenindex außerhalb des zulässigen Be-
reichs’);
end;
```

Property-Methode zwecks Zugriff auf die Eigenschaft Graph des Knotens:

```
function TKnoten.WelcherGraph:TGraph;
begin
  WelcherGraph:=Graph_;
end;
```

Property-Methode zwecks Zugriff auf die Eigenschaft Graph des Knotens:

```
procedure TKnoten.SetzeGraph(G:TGraph);
begin
  Graph_:=G;
end;
```

Property-Methode zwecks Zugriff auf die EingehendeKantenliste des Knotens:

```
function TKnoten.WelcheEingehendeKantenliste:TKantenliste;
begin
  WelcheEingehendeKantenliste:=EingehendeKantenliste_;
end;
```

Property-Methode zwecks Zugriff auf die EingehendeKantenliste des Knotens:

```
procedure TKnoten.SetzeEingehendeKantenliste(Kl:TKantenliste);
begin
  EingehendeKantenliste_:=Kl;
end;
```

Property-Methode zwecks Zugriff auf die AusgehendeKantenliste des Knotens:

```
function TKnoten.WelcheAusgehendeKantenliste:TKantenliste;
begin
  WelcheAusgehendeKantenliste:=AusgehendeKantenliste_;
end;
```

Property-Methode zwecks Zugriff auf die AusgehendeKantenliste des Knotens:

```
procedure TKnoten.SetzeAusgehendeKantenliste(Kl:TKantenliste);
begin
  AusgehendeKantenliste_:=Kl;
end;
```

Property-Methode für die Besucht-Markierung des Knotens:

```
function TKnoten.Istbesucht:Boolean;
begin
  Istbesucht:=besucht_;
end;
```

Property-Methode für die Besucht-Markierung des Knotens:

```
procedure TKnoten.Setzebesucht(B:Boolean);
begin
  besucht_:=B;
end;
```

Constructor für TKnoten:

```
constructor TKnoten.Create;
  begin
  Graph_:=nil;
  AusgehendeKantenliste_:=TKantenliste.create;
  EingehendeKantenliste_:=TKantenliste.create;
  Besucht_:=false;
end;
```

Löscht eine Instanz von TKnoten aus dem Speicher:

```
procedure TKnoten.Free;
begin
  AusgehendeKantenliste_.Free;
  AusgehendeKantenliste_:=nil;
  EingehendeKantenliste_.Free;
  EingehendeKantenliste_:=nil;
  inherited Free;
end;
```

Löscht eine Instanz von TKnoten einschließlich der mit dem Kno-
ten verbundenen Kanten aus dem Speicher:

```
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;
```

Property-Methode zwecks Zugriff auf die Knotenliste des Graphen:

```
function TGraph.WelcheKnotenliste:TKnotenliste;
begin
  WelcheKnotenliste:=Knotenliste_;
end;
```

Property-Methode zwecks Zugriff auf die Knotenliste des Graphen:

```
procedure TGraph.SetzeKnotenliste(Kl:TKnotenliste);
begin
  Knotenliste_:=Kl;
end;
```

Property-Methode zwecks Zugriff auf die Kantenliste des Graphen:

```
function TGraph.WelcheKantenliste:TKantenliste;
begin
  WelcheKantenliste:=Kantenliste_;
end;
```

Property-Methode zwecks Zugriff auf die Kantenliste des Graphen:

```
procedure TGraph.SetzeKantenliste(Kl:TKantenliste);
begin
  Kantenliste_:=Kl;
end;
```

Constructor für TGraph:

```
constructor TGraph.Create;
begin
  inherited Create;
  Knotenliste_:=TKnotenliste.create;;
  Kantenliste_:=TKantenliste.create;
end;
```

Löscht eine Instanz von TGraph aus dem Speicher:

```
procedure TGraph.Free;
begin
  Knotenliste_.Free;
  Knotenliste:=nil;
  Kantenliste_.Free;
  Kantenliste_:=nil;
  inherited Free;
end;
```

Löscht eien Instanz von TGraph samt allen Knoten und Kanten aus dem Speicher:

```
procedure TGraph.Freeall;
begin
  Kantenliste_.Freeall;
  Kantenliste_:=nil;
  Knotenliste_.Freeall;
  Knotenliste_:=nil;
  inherited Free;
end;
```

Gibt als Resultat zurück,ob der Graph ohne Knoten (und Kanten) ist:

```
function TGraph.Leer:Boolean;
begin
```

```
  Leer:=self.Knotenliste.Leer;
end;
```

Fügt den Knoten Kno in den Graph (Knotenliste) ein:

```
procedure TGraph.KnotenEinfuegen(Kno:TKnoten);
begin
  Knotenliste.AmEndeanfuegen(Kno);
  Kno.Graph:=self;
end;
```

Löscht den Knoten Kno aus dem Graph,falls dort vorhanden (dabei müssen auch die mit ihm verbundenen Kanten gelöscht werden):

```
procedure TGraph.Knotenloeschen(Kno:TKnoten);
var Index:Integer;
    Ka:TKante;
begin
  if not Kno.AusgehendeKantenliste.Leer then
    for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
    begin
      Ka:=Kno.AusgehendeKantenliste.Kante(Index);
      if not Ka.KanteistSchlinge then
      begin
      Ka.Zielknoten(Kno).EingehendeKantenliste.LoescheElement(Ka);
*        if not Ka.Gerichtet then
        Ka.Zielknoten(Kno).AusgehendeKantenliste.
        LoescheElement(Ka);
      end;
      Kantenliste.LoescheElement(Ka);
    end;
  if not Kno.eingehendeKantenliste.Leer then
    for Index:=0 to Kno.EingehendeKantenliste.Anzahl-1 do
    begin
      Ka:=Kno.EingehendeKantenliste.Kante(Index);
      if not Ka.KanteistSchlinge then
      begin
      Ka.Zielknoten(Kno).AusgehendeKantenliste.LoescheElement(Ka);
        if not Ka.gerichtet then Ka.Zielknoten(Kno).
        EingehendeKantenliste.loescheElement(Ka);
      end;
      Kantenliste.LoescheElement(Ka);
    end;
  Knotenliste.LoescheElement(Kno);
  Kno.Freeall;
  Kno:=nil;
end;
```

Fügt die Kante Ka zwischen Anfangsknoten und Endknoten (Anfangs-knoten und Endknoten müssen schon im Graph vorhanden sein) in

den Graph ein,wobei gerichtet die Eigenschaft gerichtet bestimmt (Die Kante muß dabei in die Kantenlisten von Anfangs-und Endknoten und die Kantenliste des Graphen eingefügt werden.)

```
procedure TGraph.KanteEinfuegen (Ka:Tkante;
Anfangsknoten,EndKnoten:TKnoten;gerichtet:Boolean);
label Ende;
begin
  if (Anfangsknoten=nil) or (Endknoten=nil)or(Ka=nil) then goto
Ende;
  Ka.Anfangsknoten:=Anfangsknoten;
  Ka.Endknoten:=Endknoten;
  Anfangsknoten.AusgehendeKantenliste.AmEndeanfuegen(Ka);
  if Gerichtet then
  begin
    EndKnoten.EingehendeKantenliste.AmEndeanfuegen(Ka);
    Ka.Gerichtet:=true;
  end
  else
  begin
    Ka.Gerichtet:=false;
    Anfangsknoten.EingehendeKantenliste.AmEndeanfuegen(Ka);
    if  Anfangsknoten<>Endknoten then
    begin
      EndKnoten.AusgehendeKantenliste.AmEndeanfuegen(Ka);
      Endknoten.EingehendeKantenliste.AmEndeanfuegen(Ka);
    end;
  end;
  Ka.Anfangsknoten:=Anfangsknoten;
  Ka.EndKnoten:=EndKnoten;
  Kantenliste.amEndeanfuegen(Ka);
  Ende:
end;
```

Löscht die Kante Ka aus dem Graph,falls vorhanden (Die Verweise auf die Kante müssen dabei aus den Kantenlisten von Anfangs-und Endknoten und aus der Kantenliste des Graphen gelöscht werden.)

```
procedure TGraph.Kanteloeschen(Ka:TKante);
begin
  Kantenliste.LoescheElement(Ka);
  Ka.Anfangsknoten.AusgehendeKantenliste.LoescheElement(Ka);
  if Ka.Gerichtet
  then
    Ka.EndKnoten.EingehendeKantenliste.LoescheElement(Ka)
  else
  begin
    if Ka.Anfangsknoten<>Ka.Endknoten then
    begin
      Ka.EndKnoten.AusgehendeKantenliste.LoescheElement(Ka);
```

```
      Ka.EndKnoten.EingehendeKantenliste.LoescheElement(Ka);
    end;
    Ka.Anfangsknoten.EingehendeKantenliste.LoescheElement(Ka);
  end;
  Ka.Free;
  Ka:=nil;
end;
```

Löscht die Besucht-Markierung aller Kanten des Graphen:

```
procedure TGraph.LoescheKantenbesucht;
var Index:Integer;
begin
  for Index:=0 to Kantenliste.Anzahl-1 do
    Kantenliste.Kante(Index).Besucht:=false;
end;
```

Löscht die Besucht-Markierung aller Knoten des Graphen:

```
procedure TGraph.LoescheKnotenbesucht;
var Index:Integer;
begin
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      Knotenliste.Knoten(Index).Besucht:=false;
end;
```

Gibt die Anzahl der Kanten (ohne Schlingen) des Graphen zurück:

```
function TGraph.AnzahlKanten:Integer;
begin
  AnzahlKanten:=Kantenliste.Anzahl-AnzahlSchlingen;
end;
```

Gibt die Anzahl der Schlingen des Graphen zurück:

```
function TGraph.AnzahlSchlingen:Integer;
Var Graphschlingen:Integer;
    Index:Integer;
    Ka:TKante;

begin
  Graphschlingen:=0;
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
    begin
      Ka:=Kantenliste.Kante(Index);
      if Ka.KanteistSchlinge then
        Graphschlingen:=Graphschlingen+1;
    end;
```

```
  AnzahlSchlingen:=Graphschlingen;
end;
```

Gibt die Anzahl der Kanten des Graphen einschließlich Schlingen zurück:

```
function TGraph.AnzahlKantenmitSchlingen: Integer;
begin
  AnzahlKantenmitSchlingen:=AnzahlKanten+AnzahlSchlingen;
end;
```

Gibt die Anzahl der Knoten des Graphen zurück:

```
function TGraph.AnzahlKnoten:Integer;
begin
  AnzahlKnoten:=Knotenliste.Anzahl;
end;
```

Unit UInhgrphC

Constructor für TInhaltsknoten:

```
constructor TInhaltsknoten.Create;
begin
  inherited Create;
  Inhalt_:=’ ‘;
end;
```

Löscht eine Instanz von TInhaltsknoten aus dem Speicher:

```
procedure TInhaltsknoten.Free;
begin
  inherited Free;
end;
```

Löscht eine Instanz von TInhaltsknoten einschließlich der mit ihm verbundenen Knaten aus dem Speicher:

```
procedure TInhaltsknoten.Freeall;
begin
  inherited Freeall;
end;
```

Virtuelle Property-Methode zwecks Zugriff auf die Eigenschaft Wert eines Knotens
(Zugriff auf Inhalt_):

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

```
  Inhalt_:=S;
end;
```

Virtuelle Property-Methode zwecks Zugriff auf die Eigenschaft Wert eines Knotens
(Zugriff auf Inhalt_):

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

Liest den Wert (Eigenschaft) eines Knotens von der Tastatur als string ein:

```
procedure TInhaltsknoten.Knotenwerteinlesen;
var S:string;
begin
   Write('Knotenwert eingeben: ');
   Readln(S);
   Wert:=S;
end;
```

Constructor für TInhaltskante:

```
constructor TInhaltskante.Create;
begin
  inherited Create;
  Inhalt_:=' ';
end;
```

Löscht eine Instanz von TInhaltskante aus dem Speicher:

```
procedure TInhaltskante.Free;
begin
  inherited Free;
end;
```

Löscht eine Insatanz von TInhaltskante einschließlich der Rand-knoten aus dem Speicher.

```
procedure TInhaltskante.Freeall;
begin
  inherited Freeall;
end;
```

Virtuelle Property-Methode zwecks Zugriff auf die Eigenschaft Wert einer Kante
(Zugriff auf Inhalt_):

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

Virtuelle Property-Methode zwecks Zugriff auf die Eigenschaft Wert einer Kante
(Zugriff auf Inhalt_):

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

Liest den Wert (Eigenschaft) einer Kante von der Tastatur als string ein.

```
procedure TInhaltskante.Kantenwerteinlesen;
var S:string;
begin
   Write('Kantenwert eingeben: ');
   Readln(S);
   Wert:=S;
end;
```

Gibt die Randknotenwerte der Kanten einer Kantenliste (Pfad in einem Graph) gemäß der Reihenfolge der Pfadrichtung auf dem Bildschirm (mittels Write/Writeln) einschließlich der Wertsumme und des Werprodukts der Kantenwerte gemäß der Kantenbewertung Bewertung aus:

```
procedure TInhaltskante.Listenausgabe;
var Zaehl:Integer;
begin
  if not leer then
  begin
    Write(TInhaltsknoten(Kante(0).Quellknoten(self.Kante(0).
    Pfadrichtung)).Wert,' ');
    for Zaehl:=0 to Anzahl-1 do
      Write(TInhaltsknoten(Kante(Zaehl).Pfadrichtung).Wert,' ');
    Write('Summe: ',WertsummederElemente(Bewertung):6:2,' ');
    Write('Produkt: ',WertproduktderElemente(Bewertung):6:2);
    Writeln;
  end;
end;
```

Constructor für TInhaltsgraph:

```
constructor TInhaltsgraph.Create;
begin
```

```
  inherited Create;
end;
```

Löscht eine Instanz von TInhaltsgraph aus dem Speicher:

```
procedure TInhaltsgraph.Free;
begin
  inherited Free;
end;
```

Löscht eine Instanz von TInhaltsgraph einschließlich seiner Knoten und Kanten aus dem Speicher:

```
procedure TInhaltsgraph.Freeall;
begin
  inherited Freeall;
end;
```

Gibt,falls existent,zu einem Knoten Kno mit bestimmten Wert den in der Knotenliste des Graphen zuerst gespeicherten Knoten mit demselben Wert zurück (d.h. bestimmt zu einem beliebig vorgegebenen Knoten Kno einen Knoten des Graphen mit demselben Wert):

```
function
TInhaltsgraph.Graphknoten(Kno:TInhaltsknoten):TInhaltsknoten;
var Index:Integer;
    Kn:TInhaltsknoten;
begin
  Graphknoten:=nil;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kn:=TInhaltsknoten(Knotenliste.Knoten(Index));
      if Kno.Wert=Kn.Wert
      then
      begin
        Graphknoten:=Kn;
        Kn.Graph:=self;
        exit;
      end
    end;
end;
```

Erzeugt einen Knoten vom Typ TInhaltsknoten,liest seinen Wert von der Tastatur ein und fügt den Knoten in den Graph ein:

```
procedure TInhaltsgraph.Knotenerzeugenundeinfuegen;
var Kno:TInhaltsknoten;
begin
  Kno:=TInhaltsknoten.create;
```

```
  Kno.KnotenWerteinlesen;
  self.Knoteneinfuegen(Kno);
end;
```

Löscht,falls existent,einen Knoten vom Typ TInhaltsknoten aus dem Graph,dessen Wert zuvor von der Tastatur eingelesen wurde:

```
procedure TInhaltsgraph.KnotenausGraphloeschen;
var W:string;
    Index:Integer;
    Kno:TKnoten;
    Gefunden:Boolean;
begin
  Writeln('Knoten löschen');
  Writeln;
  Write('Knotenwert eingeben: ');
  Readln(W);
  Gefunden:=false;
  if not Knotenliste.leer then
  begin
    Index:=0;
    repeat
      Kno:=Knotenliste.Knoten(Index);
      if TInhaltsknoten(Kno).Wert=W then
      begin
        Gefunden:=true;
        Knotenloeschen(Kno);
      end;
      Index:=Index+1
    until Gefunden or (Index>Knotenliste.Anzahl-1);
    if Gefunden then Writeln('Knoten ',W,' gelöscht!') else
Writeln('Knoten ',W,' nicht im Graph!');
  end;
end;
```

Fügt eine Kante Ka vom Typ TInhaltskante mit dem Wert gerichtet für die Eigenschaft gerichtet zwischen den Knoten Kno1 und Kno2 des Graphen vom Typ TInhaltsknoten ein.Falls die Knoten noch nicht im Graph vorhanden sind,werden sie in den Graph eingefügt.

```
procedure
TInhaltsgraph.FuegeKanteein(Kno1,Kno2:TInhaltsknoten;Gerichtet:Boolean;
Ka:TInhaltskante);
label Endschleife1,Endschleife2;
var Richtung:Boolean;
    Pos1,Pos2,Index:Integer;
    Knoa,Knoe:TKnoten;
    Kno:TInhaltsknoten;
begin
  if Gerichtet then Richtung:=true else Richtung:=false;
```

```
  Pos1:=-1;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TInhaltsknoten(Knotenliste.Knoten(Index));
      if Kno.Wert=Kno1.Wert then
      begin
        Pos1:=Index;
        goto Endschleife1;
      end;
    end;
  Endschleife1:
  Pos2:=-1;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TInhaltsknoten(Knotenliste.Knoten(Index));
      if Kno.Wert=Kno2.Wert then
      begin
        Pos2:=Index;
        goto Endschleife2;
      end;
    end;
  Endschleife2:
  if Pos1>=0
  then
    Knoa:=Knotenliste.Knoten(Pos1)
  else
  begin
    Knoteneinfuegen(Kno1);
    Knoa:=Kno1;
  end;
  if Pos2>=0
  then
    Knoe:=self.Knotenliste.Knoten(Pos2)
  else
  begin
    Knoteneinfuegen(Kno2);
    Knoe:=Kno2;
  end;
  if (Knoa<>nil)and (Knoe<>nil)and (Ka<>nil)
  then Kanteeinfuegen(Ka,Knoa,Knoe,Richtung);
end;
```

Erzeugt eine Kante,liest ihren Wert von der Tastatur ein,erzeugt die Randknoten der Kante,liest ihre Werte von der Tastatur ein,liest die Eigenschaft gerichtet der Kante von der Tastatur ein und fügt die Kante in den Graphen mittels der Methode FuegeKanteein ein:

```
procedure TInhaltsgraph.Kanteerzeugenundeinfuegen;
var Kw1,Kw2,W,Antwort:string;
    Index:Integer;
    Gerichtet:Boolean;
    Kno1,Kno2:TInhaltsknoten;
    Ka:TInhaltskante;
begin
  Writeln('Kante erzeugen:');
  Write('Anfangsknotenwert: ');
  Readln(Kw1);
  Write('Endknotenwert: ');
  Readln(Kw2);
  Write('Kantenwert: ');
  Readln(W);
  Write('Gerichtetet? (j/n): ');
  Readln(Antwort);
  if (Antwort='j') or (Antwort='J') then Gerichtet:=true else
Gerichtet:=false;
  Kno1:=TInhaltsknoten.Create;
  Kno1.Wert:=Kw1;
  Kno2:=TInhaltsknoten.Create;
  Kno2.Wert:=kw2;
  Ka:=TInhaltskante.Create;
  Ka.Wert:=w;
  self.FuegeKanteein(Kno1,Kno2,Gerichtet,Ka);
end;
```

Löscht,falls existent,eine Kante Ka aus dem Graph,deren Rand-knoten Kno1 und Kno2 sind:

```
procedure
TInhaltsgraph.LoescheKante(Kno1,Kno2:TInhaltsknoten;Ka:TInhaltskante);
var Index:Integer;
    kant:TKante;
begin
  if not Kantenliste.leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
    begin
      Kant:=Self.Kantenliste.Kante(Index);
      if ((Kant.Anfangsknoten.Wert=Kno1.Wert) and
(Kant.Endknoten.Wert=Kno2.Wert)and
        (Kant.Wert=Ka.Wert))
      then
      begin
        Kanteloeschen(Kant);
        exit;
      end;
    end;
end;
```

Liest einen Kantenwert,zwei Randknotenwerte und den Wert für die Eigenschaft gerichtet von der Tastatur ein und löscht eine entsprechende Kante aus dem Graph,falls sie existiert:

```
procedure TInhaltsgraph.KanteausGraphloeschen;
var Kw1,Kw2,W,Antwort:string;
    Index:Integer;
    Kno1,Kno2:TInhaltsknoten;
    Ka:TInhaltskante;
begin
  Writeln('Kante löschen:');
  Writeln;
  Write('Anfangsknotenwert: ');
  Readln(Kw1);
  Write('Endknotenwert: ');
  Readln(Kw2);
  Write('Kantenwert: ');
  Readln(W);
  Kno1:=TInhaltsknoten.Create;
  Kno1.Wert:=Kw1;
  Kno2:=TInhaltsknoten.Create;
  Kno2.Wert:=Kw2;
  Ka:=TInhaltskante.Create;
  Ka.Wert:=W;
  LoescheKante(Kno1,Kno2,Ka);
  Kno1.Freeall;
  Kno1:=nil;
  Kno2.Freeall;
  Kno2:=nil;
  Ka.Free;
  Ka:=nil;
end;
```

Gibt den Graph unter Anzeige (mittels Write/Writeln) der Knoten- und Kantenwerte auf dem Bildschirm aus:

```
procedure TInhaltsgraph.ZeigeGraph;
var Index:Integer;
    Kno:TKnoten;
    Ka:TInhaltskante;
begin
  Writeln('Anzeige des Graphen:');
  Writeln;
  Writeln('Knoten:');
  Writeln;
  if self.Knotenliste.leer then Writeln('Knotenliste leer!');
  if not self.Knotenliste.leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=Knotenliste.Knoten(Index);
```

```
      Writeln(Index,': Wert: ',Kno.Wert);
    end;
  Writeln;
  Writeln('Kanten:');
  Writeln;
  if Kantenliste.leer then Writeln('Kantenliste leer!');
  if not Kantenliste.leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
    begin
      Ka:=TInhaltskante(self.Kantenliste.Kante(Index));
      Write(Index,': Wert: ',Ka.Wert,' Anfangsknoten:',
      Ka.Anfangsknoten.Wert,'
       Endknoten: ',Ka.Endknoten.Wert);
      if TInhaltskante(Ka).gerichtet
      then Writeln(' gerichtet') else Writeln(' ungerichtet');
    end;
  Writeln;
end;
```

Die Bewertung einer Kante bzw. eines Knoten (vom Typ TInhaltskante bzw. TInhaltsknoten) ist der Wert der Kante umgewandelt in eine Real-Zahl:

```
function Bewertung (X:TObject):Extended;
begin
  if  X is TInhaltskante
  then
    Bewertung := StringtoReal(TInhaltskante(X).Wert)
  else
    begin
      if X is TInhaltsknoten
      then
        Bewertung:=StringtoReal(TInhaltsknoten(X).Wert)
      else
        Bewertung:=1;
    end;
end;
```

Unit UmathC:

Färbarkeit eines Graphen:

Constructor für TFarbCknoten:

```
constructor TFarbCknoten.Create;
begin
  inherited create;
  Farbzahl_:=0;
```

```
  Ergebnis_:=’’;
end;
```

Löscht eine Instanz von TFarbCknoten aus dem Speicher:

```
procedure TFarbCknoten.Free;
begin
  inherited Free;
end;
```

Entfernt eine Instanz von TFarbCknoten einschließlich der mit ihm verbundenen Kanten aus dem Speicher:

```
procedure TFarbCknoten.Freeall;
begin
  inherited Freeall;
end;
```

Property-Methode zwecks Zugriff auf die Farbzahl des Knotens:

```
function TFarbCknoten.WelcheFarbzahl:Integer;
begin
  WelcheFarbzahl:=Farbzahl_;
end;
```

Property-Methode zwecks Zugriff auf die Farbzahl des Knotens:

```
procedure TFarbCknoten.SetzeFarbzahl(Fa:Integer);
begin
  Farbzahl_:=Fa;
end;
```

Property-Methode zwecks Zugriff auf das Ergebnis des Knotens:

```
function TFarbCknoten.WelchesErgebnis:String;
begin
  WelchesErgebnis:=Ergebnis_;
end;
```

Property-Methode zwecks Zugriff auf das Ergebnis des Knotens:

```
procedure TFarbCknoten.SetzeErgebnis(S:string);
begin
  Ergebnis_:=S;
end;
```

Constructor für TFarbgraph:

```
constructor TFarbCgraph.Create;
begin
```

```
  inherited Create;
end;
```

Löscht eine Instanz von TFarbgraph aus dem Speicher:

```
procedure TFarbCgraph.Free;
begin
  inherited Free;
end;
```

Löscht eine Instanz von TFarbgraph einschließlich der Knoten und Kanten aus dem Speicher:

```
procedure TFarbCgraph.Freeall;
begin
  inherited Freeall;
end;
```

Setzt bei allen Knoten die Anfangsfarbe auf den Wert 0:

```
procedure TFarbCgraph.SetzebeiallenKnotenAnfangsfarbe;
var Index:Integer;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl -1 do
      TFarbCknoten(Knotenliste.Knoten(Index)).KnotenFarbe:=0;
end;
```

Diese Methode erzeugt in den Ergebnisfeldern der Knoten das anzuzeigende Ergebnis:

```
procedure TFarbCgraph.ErzeugeErgebnis;
var Index:Integer;
    Kno:TFarbCknoten;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TFarbCknoten(Knotenliste.Knoten(Index));
    Kno.SetzeErgebnis(Kno.Wert+':'+IntegertoString(Kno.Knotenfarbe));
    end;
end;
```

6)Gesamtquelltext des Programms CAK (Projekt zum Aufbau der objektorientierten Datenstruktur eines Graphen mit text-orientierter Ausgabe als Consolenanwendung)

Unit UListC:

```
unit UListc;

{$F+}

interface

uses
 Classes,Winprocs,Sysutils;

type

  TWert=function(Ob:TObject):Extended;
  TElement=class;

  TListe=class(Tlist)
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    function Element(Index:Integer):TElement;
    property Items[Index:Integer]:TElement read Element;
    procedure AmEndeanfuegen(Ob:TObject);
    procedure AmAnfanganfuegen(Ob:TObject);
    procedure AmAnfangloeschen(var Ob:TObject);
    procedure AmEndeloeschen(var Ob:TObject);
    procedure LoescheElement(Ob:TObject);
    procedure Loeschen;
    function Anzahl:Integer;
    function WertSummederElemente(Wert:TWert):Extended;
    function WertproduktderElemente(Wert:TWert):Extended;
    function Leer:Boolean;
  end;

  TElement=class(TListe)
  public
    constructor Create;
    procedure Free;
  end;


  function StringtoReal(S:string):Extended;
  function RealtoString(R:Extended):string;
  function IntegertoString(I:Integer):string;
  function StringtoInteger(S:string):Integer;
```

```
implementation

constructor TListe.Create;
begin
  inherited Create;
end;

procedure TListe.Free;
begin
  inherited Free;
end;

procedure TListe.Freeall;
var Index:Integer;
    El:TElement;
begin
  if not Leer then
  for Index:=0 to Anzahl-1 do
  begin
    El:=Element(Index);
    El.Freeall;
    El:=nil;
  end;
  Free;
end;


function TListe.Element(Index:Integer):TElement;
begin
  if (Index<=Anzahl-1)and (Index>=0)
  then
    result:=TElement(Tlist(self).Items[Index])
  else
    Writeln('Fehler! Listenindex außerhalb des zulässigen Be-
reichs');
end;


procedure TListe.AmEndeanfuegen(Ob:TObject);
begin
  Capacity:=Count;
  Add(Ob);
end;


procedure TListe.AmAnfanganfuegen(Ob:TObject);
begin
  Capacity:=self.Count;
  Insert(0,Ob);
end;
```

```
procedure TListe.AmAnfangloeschen(var Ob:TObject);
begin
  Ob:=Items[0];
  Delete(0);
end;


procedure TListe.AmEndeloeschen(var Ob:TObject);
begin
  Ob:=Items[Count-1];
  Delete(Count-1);
end;


procedure TListe.LoescheElement(Ob:TObject);
begin
  Remove(Ob);
  Pack;
end;


procedure TListe.Loeschen;
begin
  Clear;
end;

function TListe.Anzahl:Integer;
begin
  Anzahl:=Count;
end;


function TListe.WertsummederElemente(Wert:Twert):Extended;
var Index:Integer;
    Wertsumme:Extended;

  procedure SummiereWert(Ob:TObject);
  var Z:Real;
  begin
    if abs(Wertsumme+Wert(Ob))<1E40
    then
      Wertsumme:=Wertsumme+Wert(Ob)
    else
      begin
          Writeln('Wertsumme zu groß!');
        Z:=0;
        Wertsumme:=1E40/Z;
      end;
  end;
```

```
begin
  Wertsumme:=0;
  if not self.Leer then
    for Index:=0 to Anzahl-1 do
      SummiereWert(Element(Index));
  WertsummederElemente:=Wertsumme;
end;


function TListe.WertproduktderElemente(Wert:TWert):Extended;
var Index:Integer;
    Wertprodukt:Extended;

  procedure MultipliziereWert(Ob:TObject);
  var Z:Real;
  begin
    if abs(Wertprodukt*Wert(Ob))<1E40
    then
      Wertprodukt:=Wertprodukt*Wert(Ob)
    else
      begin
          Writeln('Wertprodukt zu groß!');
        Z:=0;
        Wertprodukt:=1E40/Z;
      end;
  end;


begin
  if Anzahl>0
  then
    Wertprodukt:=1
  else
    Wertprodukt:=0;
  if not Leer then
    for Index:=0 to Anzahl-1 do
      MultipliziereWert(Element(Index));
    WertproduktderElemente:=Wertprodukt;
end;


function TListe.Leer:Boolean;
begin
  Leer:=(self=nil) or (Count=0);
end;

constructor TElement.Create;
begin
  inherited Create;
end;
```

```
procedure TElement.Free;
begin
  inherited Free;
end;


function StringtoReal(S:string):Extended;
var Feld:Array[0..101] of char;
    R:Extended;
begin
  if Length(S)<101 then
  begin
    Strpcopy(Feld,S);
    Decimalseparator:='.';
    if TexttoFloat(Feld,R)
    then
      StringtoReal:=R
    else
      StringtoReal:=0;
  end
  else
    StringtoReal:=0;
end;

function RealtoString(R:Extended):string;
begin
  Decimalseparator:='.';
  Realtostring:=Floattostr(R);
end;

function IntegertoString(I:Integer):string;
begin
  Integertostring:=InttoStr(I);
end;

function StringtoInteger(S:string):Integer;
begin
  StringtoInteger:=StrtoIntDef(S,0);
end;


end.

Unit UGraphC:

unit UGraphC;
{$F+}

interface
```

```
uses
 Classes,Winprocs,Sysutils,UListc;

type

  TString=function(X:TObject):string;

  TKante  = class;

  TKnotenliste = class;

  TKnoten = class;

  TGraph  = class;

  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 Setze
  gerichtet;
  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;

  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;


implementation

constructor TKantenliste.Create;
begin
  inherited Create;
end;

procedure TKantenliste.Free;
begin
  inherited Free;
end;

procedure TKantenliste.Freeall;
var Index:Integer;
    Ka:TKante;
```

```
begin
  if not Leer then
    for Index:=0 to self.Count-1 do
    begin
      Ka:=Kante(Index);
      Ka.Free;
    end;
  inherited Free;
end;


function TKantenliste.Kante(Index:Integer):TKante;
begin
  if (Index<=Self.Anzahl-1)and(Index>=0)
  then
    result:=TKante(Tlist(self).Items[Index])
   else
    Writeln('Fehler! Listenindex außerhalb des zulässigen Be
     reichs');
end;

procedure TKante.SetzeAnfangsknoten(Kno:TKnoten);
begin
  Anfangsknoten_:=Kno;
end;

function TKante.WelcherAnfangsknoten:TKnoten;
begin
  WelcherAnfangsknoten:=Anfangsknoten_;
end;

procedure  TKante.SetzeEndknoten(Kno:TKnoten);
begin
  Endknoten_:=Kno;
end;

function TKante.WelcherEndknoten:TKnoten;
begin
  WelcherEndknoten:=Endknoten_;
end;

procedure  TKante.SetzePfadrichtung(Kno:TKnoten);
begin
  Pfadrichtung_:=Kno;
end;

function TKante.WelchePfadrichtung:TKnoten;
begin
  WelchePfadrichtung:=Pfadrichtung_;
end;
```

```
procedure TKante.Setzegerichtet(G:Boolean);
begin
  Gerichtet_:=G;
end;

function TKante.Istgerichtet:Boolean;
begin
  Istgerichtet:=Gerichtet_;
end;

procedure TKante.SetzeBesucht(B:Boolean);
begin
  Besucht_:=B;
end;

function TKante.Istbesucht:Boolean;
begin
  Istbesucht:=Besucht_;
end;

constructor TKante.Create;
begin
  inherited Create;
  Anfangsknoten_:=nil;
  EndKnoten_:=nil;
  Pfadrichtung_:=nil;
  Gerichtet_:=true;
  Besucht_:=false;
end;

procedure TKante.Free;
begin
  inherited Free;
end;

procedure TKante.Freeall;
begin
  Anfangsknoten_.Free;
  Anfangsknoten_:=nil;
  Endknoten_.Free;
  Endknoten_:=nil;
  inherited Free;
end;


function TKante.Zielknoten(Kno:TKnoten):TKnoten;
begin
  Zielknoten:=Endknoten;
  if Kno=Endknoten then Zielknoten:=Anfangsknoten;
end;
```

```
function TKante.Quellknoten(Kno:TKnoten):TKnoten;
begin
  Quellknoten:=Anfangsknoten;
  if Kno=Endknoten then Quellknoten:=Endknoten;
end;


function TKante.KanteistSchlinge:Boolean;
begin
  KanteistSchlinge:=(EndKnoten=Anfangsknoten);
end;

constructor TKnotenliste.Create;
begin
  inherited Create;
end;

procedure TKnotenliste.Free;
begin
  inherited Free;
end;

procedure TKnotenliste.Freeall;
var Index:Integer;
    Kno:TKnoten;
begin
  if not Leer then
  for Index:=0 to Anzahl-1 do
  begin
    Kno:=self.Knoten(Index);
    Kno.Free;
    Kno:=nil;
  end;
  inherited Free
end;


function TKnotenliste.Knoten(Index:Integer):TKnoten;
begin
  if (Index<=Self.Anzahl-1)and (Index>=0)
  then
    result:=TKnoten(Tlist(self).Items[Index])
   else
    Writeln('Fehler! Listenindex außerhalb des zulässigen Be-
reichs');
end;


function TKnoten.WelcherGraph:TGraph;
begin
```

```
  WelcherGraph:=Graph_;
end;

procedure TKnoten.SetzeGraph(G:TGraph);
begin
  Graph_:=G;
end;

function TKnoten.WelcheEingehendeKantenliste:TKantenliste;
begin
  WelcheEingehendeKantenliste:=EingehendeKantenliste_;
end;

procedure TKnoten.SetzeEingehendeKantenliste(Kl:TKantenliste);
begin
  EingehendeKantenliste_:=Kl;
end;

function TKnoten.WelcheAusgehendeKantenliste:TKantenliste;
begin
  WelcheAusgehendeKantenliste:=AusgehendeKantenliste_;
end;

procedure TKnoten.SetzeAusgehendeKantenliste(Kl:TKantenliste);
begin
  AusgehendeKantenliste_:=Kl;
end;

function TKnoten.Istbesucht:Boolean;
begin
  Istbesucht:=besucht_;
end;

procedure TKnoten.Setzebesucht(B:Boolean);
begin
  besucht_:=B;
end;

constructor TKnoten.Create;
  begin
  Graph_:=nil;
  AusgehendeKantenliste_:=TKantenliste.create;
  EingehendeKantenliste_:=TKantenliste.create;
  Besucht_:=false;
end;

procedure TKnoten.Free;
begin
  AusgehendeKantenliste_.Free;
  AusgehendeKantenliste_:=nil;
```

```
  EingehendeKantenliste_.Free;
  EingehendeKantenliste_:=nil;
  inherited Free;
end;

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;

function TGraph.WelcheKnotenliste:TKnotenliste;
begin
  WelcheKnotenliste:=Knotenliste_;
end;

procedure TGraph.SetzeKnotenliste(Kl:TKnotenliste);
begin
  Knotenliste_:=Kl;
end;

function TGraph.WelcheKantenliste:TKantenliste;
begin
  WelcheKantenliste:=Kantenliste_;
end;

procedure TGraph.SetzeKantenliste(Kl:TKantenliste);
begin
  Kantenliste_:=Kl;
end;
```

```
constructor TGraph.Create;
begin
  inherited Create;
  Knotenliste_:=TKnotenliste.create;;
  Kantenliste_:=TKantenliste.create;
end;


procedure TGraph.Free;
begin
  Knotenliste_.Free;
  Knotenliste_:=nil;
  Kantenliste_.Free;
  Kantenliste_:=nil;
  inherited Free;
end;


procedure TGraph.Freeall;
begin
  Kantenliste_.Freeall;
  Kantenliste_:=nil;
  Knotenliste_.Freeall;
  Knotenliste_:=nil;
  inherited Free;
end;
function TGraph.Leer:Boolean;
begin
  Leer:=self.Knotenliste.Leer;
end;


procedure TGraph.KnotenEinfuegen(Kno:TKnoten);
begin
  Knotenliste.AmEndeanfuegen(Kno);
  Kno.Graph:=self;
end;


procedure TGraph.Knotenloeschen(Kno:TKnoten);
var Index:Integer;
    Ka:TKante;
begin
  if not Kno.AusgehendeKantenliste.Leer then
    for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
    begin
      Ka:=Kno.AusgehendeKantenliste.Kante(Index);
      if not Ka.KanteistSchlinge then
      begin
      Ka.Zielknoten(Kno).EingehendeKantenliste.LoescheElement(Ka);
```

```
        if not Ka.Gerichtet then
Ka.Zielknoten(Kno).AusgehendeKantenliste.
        LoescheElement(Ka);
      end;
      Kantenliste.LoescheElement(Ka);
    end;
  if not Kno.eingehendeKantenliste.Leer then
    for Index:=0 to Kno.EingehendeKantenliste.Anzahl-1 do
    begin
      Ka:=Kno.EingehendeKantenliste.Kante(Index);
      if not Ka.KanteistSchlinge then
      begin
      Ka.Quellknoten(Kno).AusgehendeKantenliste.LoescheElement(Ka);
        if not Ka.gerichtet then
Ka.Quellknoten(Kno).EingehendeKantenliste.
        loescheElement(Ka);
      end;
      Kantenliste.LoescheElement(Ka);
    end;
  Knotenliste.LoescheElement(Kno);
  Kno.Freeall;
  Kno:=nil;
end;


procedure TGraph.KanteEinfuegen
(Ka:Tkante;Anfangsknoten,EndKnoten:TKnoten;
  gerichtet:Boolean);
label Ende;
begin
  if (Anfangsknoten=nil) or (Endknoten=nil)or(Ka=nil) then goto
Ende;
  Ka.Anfangsknoten:=Anfangsknoten;
  Ka.Endknoten:=Endknoten;
  Anfangsknoten.AusgehendeKantenliste.AmEndeanfuegen(Ka);
  if Gerichtet then
  begin
    EndKnoten.EingehendeKantenliste.AmEndeanfuegen(Ka);
    Ka.Gerichtet:=true;
  end
  else
  begin
    Ka.Gerichtet:=false;
    Anfangsknoten.EingehendeKantenliste.AmEndeanfuegen(Ka);
    if  Anfangsknoten<>Endknoten then
    begin
      EndKnoten.AusgehendeKantenliste.AmEndeanfuegen(Ka);
      Endknoten.EingehendeKantenliste.AmEndeanfuegen(Ka);
    end;
  end;
```

```
  Ka.Anfangsknoten:=Anfangsknoten;
  Ka.EndKnoten:=EndKnoten;
  Kantenliste.amEndeanfuegen(Ka);
  Ende:
end;
procedure TGraph.Kanteloeschen(Ka:TKante);
begin
  Kantenliste.LoescheElement(Ka);
  Ka.Anfangsknoten.AusgehendeKantenliste.LoescheElement(Ka);
  if Ka.Gerichtet
  then
    Ka.EndKnoten.EingehendeKantenliste.LoescheElement(Ka)
  else
  begin
    if Ka.Anfangsknoten<>Ka.Endknoten then
    begin
      Ka.EndKnoten.AusgehendeKantenliste.LoescheElement(Ka);
      Ka.EndKnoten.EingehendeKantenliste.LoescheElement(Ka);
    end;
    Ka.Anfangsknoten.EingehendeKantenliste.LoescheElement(Ka);
  end;
  Ka.Free;
  Ka:=nil;
end;


procedure TGraph.LoescheKantenbesucht;
var Index:Integer;
begin
  for Index:=0 to Kantenliste.Anzahl-1 do
    Kantenliste.Kante(Index).Besucht:=false;
end;


procedure TGraph.LoescheKnotenbesucht;
var Index:Integer;
begin
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      Knotenliste.Knoten(Index).Besucht:=false;
end;


function TGraph.AnzahlKanten:Integer;
begin
  AnzahlKanten:=Kantenliste.Anzahl-AnzahlSchlingen;
end;
```

```
function TGraph.AnzahlSchlingen:Integer;
Var Graphschlingen:Integer;
    Index:Integer;
    Ka:TKante;

begin
  Graphschlingen:=0;
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
    begin
      Ka:=Kantenliste.Kante(Index);
      if Ka.KanteistSchlinge then
        Graphschlingen:=Graphschlingen+1;
    end;
  AnzahlSchlingen:=Graphschlingen;
end;function TGraph.AnzahlKantenmitSchlingen: Integer;
begin
  AnzahlKantenmitSchlingen:=AnzahlKanten+AnzahlSchlingen;
end;


function TGraph.AnzahlKnoten:Integer;
begin
  AnzahlKnoten:=Knotenliste.Anzahl;
end;


end.


Unit UInhgrphC:

unit UInhgrphC;
{$F+}

interface
uses

  UGraphC,
  WinCrt,
  Sysutils,Wintypes,WinProcs, Classes,UListC;

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;

implementation

constructor TInhaltsknoten.Create;
begin
  inherited Create;
  Inhalt_:=' ';
end;

procedure TInhaltsknoten.Free;
begin
```

```
  inherited Free;
end;

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

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

procedure TInhaltsknoten.Knotenwerteinlesen;
var S:string;
begin
   Write(‘Knotenwert eingeben: ‘);
   Readln(S);
   Wert:=S;
end;


constructor TInhaltskante.Create;
begin
  inherited Create;
  Inhalt_:=’ ‘;
end;

procedure TInhaltskante.Free;
begin
  inherited Free;
end;

procedure TInhaltskante.Freeall;
begin
  inherited Freeall;
end;


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

function TInhaltskante.Wertlesen:string;
begin
```

```
  Wertlesen:=Inhalt_;
end;

procedure TInhaltskante.Kantenwerteinlesen;
var S:string;
begin
   Write('Kantenwert eingeben: ');
   Readln(S);
   Wert:=S;
end;

procedure TInhaltskante.Listenausgabe;
var Zaehl:Integer;
begin
  if not leer then
  begin
   Write(TInhaltsknoten(Kante(0).Zielknoten(self.Kante(0).Pfadrichtung)).Wert,
  ' ');
    for Zaehl:=0 to Anzahl-1 do
      Write(TInhaltsknoten(Kante(Zaehl).Pfadrichtung).Wert,' ');
    Write('Summe: ',WertsummederElemente(Bewertung):6:2,' ');
    Write('Produkt: ',WertproduktderElemente(Bewertung):6:2);
    Writeln;
  end;
end;

constructor TInhaltsgraph.Create;
begin
  inherited Create;
end;

procedure TInhaltsgraph.Free;
begin
  inherited Free;
end;

procedure TInhaltsgraph.Freeall;
begin
  inherited Freeall;
end;


function
TInhaltsgraph.Graphknoten(Kno:TInhaltsknoten):TInhaltsknoten;
var Index:Integer;
    Kn:TInhaltsknoten;
begin
  Graphknoten:=nil;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
```

```
    begin
      Kn:=TInhaltsknoten(Knotenliste.Knoten(Index));
      if Kno.Wert=Kn.Wert
      then
      begin
        Graphknoten:=Kn;
        Kn.Graph:=self;
        exit;
      end
    end;
end;


procedure TInhaltsgraph.Knotenerzeugenundeinfuegen;
var Kno:TInhaltsknoten;
begin
  Kno:=TInhaltsknoten.create;
  Kno.KnotenWerteinlesen;
  self.Knoteneinfuegen(Kno);
end;

procedure TInhaltsgraph.KnotenausGraphloeschen;
var W:string;
    Index:Integer;
    Kno:TKnoten;
    Gefunden:Boolean;
begin
  Writeln('Knoten löschen');
  Writeln;
  Write('Knotenwert eingeben: ');
  Readln(W);
  Gefunden:=false;
  if not Knotenliste.leer then
  begin
    Index:=0;
    repeat
      Kno:=Knotenliste.Knoten(Index);
      if TInhaltsknoten(Kno).Wert=W then
      begin
        Gefunden:=true;
        Knotenloeschen(Kno);
      end;
      Index:=Index+1
    until Gefunden or (Index>Knotenliste.Anzahl-1);
    if Gefunden then Writeln('Knoten ',W,' gelöscht!') else
     Writeln
    ('Knoten ',W,' nicht im Graph!');
  end;
end;
```

```
procedure TInhaltsgraph.FuegeKanteein(Kno1,Kno2:TInhaltsknoten;
  Gerichtet:Boolean;Ka:TInhaltskante);
label Endschleife1,Endschleife2;
var Richtung:Boolean;
    Pos1,Pos2,Index:Integer;
    Knoa,Knoe:TKnoten;
    Kno:TInhaltsknoten;
begin
  if Gerichtet then Richtung:=true else Richtung:=false;
  Pos1:=-1;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TInhaltsknoten(Knotenliste.Knoten(Index));
      if Kno.Wert=Kno1.Wert then
      begin
        Pos1:=Index;
        goto Endschleife1;
      end;
    end;
  Endschleife1:
  Pos2:=-1;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TInhaltsknoten(Knotenliste.Knoten(Index));
      if Kno.Wert=Kno2.Wert then
      begin
        Pos2:=Index;
        goto Endschleife2;
      end;
    end;
  Endschleife2:
  if Pos1>=0
  then
    Knoa:=Knotenliste.Knoten(Pos1)
  else
  begin
    Knoteneinfuegen(Kno1);
    Knoa:=Kno1;
  end;
  if Pos2>=0
  then
    Knoe:=self.Knotenliste.Knoten(Pos2)
  else
  begin
    Knoteneinfuegen(Kno2);
    Knoe:=Kno2;
  end;
  if (Knoa<>nil)and (Knoe<>nil)and (Ka<>nil) then Kanteeinfuegen
```

```
  (Ka,Knoa,Knoe,Richtung);
end;

procedure TInhaltsgraph.Kanteerzeugenundeinfuegen;
var Kw1,Kw2,W,Antwort:string;
    Index:Integer;
    Gerichtet:Boolean;
    Kno1,Kno2:TInhaltsknoten;
    Ka:TInhaltskante;
begin
  Writeln('Kante erzeugen:');
  Write('Anfangsknotenwert: ');
  Readln(Kw1);
  Write('Endknotenwert: ');
  Readln(Kw2);
  Write('Kantenwert: ');
  Readln(W);
  Write('Gerichtetet? (j/n): ');
  Readln(Antwort);
  if (Antwort='j') or (Antwort='J') then Gerichtet:=true else
Gerichtet:=false;
  Kno1:=TInhaltsknoten.Create;
  Kno1.Wert:=Kw1;
  Kno2:=TInhaltsknoten.Create;
  Kno2.Wert:=kw2;
  Ka:=TInhaltskante.Create;
  Ka.Wert:=w;
  self.FuegeKanteein(Kno1,Kno2,Gerichtet,Ka);
end;



procedure TInhaltsgraph.LoescheKante(Kno1,Kno2:TInhaltsknoten;
Ka:TInhaltskante);
var Index:Integer;
    kant:TKante;
begin
  if not Kantenliste.leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
    begin
      Kant:=Self.Kantenliste.Kante(Index);
      if ((Kant.Anfangsknoten.Wert=Kno1.Wert) and
      (Kant.Endknoten.Wert=Kno2.Wert)
        and
      (Kant.Wert=Ka.Wert))
      then
      begin
        Kanteloeschen(Kant);
        exit;
      end;
```

```
    end;
end;


procedure TInhaltsgraph.KanteausGraphloeschen;
var Kw1,Kw2,W,Antwort:string;
    Index:Integer;
    Kno1,Kno2:TInhaltsknoten;
    Ka:TInhaltskante;
begin
  Writeln('Kante löschen:');
  Writeln;
  Write('Anfangsknotenwert: ');
  Readln(Kw1);
  Write('Endknotenwert: ');
  Readln(Kw2);
  Write('Kantenwert: ');
  Readln(W);
  Kno1:=TInhaltsknoten.Create;
  Kno1.Wert:=Kw1;
  Kno2:=TInhaltsknoten.Create;
  Kno2.Wert:=Kw2;
  Ka:=TInhaltskante.Create;
  Ka.Wert:=W;
  LoescheKante(Kno1,Kno2,Ka);
  Kno1.Freeall;
  Kno1:=nil;
  Kno2.Freeall;
  Kno2:=nil;
  Ka.Free;
  Ka:=nil;
end;

procedure TInhaltsgraph.ZeigeGraph;
var Index:Integer;
    Kno:TKnoten;
    Ka:TInhaltskante;
begin
  Writeln('Anzeige des Graphen:');
  Writeln;
  Writeln('Knoten:');
  Writeln;
  if self.Knotenliste.leer then Writeln('Knotenliste leer!');
  if not self.Knotenliste.leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=Knotenliste.Knoten(Index);
      Writeln(Index,': Wert: ',Kno.Wert);
    end;
  Writeln;
```

```
  Writeln('Kanten:');
  Writeln;
  if Kantenliste.leer then Writeln('Kantenliste leer!');
  if not Kantenliste.leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
    begin
      Ka:=TInhaltskante(self.Kantenliste.Kante(Index));
      Write(Index,': Wert: ',Ka.Wert,' Anfangsknoten:
      ',Ka.Anfangsknoten.Wert,
      ' Endknoten: ',Ka.Endknoten.Wert);
      if TInhaltskante(Ka).gerichtet
        then Writeln(' gerichtet') else Writeln(' ungerichtet');
    end;
  Writeln;
end;


function Bewertung (X:TObject):Extended;
begin
  if  X is TInhaltskante
  then
    Bewertung := StringtoReal(TInhaltskante(X).Wert)
  else
    begin
      if X is TInhaltsknoten
      then
        Bewertung:=StringtoReal(TInhaltsknoten(X).Wert)
      else
        Bewertung:=1;
    end;
end;


end.

Unit UMathC:

unit UMathC;
 {$F+}

interface

uses  UGraphC,UInhgrphC,UListC,
      Wincrt,Classes,
      SysUtils, WinTypes, WinProcs;

type

  TEulerCgraph   = class(TInhaltsgraph)
  public
```

```
  constructor Create;
  procedure Free;
  procedure Freeall;
  procedure Euler
  (Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;var Kliste:
  TKantenliste;EineLoesung:Boolean;var Gefunden:Boolean);
  procedure Eulerlinie
 (Anfangsknoten,Endknoten:TInhaltsknoten);
  procedure Menu;
end;

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;


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;
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;

  TPfadCknoten=class(TInhaltsknoten)
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    procedure ErzeugeTiefeBaumPfade(Preorder:Boolean);
    procedure AllePfadeundMinimalerPfad(Kno:TInhaltsknoten;var
    Minlist:TKantenliste);
  end;

  TPfadCgraph=class(TInhaltsgraph)
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    procedure Menu1;
    procedure Menu2;
  end;

implementation

constructor TEulerCgraph.Create;
begin
   inherited Create;
end;

procedure TEulerCgraph.Free;
begin
  inherited Free;
end;

procedure TEulerCgraph.Freeall;
begin
  inherited Freeall;
end;

procedure TEulerCgraph.Euler(Stufe:Integer;Kno,Zielknoten:
TInhaltsknoten;var Kliste:TKantenliste;EineLoesung:Boolean;var
Gefunden:Boolean);
var Index:Integer;
    Zkno:TInhaltsknoten;
    Ka:TInhaltskante;
    Zaehl:Integer;
```

```
begin
   if EineLoesung and Gefunden then exit;
   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<=Self.AnzahlKantenmitSchlingen)) then
        begin
          Ka.Pfadrichtung:=Zkno;
          Ka.Besucht:=true;
          Kliste.AmEndeanfuegen(Ka);
          Stufe:=Stufe+1;
          if (Zkno=Zielknoten)and
         (Stufe=Self.AnzahlKantenmitSchlingen+1)  then
          begin
            TInhaltskante(Kliste).Listenausgabe;
            Gefunden:=true;
            if EineLoesung then exit;
          end
          else
         Euler(Stufe,Zkno,Zielknoten,Kliste,EineLoesung,Gefunden);
          Stufe:=Stufe-1;
          Ka.Besucht:=false;
          if not Kliste.Leer then
            Kliste.AmEndeloeschen(TObject(Ka));;
        end;
     end;
end;

procedure TEulerCgraph.Eulerlinie
(Anfangsknoten,Endknoten:TInhaltsknoten);
var MomentaneKantenliste:TKantenliste;
    Zaehl:Integer;
    EineLoesung:Boolean;
    Gefunden:Boolean;
    Antwort:string;
begin
   EineLoesung:=false;
   Gefunden:=false;
   Write('Nur eine Lösung? (j/n): ');
   Readln(Antwort);
   if (Antwort='j')or(Antwort='J') then EineLoesung:=true else
EineLoesung:=false;
   if Leer then exit;
   MomentaneKantenliste:=TKantenliste.create;
   LoescheKantenbesucht;
   Writeln('Die Eulerlinien sind:');
   Writeln;
```

```
    Euler(1,Anfangsknoten,Endknoten,MomentaneKantenliste,EineLoesung,Gefunden);
   if not Gefunden then Writeln('Keine Lösung!');
   MomentaneKantenliste.Free;
   MomentaneKantenliste:=nil;
end;

Procedure TEulerCgraph.Menu;
var Kno1,Kno2,K1,K2:TInhaltsknoten;
    W,Antwort:string;
begin
   if Leer
   then
   begin
      Writeln('leerer Graph');
      Readln;
      exit;
   end;
   Clrscr;
   Writeln('Eulerlinien');
   Writeln;
   Write('Geschlossene oder offene Eulerlinie? (g/o) ');
   Readln(Antwort);
   Writeln;
   Write('Eingabe des Startknoten: ');
   Kno1:=TInhaltsknoten.Create;
   repeat
      Readln(W);
      Kno1.Wert:=W;
      K1:=Graphknoten(Kno1);
      K2:=K1;
   until K1<>nil;
   Kno1.Free;
   Kno1:=nil;
   if (Antwort='o')or(Antwort='O') then
   begin
     Write('Eingabe des Endknoten: ');
     Kno2:=TInhaltsknoten.create;
     repeat
       Readln(W);
       Kno2.Wert:=W;
       K2:=Graphknoten(Kno2);
     until K2<>nil;
     Kno2.Free;
     Kno2:=nil;
   end;
   Eulerlinie(K1,K2);
   Writeln;
   Readln;
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;


constructor TFarbCknoten.Create;
begin
  inherited create;
  Farbzahl_:=0;
  Ergebnis_:='';
end;

procedure TFarbCknoten.Free;
begin
  inherited Free;
end;

procedure TFarbCknoten.Freeall;
begin
  inherited Freeall;
end;
function TFarbCknoten.WelcheFarbzahl:Integer;
begin
  WelcheFarbzahl:=Farbzahl_;
end;

procedure TFarbCknoten.SetzeFarbzahl(Fa:Integer);
begin
  Farbzahl_:=Fa;
end;

function TFarbCknoten.WelchesErgebnis:String;
begin
  WelchesErgebnis:=Ergebnis_;
end;

procedure TFarbCknoten.SetzeErgebnis(S:string);
begin
  Ergebnis_:=S;
end;
constructor TFarbCgraph.Create;
begin
```

```
  inherited Create;
end;

procedure TFarbCgraph.Free;
begin
  inherited Free;
end;

procedure TFarbCgraph.Freeall;
begin
  inherited Freeall;
end;

procedure TFarbCgraph.SetzebeiallenKnotenAnfangsfarbe;
var Index:Integer;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl -1 do
      TFarbCknoten(Knotenliste.Knoten(Index)).KnotenFarbe:=0;
end;


function TFarbCgraph.Knotenistzufaerben
(Index:Integer;AnzahlFarben:Integer):Boolean;
var Kno:TFarbCknoten;
    GleicheFarbe:Boolean;
    Zaehl:Integer;

  function NachbarknotenhabengleicheFarbe(Ka:TKante):Boolean;
  var kno1,kno2:TFarbCknoten;
  begin
    if Ka.KanteistSchlinge then
    NachbarknotenhabengleicheFarbe:=false;
    Kno1:=TFarbCknoten(Ka.Anfangsknoten);
    Kno2:=TFarbCknoten(Ka.Endknoten);
    if not Ka.KanteistSchlinge then
    begin
      if Kno1.KnotenFarbe =Kno2.KnotenFarbe
      then
        NachbarknotenhabengleicheFarbe:=true
      else
        NachbarknotenhabengleicheFarbe:=false;
    end;
  end;

begin
  Kno:=TFarbCknoten(Knotenliste.Knoten(Index));
  repeat
    Kno.KnotenFarbe:=(Kno.KnotenFarbe+1) mod (AnzahlFarben+1);
    if Kno.Knotenfarbe=0
```

```
    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;
    if not Kno.eingehendeKantenliste.Leer  then
      for Zaehl:=0 to Kno.EingehendeKantenliste.Anzahl-1 do
        if NachbarknotenhabengleicheFarbe
        (Kno.EingehendeKantenliste.Kante(Zaehl)) then
          GleicheFarbe:=true;
  until (Not GleicheFarbe) ;
  KnotenistzuFaerben:=true;
end;



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;

procedure TFarbCgraph.ErzeugeErgebnis;
var Index:Integer;
    Kno:TFarbCknoten;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TFarbCknoten(Knotenliste.Knoten(Index));
    Kno.SetzeErgebnis(Kno.Wert+':'+IntegertoString(Kno.Knotenfarbe));
    end;
end;

procedure TFarbCgraph.FaerbeGraph(var Sliste:TStringlist);
var AnzahlFarben:Integer;
    StringFarben,Antwort:string;
    Gefunden,EineLoesung:Boolean;
    Index:Integer;
    Ausgabeliste:TStringlist;
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;

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;
  Farbgraph:=nil;
end;

constructor TPfadCknoten.Create;
begin
   inherited Create;
end;

procedure TPfadCknoten.Free;
begin
  inherited Free;
end;

procedure TPfadCknoten.Freeall;
begin
  inherited Freeall;
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;
      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;


procedure TPfadCknoten.
AllePfadeundMinimalerPfad(Kno:TInhaltsknoten;
var Minlist:TKantenliste);
var Index:Integer;
    MomentaneKantenliste:TKantenliste;


  procedure GehezuNachbarknoten(K:TKnoten;Ka:TKante);
  label endproc;
  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;
```

```
    Endproc:
  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;

constructor TPfadCgraph.Create;
begin
   inherited Create;
end;

procedure TPfadCgraph.Free;
begin
  inherited Free;
end;

procedure TPfadCgraph.Freeall;
begin
  inherited Freeall;
end;

procedure TPfadCgraph.Menu1;
var Kno,K:TPfadCknoten;
    W,Antwort:string;
    Sliste:TStringlist;
begin
  if Leer
  then
  begin
     Writeln('leerer Graph');
     Readln;
     exit;
  end;
  Clrscr;
  Writeln('Tiefe Baumpfade');
  Writeln;
  Write('Eingabe des Startknoten: ');
  Kno:=TPfadCknoten.Create;
  repeat
    Readln(W);
    Kno.Wert:=W;
```

```
    K:=TPfadCknoten(Graphknoten(Kno));
  until K<>nil;
  Kno.Free;
  Kno:=nil;
  Write('Preorder-Reihenfolge? (j/n) (sonst Postorder): ');
  Readln(Antwort);
  Sliste:=TStringlist.Create;
  if (Antwort='J')or(Antwort='j')
  then
    K.ErzeugetiefeBaumpfade(true)
  else
    K.ErzeugetiefeBaumpfade(false);
  Writeln;
  Readln;
end;

procedure TPfadCgraph.Menu2;
var Kno,Kn,K:TPfadCknoten;
    W,Antwort:string;
    Sliste:TStringlist;
    Index:Integer;
    Minlist:TKantenliste;
begin
  if Leer
  then
  begin
     Writeln('leerer Graph');
     Readln;
     exit;
  end;
  Clrscr;
  Writeln('Alle Pfade zwischen zwei Knoten und minimaler Pfad');
  Writeln;
  Write('Eingabe des Startknoten: ');
  Kno:=TPfadCknoten.Create;
  Repeat
    Readln(W);
    Kno.Wert:=W;
    K:=TPfadCknoten(Graphknoten(Kno));
  until K<>nil;
  Kno.Free;
  Kno:=nil;
   Writeln;
  Write('Eingabe des Endknoten: ');
  Kno:=TPfadCknoten.create;
  Repeat
    Readln(W);
    Kno.Wert:=W;
    Kn:=TPfadCknoten(Graphknoten(Kno));
  until Kn<>nil;
```

```
  Kno.Free;
  Kno:=nil;
  Writeln;
  Writeln('Alle Pfade');
  Writeln;
  Minlist:=TKantenliste.Create;
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
      Minlist.AmEndeAnfuegen(Kantenliste.Kante(Index));
  K.AllePfadeundMinimalerPfad(Kn,Minlist);
  Writeln;
  Writeln('Minimaler Pfad:');
  Writeln;
  TInhaltskante(Minlist).Listenausgabe;
  Readln;
  Minlist.Free;
  Minlist:=nil;
end;


end.
```

Projekt-Quelltext/Hauptprogramm:

```
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('Eulerkreis...........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.
```

7)Quelltext der Programme Knotengraph DWK und der objekt-orientierten Entwicklungsumgebung EWK

Unit UList:

```
unit UList;
{$F+}

interface

uses
  Classes,Winprocs,Forms,Dialogs,Sysutils;

type
  TVorgang=procedure(Ob:TObject);
  THandlung=procedure(Ob1,Ob2:TObject);
  TBedingung=function(Ob:TObject):Boolean;
  TWert=function(Ob:TObject):Extended;
  TVergleich=function(Ob1,Ob2:TObject;Wert:TWert):Boolean;

  TElement=class;

  TListe=class(Tlist)
  public
    constructor Create;
    procedure Free;
    procedure Freeall;
    function Element(Index:Integer):TElement;
    property Items[Index:Integer]:TElement read Element;
    procedure AmEndeanfuegen(Ob:TObject);
    procedure AmAnfanganfuegen(Ob:TObject);
    procedure AnPositioneinfuegen(Ob:TObject;Index:Integer);
    procedure LoescheanderPosition(var Ob:TObject;N:Integer);
    procedure AmAnfangloeschen(var Ob:TObject);
    procedure AmEndeloeschen(var Ob:TObject);
    procedure LoescheElement(Ob:TObject);
    procedure VertauscheElemente(Index1,Index2:Integer);
    procedure VerschiebeElement(Index1,Index2:Integer);
    procedure FuerjedesElement(Vorgang:TVorgang);
    procedure FuerjedesElementzurueck(Vorgang:TVorgang);
    procedure FueralleElemente(Ob:TObject;Handlung:THandlung);
     procedure
    FueralleElementezurueck(Ob:TObject;Handlung:THandlung);
    procedure Loeschen;
    procedure Sortieren(Vergleich:TVergleich;Wert:TWert);
    function Anzahl:Integer;
    function WertSummederElemente(Wert:TWert):Extended;
    function WertproduktderElemente(Wert:Twert):Extended;
    function Leer:Boolean;
```

```
    function Erstes:Integer;
    function Letztes:Integer;
    function Position(Ob:TObject):Integer;
    function ElementistinListe(Ob:TObject):Boolean;
    function ErsterichtigePosition
    (Bedingung:TBedingung):Integer;
    function ErstefalschePosition(Bedingung:TBedingung):Integer;
    function LetzterichtigePosition
    (Bedingung:TBedingung):Integer;
    function ErstepassendePosition
    (Vergleich:TVergleich;Ob:TObject;
    Wert:TWert):Integer;
    function ErsteunpassendePosition
    (Vergleich:TVergleich;Ob:TObject;
    Wert:TWert):Integer;
    function ErstebestePosition
    (Vergleich:TVergleich;Wert:TWert):Integer;
    function LetztebestePosition
    (Vergleich:TVergleich;Wert:TWert):Integer;
  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;

  function GGT(A,B:Longint):Longint;
  function Tan(X:Extended):Extended;
  function StringtoReal(S:string):Extended;
  function RealtoString(R:Extended):string;
  function Integertostring(I:Integer):string;
  function StringtoInteger(S:string):Integer;
  function StringistRealZahl(S:string):Boolean;
  function RundeStringtoString(S:string;Stelle:Integer):string;
  function RundeZahltoString(R:Real;Stelle:Integer):string;
  function Minimum(R1,R2:Extended):Extended;
```

```
  function Maximum(R1,R2:Extended):Extended;
  procedure Pause (N:Longint);


implementation

constructor TListe.Create;
begin
  inherited Create;
end;


procedure TListe.Free;
begin
  inherited Free;
end;

procedure TListe.Freeall;
var Index:Integer;
    El:TElement;
begin
  if not Leer then
  for Index:=0 to Anzahl-1 do
  begin
    El:=Element(Index);
    El.Free;
    El:=nil;
  end;
  inherited Free;
end;

function TListe.Element(Index:Integer):TElement;
begin
  if (Index<=Anzahl-1)and (Index>=0)
  then
    result:=TElement(Tlist(self).Items[Index])
  else
    ShowMessage('Fehler! Listenindex außerhalb des zulässigen
Bereichs');
end;


procedure TListe.AmEndeanfuegen(Ob:TObject);
begin
  try
    Capacity:=Count;
    Add(Ob);
  except
    end;
end;
```

```
procedure TListe.AmAnfanganfuegen(Ob:TObject);
begin
    Capacity:=Count;
    Insert(0,Ob);
end;
procedure TListe.AnPositioneinfuegen(Ob:TObject;Index:Integer);
begin
    Capacity:=Count;
    Insert(Index,Ob);
end;

procedure TListe.LoescheanderPosition(var Ob:TObject;N:Integer);
begin
  Ob:=Items[n];
  Delete(n);
end;


procedure TListe.AmAnfangloeschen(var Ob:TObject);
begin
  Ob:=Items[0];
  Delete(0);
end;


procedure TListe.AmEndeloeschen(var Ob:TObject);
begin
  Ob:=Items[Count-1];
  Delete(Count-1);
end;


procedure TListe.LoescheElement(Ob:TObject);
begin
  Remove(Ob);
  Pack;
end;


procedure TListe.VertauscheElemente(Index1,Index2:Integer);
begin
  Exchange(Index1,Index2);
end;


procedure TListe.VerschiebeElement(Index1,Index2:Integer);
begin
  Move(Index1,Index2);
end;
```

```
procedure TListe.FuerjedesElement(Vorgang:TVorgang);
var Index:Integer;
begin
  If not Leer then
    for  Index:=0 to Count-1 do
      Vorgang(Items[Index]);
end;


procedure TListe.FuerjedesElementzurueck(Vorgang:TVorgang);
var Index:Integer;
begin
  for  Index:=Count-1 to 0 do
    Vorgang(Items[Index]);
end;


procedure TListe.FueralleElemente
(Ob:TObject;Handlung:THandlung);
var Index:Integer;
begin
 for  Index:=0 to Count-1 do
   Handlung(Ob,Items[Index]);
end;

procedure TListe.FueralleElementezurueck
(Ob:TObject;Handlung:THandlung);
var Index:Integer;
begin
 for  Index:=Count-1 to 0 do
   Handlung(Ob,Items[Index]);
end;

procedure TListe.Loeschen;
begin
  Clear;
end;


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;



function TListe.Anzahl:Integer;
begin
  Anzahl:=Count;
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 raise
    end;
  end;

begin
  Wertsumme:=0;
  if not Leer then
```

```
    for Index:=0 to Anzahl-1 do
      SummiereWert(Element(Index));
  WertsummederElemente:=Wertsumme;
end;
function TListe.WertproduktderElemente(Wert:TWert):Extended;
var Index:Integer;
    Wertprodukt:Extended;

  procedure MultipliziereWert(Ob:TObject);
  var Z:Real;
  begin
    try
      if abs(Wertprodukt*Wert(Ob))<1E40
      then
        Wertprodukt:=Wertprodukt*Wert(Ob)
      else
      begin
        ShowMessage('Wertprodukt zu groß!');
        Z:=0;
        Wertprodukt:=1E40/Z;
      end;
    except raise
    end;
  end;


begin
  if Anzahl>0
  then
    Wertprodukt:=1
  else
    Wertprodukt:=0;
  if not Leer then
    for Index:=0 to Anzahl-1 do
      MultipliziereWert(Element(Index));
    WertproduktderElemente:=Wertprodukt;
end;


function TListe.Leer:Boolean;
begin
  Leer:=(self=nil) or (Count=0);
end;


function TListe.Erstes:Integer;
begin
  Erstes:=0;
end;
```

```
function TListe.Letztes:Integer;
begin
  Letztes:=Count-1;
end;


function TListe.Position(Ob:TObject):Integer;
begin
  Position:=Indexof(Ob);
end;


function TListe.ElementistinListe(Ob:TObject):Boolean;
begin
  if Position(Ob)>=0
  then
    ElementistinListe:=true
  else
    ElementistinListe:=false;
end;


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;


function TListe.ErstefalschePosition
(Bedingung:TBedingung):Integer;
var Index,Stelle:Integer;
begin
  Index:=0;
  while (Index<=Count-1)and  Bedingung(Items[Index]) do
    Index:=Index+1;
  Stelle:=Index;
  if Stelle>Count-1 then Stelle:=-1;
  ErstefalschePosition:=Stelle;
end;

function TListe.LetzterichtigePosition(Bedingung:
  TBedingung):Integer;
var Index,Stelle:Integer;
```

```
begin
  Stelle:=Count;
  for Index:=0 to Count-1 do
  begin
    if Bedingung(Items[Index]) then Stelle:=Index;
  end;
  if Stelle>Count-1 then Stelle:=-1;
  LetzterichtigePosition:=Stelle;
end;


function TListe.ErstepassendePosition
(Vergleich:TVergleich;Ob:TObject;Wert:TWert):Integer;
var Index,Stelle:Integer;
begin
  Index:=0;
  while (Index<=Count-1)and not Vergleich(Items[Index],Ob,Wert)
do
    Index:=Index+1;
  Stelle:=Index;
  if Stelle>Count-1 then Stelle:=-1;
  ErstepassendePosition:=Stelle;
end;



function
TListe.ErsteunpassendePosition(Vergleich:TVergleich;Ob:TObject;Wert:TWert):Integer;
var Index,Stelle:Integer;
begin
  Index:=0;
  while (Index<=Count-1)and not Vergleich(Items[Index],Ob,Wert)
do Index:=Index+1;
  Stelle:=Index;
  if Stelle>Count-1 then Stelle:=-1;
  ErsteunpassendePosition:=Stelle;
end;


function
TListe.ErstebestePosition(Vergleich:TVergleich;Wert:TWert):Integer;
var Index,Stelle:Integer;
begin
  Stelle:=-1;
  for Index:=Count-1 to 0 do
    if Vergleich(Items[Index],Items[Stelle],Wert) then
Stelle:=Index;
  ErstebestePosition:=Stelle;
end;
```

```
function TListe.LetztebestePosition(Vergleich:TVergleich;
Wert:TWert):Integer;
var Index,Stelle:Integer;
begin
  Stelle:=-1;
  for Index:=0 to Count-1 do
    if Vergleich(Items[Index],Items[Stelle],Wert) then
Stelle:=Index;
  LetztebestePosition:=Stelle;
end;


constructor TElement.Create;
begin
  Wertposition_:=0;
  Wertliste_:=nil;
  inherited Create;
end;


procedure TElement.Free;
begin
  Wertliste_.Free;
  Wertliste_:=nil;
  inherited Free;
end;


procedure TElement.SetzeWertposition(P:Integer);
begin
  if p>=0 then Wertposition_:=P;
end;


function TElement.WelcheWertposition:Integer;
begin
  WelcheWertposition:=Wertposition_;
end;


procedure TElement.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 TElement.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;


function GGT(A,B:Longint):Longint;
var H,C:Longint;
begin
  if A<B then
  begin
    H:=A;
    A:=B;
    B:=H;
  end;
  repeat
    C:=A mod B;
    A:=B;
    B:=C
  until C=0;
  GGT:=A;
end;

function Tan(X:Extended):Extended;
begin
  Tan:=sin(X)/cos(X);
end;
function StringtoReal(S:string):Extended;
var Feld:array[0..101] of char;
    R:Extended;
begin
  try
    if Length(S)<101 then
    begin
      Strpcopy(Feld,S);
      Decimalseparator:='.';
      if TexttoFloat(Feld,R,fvExtended)
      then
        StringtoReal:=R
      else
```

```
        StringtoReal:=0;
    end
    else
      StringtoReal:=0;
  except
    on EConvertError do
    begin
      ShowMessage('Umwandlung von Zahlen außerhalb des zulässi-
gen Wertebereichs!');
      raise;
    end;
  end;
end;

function RealtoString(R:Extended):string;
begin
  try
    Decimalseparator:='.';
    Realtostring:=Floattostr(R);
  except
    on EConvertError do
    begin
      ShowMessage('Umwandlung von Zahlen außerhalb des zulässi-
gen Wertebereichs!');
      raise;
    end;
  end;
end;

function Integertostring(I:Integer):string;
begin
  try
    Integertostring:=InttoStr(I);
  except
    on EConvertError do
    begin
      ShowMessage('Umwandlung von Zahlen außerhalb des zulässi-
gen Wertebereichs!');
      raise;
    end;
  end;
end;

function StringtoInteger(S:string):Integer;
begin
  try
    StringtoInteger:=StrtoIntDef(S,0);
  except
    on EConvertError do
    begin
```

```
      ShowMessage('Umwandlung von Zahlen außerhalb des zulässi-
gen Wertebereichs!');
      raise;
    end;
  end;
end;


function StringistRealZahl(S:string):Boolean;
var Zaehl:Integer;
    IstRealZahl,Ezahl:Boolean;
begin
  IstRealZahl:=true;
  Ezahl:=true;
  for Zaehl:=1 to Length(S) do
  begin
    if (not (S[Zaehl] in
    ['0','1','2','3','4','5','6','7','8','9','.','+','-',' ']))
    then
    begin
      IstRealZahl:=false;
      if (not (S[Zaehl]='E')) then Ezahl:=false;
    end;
    if (S[Zaehl]='E') and (Zaehl>1)and (Ezahl=true)
    then
      IstRealZahl:=true;
  end;
  StringistRealZahl:=IstRealZahl;
end;

function RundeStringtoString(S:string;Stelle:Integer):string;
label Endproc,Ende,Stelle0;
var Zaehl,I:Integer;
    St,St1,St2,Sd,T:string;
    Position:Integer;
    R:Extended;
    Z:Integer;
    Exponent:Boolean;
begin
  try
    if Stelle=100 then
    begin
      RundeStringtoString:=S;
      goto endproc;
    end;
    Exponent:=false;
    if Stelle<0 then goto Ende;
    Stelle:=Abs(Stelle);
    if Stelle>7 then Stelle:=7;
    if StringistRealZahl(S) then
```

```
begin
  if Stelle=0 then
  begin
    St1:=S;
    Z:=pos(‘E’,St1);
    if Z>0 then
    begin
      Delete(St1,Z,255);
      St2:=S;
      Delete(St2,1,Z-1);
      Exponent:=true;
      S:=St1;
    end;
    St:=S;
    Z:=pos(‘.’,St);
    if Z>0 then
      Delete(St,Z,255);
    RundeStringtoString:=St;
    goto Stelle0;
  end;
  St1:=S;
  Z:=pos(‘E’,St1);
  if Z>0 then
  begin
    Delete(St1,Z,255);
    St2:=S;
    Delete(St2,1,Z-1);
    exponent:=true;
    S:=St1;
  end;
    St:=S;
    Z:=pos(‘.’,St);
    if Z>0
    then
      Delete(St,Z+Stelle+1,255)
    else
      if pos(‘.’,St)=0 then St:=St+’.0000000';
    if pos(‘.’,St)=length(S) then St:=St+’0000000';
    Z:=pos(‘.’,St);
    Sd:=St;
    Delete(Sd,1,Z);
    T:=’’;
    if 7-length(S)>0 then
    begin
      for I:=1 to 7-length(S) do T:=T+’0';
      St:=St+T;
    end;
    position:=pos(‘.’,St);
    Delete(St,Position+Stelle+1,30);
    Stelle0:
```

```
        if exponent then St:=St+St2;
        RundeStringtoString:=St;
    end
    else
      RundeStringtoString:=S;
    goto endproc;
    ende:
    Delete(S,Abs(Stelle),255);
    RundeStringtoString:=S;
    endproc:
  except
    begin
      ShowMessage('Rundung von Zahlen nicht ausführbar!');
      raise;
    end;
  end;
end;


function RundeZahltoString(R:Real;Stelle:Integer):string;
var Zaehl:Integer;
    St:string;
    Position:Integer;
begin
  try
    St:=RealtoString(R);
    RundeZahltoString:=RundeStringtoString(St,Stelle);
  except
    on EConvertError do
    begin
      ShowMessage('Umwandlung von Zahlen außerhalb des zulässi-
gen Wertebereichs!');
      raise;
    end;
  end;
end;


function Minimum(R1,R2:Extended):Extended;
begin
  if R1 <=R2 then
    Minimum:=R1
  else
    Minimum:=R2;
end;

function Maximum(R1,R2:Extended):Extended;
begin
  if R1 >=R2
  then
```

```
    Maximum:=R1
  else
    Maximum:=R2;
end;


procedure Pause(N:Longint);
var J:Longint;
begin
  J:=Gettickcount;
  repeat
    Application.processmessages;
  until (Gettickcount-J>N);
end;

end.
```

Unit Ugraph:

```
unit UGraph;
{$F+}

interface

uses
 Classes,Winprocs,Forms,Dialogs,Sysutils,UList;

type

  TString=function(Ob:TObject):string;

  TKante  = class;

  TPfad   = class;

  TKnotenliste = class;

  TKnoten = class;

  TGraph  = class;

  TKantenliste = class(TListe)
    public
    constructor Create;
    procedure Free;
    procedure Freeall;
    function Kante(Index:Integer):TKante;
    property Items[Index:Integer]:TKante read Kante;
    function Kopie:TKantenliste;
    function Graph:TGraph;
```

```
  function UGraph:TGraph;
  function Kantenlistealsstring:string;
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 Setze
  gerichtet;
  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 Setzeer
```

```
   reicht;
  property Wert:string read Wertlesen write Wertschreiben;
  function Wertlisteschreiben:Tstringlist;virtual;abstract;
  procedure Wertlistelesen;virtual;abstract;
  function Zielknoten(Kno:TKnoten):TKnoten;
  function Quellknoten(Kno:TKnoten):TKnoten;
  function KanteistSchlinge:Boolean;
  function KanteistKreiskante:Boolean;
end;

TPfadliste  = class(TListe)
  public
  constructor Create;
  procedure Free;
  procedure Freeall;
  function Pfad(Index:Integer):TPfad;
  property Items[Index:Integer]:TPfad read Pfad;
  function Kopie:TPfadliste;
end;

TPfad   = class(TPfadliste)
  public
  constructor Create;
  procedure Free;
  function Knotenliste:TKnotenliste;
  function Kantenliste:TKantenliste;
  function Pfadlaenge:Extended;
  function PfadSumme(Wert:TWert): Extended;
  function Pfadprodukt (Wert:TWert): Extended;
  function Pfadstring(Sk:TString):string;
  function Pfadstringliste(Sk:TString):TStringlist;
end;

TKnotenliste = class(TListe)
public
  constructor Create;
  procedure Free;
  procedure Freeall;
  function Knoten(Index:Integer):TKnoten;
  property Items[Index:Integer]:TKnoten read Knoten;
  function Kopie:TKnotenliste;
  function Knotenlistealsstring:string;
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 Setzeer
   reicht;
  property Wert:string read Wertlesen write Wertschreiben;
  function Wertlisteschreiben:Tstringlist;virtual;abstract;
  procedure Wertlistelesen;virtual;abstract;
  procedure FueralleausgehendenKanten(Handlung:THandlung);
  procedure FueralleeingehendenKanten(Handlung:THandlung);
  procedure FueralleKanten(Handlung:THandlung);
  procedure FuerjedeausgehendeKante(Vorgang:TVorgang);
  procedure FuerjedeeingehendeKante(Vorgang:TVorgang);
```

```
  procedure FuerjedeKante(Vorgang:TVorgang);
  function Kantenzahlausgehend:Integer;
  function Kantenzahleingehend:Integer;
  function Kantenzahlungerichtet:Integer;
  function Kantenzahl:Integer;
  function AnzahlSchlingen:Integer;
  function Grad(Gerichtet:Boolean):Integer;
  function IstimPfad:Boolean;
  procedure LoeschePfad;
  procedure ErzeugeallePfade;{*}
  procedure ErzeugeallePfadeZielKnoten(Kno:TKnoten);{*}
  function PfadzumZielknoten(Kno:TKnoten;Ka:TKante):Boolean;
  procedure ErzeugeTiefeBaumPfade(Preorder:Boolean);{*}
  procedure ErzeugeWeiteBaumpfade;{*}
  procedure ErzeugeKreise;
  procedure ErzeugeminimalePfade(Wert:TWert);
  procedure ErzeugeminimalePfadenachDijkstra(Wert:TWert);
  procedure SortierePfadliste(Wert:TWert);
  function AnzahlPfadZielknoten: Integer;
  function MinimalerPfad(Wert:TWert):TPfad;
  function MaximalerPfad(Wert:TWert):TPfad;
  function KnotenistKreisknoten:Boolean;
end;

TGraph = class(TObject)
private
  Knotenliste_:TKnotenliste;
  Kantenliste_:TKantenliste;
  Wertposition_:Integer;
  Wertliste_:TStringlist;
  Unterbrechung_:Boolean;
  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);
  procedure SetzeUnterbrechung(Ub:Boolean);
  function WelcheUnterbrechung:Boolean;
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 Abbruch:Boolean read WelcheUnterbrechung write
SetzeUnterbrechung;
property Wert:string read Wertlesen write Wertschreiben;
function Wertlisteschreiben:TStringlist;virtual;abstract;
procedure Wertlistelesen;virtual;abstract;
procedure ImGraphKnotenundKantenloeschen;
function  Leer:Boolean;
procedure KnotenEinfuegen(Kno:TKnoten);
procedure Knotenloeschen(Kno:TKnoten);
procedure KanteEinfuegen
(Ka:TKante;Anfangsknoten,EndKnoten:TKnoten;
Gerichtet:Boolean);
procedure EinfuegenKante(Ka:TKante);
procedure Kanteloeschen(Ka:TKante);
procedure LoescheKantenbesucht;
procedure LoescheKantenerreicht;
procedure LoescheKnotenbesucht;
procedure LoescheKnotenerreicht;
procedure FuerjedenKnoten(Vorgang:TVorgang);
procedure FuerjedeKante(Vorgang:TVorgang);
function Anfangsknoten:TKnoten;
function Anfangskante:TKante;
function AnzahlKanten: Integer;
function AnzahlSchlingen:Integer;
function AnzahlKantenmitSchlingen:Integer;
function AnzahlKnoten:Integer;
function AnzahlgerichteteKanten:Integer;
function AnzahlungerichteteKanten:Integer;
function AnzahlKomponenten:Integer;
function AnzahlparallelerKanten:Integer;
function AnzahlantiparallelerKanten:Integer;
function AnzahlparallelerKantenungerichtet:Integer;
function Kantensumme(Wert:TWert):Extended;
function Kantenprodukt(Wert:TWert):Extended;
function ListemitKnotenInhalt(Sk:TString):TStringlist;
function InhaltallerKnoten(Sk:TString):string;
function ListemitInhaltKantenoderKnoten
(Sk:TString):TStringlist;
function InhaltallerKantenoderKnoten(Sk:TString):string;
procedure Pfadlistenloeschen;
procedure SortiereallePfadlisten(Wert:TWert);
function BestimmeminimalenPfad
(Kno1,Kno2:TKnoten;Wert:TWert):TPfad;
function GraphhatKreise:Boolean;
 function
```

```
    GraphhatgeschlosseneEulerlinie(Gerichtet:Boolean):Boolean;
    function GraphhatoffenenEuerlinie(var Kno1,Kno2:TKnoten;
    Gerichtet:Boolean):Boolean;
    function Kopie:TGraph;
    function KanteverbindetKnotenvonnach
    (Kno1,Kno2:TKnoten):Boolean;
    function ErsteKantevonKnotenzuKnoten
    (Kno1,Kno2:TKnoten):TKante;
    function ErsteSchlingezuKnoten(Kno:TKnoten):TKante;
    function Graphistpaar:Boolean;
    function GraphistBinaerbaum:Boolean;
  end;

implementation

constructor TKantenliste.Create;
begin
  inherited Create;
end;

procedure TKantenliste.Free;
begin
  inherited Free;
end;

procedure TKantenliste.Freeall;
var Index:Integer;
    Ob:TObject;
begin
  if not Leer then
    for Index:=0 to Count-1 do
    begin
      Ob:=Kante(Index);
      if Ob is TKante then TKante(Ob).Free
      else
      if Ob is TGraph then TGraph(Ob).Free;
      Ob:=nil;
    end;
  Free;
end;

function TKantenliste.Kante(Index:Integer):TKante;
begin
  if (Index<=Anzahl-1)and (Index>=0)
  then
    result:=TKante(Tlist(self).Items[Index])
   else
     ShowMessage('Fehler! Listenindex außerhalb des zulässigen
Bereichs');
end;
```

```
function TKantenliste.Kopie:TKantenliste;
var NeueKantenliste:TKantenliste;
    Index:Integer;
begin
  NeueKantenliste:=TKantenliste.Create;
  for Index:=0 to Anzahl-1 do
    NeueKantenliste.AmEndeanfuegen(Kante(Index));
  Kopie:=NeueKantenliste;
end;


function TKantenliste.Graph:TGraph;
var NeuerGraph:TGraph;
    Index:Integer;
    Kno:TKnoten;
begin
  NeuerGraph:=TGraph.Create;
  if not Leer then
  begin
    if Kante(0).Endknoten=Kante(0).Pfadrichtung
    then
      Kno:=Kante(0).Anfangsknoten
    else
      Kno:=Kante(0).Endknoten;
    Neuergraph.Knotenliste.AmEndeanfuegen(Kno);
    for Index:=0 to Anzahl-1 do
    begin
      if Kante(Index).Pfadrichtung<>nil
      then
     NeuerGraph.Knotenliste.AmEndeanfuegen(Kante(Index).Pfadrichtung)
      else
     NeuerGraph.Knotenliste.AmEndeanfuegen(Kante(Index).Endknoten);
    end;
    NeuerGraph.Kantenliste:=self;
  end;
  Graph:=NeuerGraph;
end;


function TKantenliste.UGraph:TGraph;
var NeuerGraph:TGraph;
    Index:Integer;
    Kno:TKnoten;
begin
  if not Leer then
  begin
    NeuerGraph:=TGraph.Create;
   Neuergraph.Knotenliste.AmEndeanfuegen(Kante(0).Anfangsknoten);
    for Index:=0 to Anzahl-1 do
    begin
```

```
     NeuerGraph.Knotenliste.AmEndeanfuegen(Kante(Index).Endknoten);
    end;
    NeuerGraph.Kantenliste:=self;
    UGraph:=NeuerGraph;
  end;
end;


function TKantenliste.Kantenlistealsstring:string;
var Index:Integer;
    Ka:TKante;
    Str:string;
begin
  Str:='';
  if not Leer then
  for Index:=0 to Anzahl-1 do
  begin
    Ka:=Kante(Index);
    if length(Str)+length(Ka.Wert)<254 then
    Str:=Str+Ka.Wert+' ';
  end;
  KantenlistealsString:=Str;
end;


constructor TKante.Create;
begin
  inherited Create;
  Anfangsknoten_:=Nil;
  EndKnoten_:=Nil;
  Pfadrichtung_:=Nil;
  Gerichtet_:=true;
  Besucht_:=false;
  Wertposition_:=0;
  Wertliste_:=Tstringlist.Create;
end;


procedure TKante.Free;
begin
  Wertliste_.Free;
  Wertliste_:=nil;
  inherited Free;
end;

procedure TKante.Freeall;
begin
  Anfangsknoten_.Free;
  Anfangsknoten_:=nil;
  Endknoten_.Free;
```

```
  Endknoten_:=nil;
  Wertliste_.Free;
  Wertliste_:=nil;
  inherited Free;
end;

 function TKante.WelcherAnfangsknoten:TKnoten;
 begin
    WelcherAnfangsknoten:=Anfangsknoten_;
 end;

 procedure TKante.SetzeAnfangsknoten(Kno:TKnoten);
 begin
    Anfangsknoten_:=Kno;
 end;

 function TKante.WelcherEndknoten:TKnoten;
 begin
    WelcherEndknoten:=Endknoten_;
 end;

 procedure TKante.SetzeEndknoten(Kno:TKnoten);
 begin
    Endknoten_:=Kno;
 end;

 function TKante.WelchePfadrichtung:TKnoten;
 begin
    WelchePfadrichtung:=Pfadrichtung_;
 end;

 procedure TKante.SetzePfadrichtung(Kno:TKnoten);
 begin
    Pfadrichtung_:=Kno;
 end;

 function TKante.Istgerichtet:Boolean;
begin
  Istgerichtet:=Gerichtet_;
end;
procedure TKante.Setzegerichtet(Gerichtet:Boolean);
begin
  Gerichtet_:=Gerichtet;
end;

procedure TKante.SetzeWertposition(P:Integer);
begin
  if P>=0 then Wertposition_:=P;
end;
```

```
function TKante.WelcheWertposition:Integer;
begin
  WelcheWertposition:=Wertposition_;
end;

 function TKante.WelcheWertliste:TStringlist;
 begin
   WelcheWertliste:=Wertliste_;
 end;

 procedure TKante.SetzeWertliste(W:TStringlist);
 begin
    Wertliste_:=W;
 end;

function TKante.Istbesucht:boolean;
begin
  Istbesucht:=Besucht_;
end;

procedure TKante.Setzebesucht(B:Boolean);
begin
  Besucht_:=B;
end;

function TKante.Isterreicht:boolean;
begin
  Isterreicht:=Erreicht_;
end;

procedure TKante.Setzeerreicht(E:Boolean);
begin
  Erreicht_:=E;
end;

procedure TKante.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 TKante.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;


function TKante.Zielknoten(Kno:TKnoten):TKnoten;
begin
  Zielknoten:=Endknoten;
  if Kno=Endknoten then Zielknoten:=Anfangsknoten;
end;


function TKante.Quellknoten(Kno:TKnoten):TKnoten;
begin
  Quellknoten:=Anfangsknoten;
  if Kno=Anfangsknoten then Quellknoten:=Endknoten;
end;


function TKante.KanteistSchlinge:Boolean;
begin
  KanteistSchlinge:=(EndKnoten=Anfangsknoten);
end;

function TKante.KanteistKreiskante:Boolean;
var Kreiskante:Boolean;
    Index:Integer;
begin
  Kreiskante:=false;
  if not KanteistSchlinge
  then
  begin
    Anfangsknoten.ErzeugeKreise;
    if not Anfangsknoten.Pfadliste.leer
    then
      for Index:=0 to Anfangsknoten.Pfadliste.Anzahl-1 do
        if TGraph(Anfangsknoten.Pfadliste.Pfad(Index)).
        Knotenliste.ElementistinListe(Endknoten)
        then
          Kreiskante:=true;
  end;
  KanteistKreiskante:=Kreiskante;
end;
```

```
constructor TPfadliste.Create;
begin
  inherited Create;
end;

procedure TPfadliste.Free;
begin
  inherited Free;
end;

procedure TPfadliste.Freeall;
var Index:Integer;
    Ob:TObject;
begin
  if not Leer then
  for Index:=0 to Anzahl-1 do
  begin
    Ob:=Pfad(Index);
    if Ob is TKantenliste
    then
      TKantenliste(Ob).Free
    else
      if Ob is TGraph then TGraph(Ob).Free;
      Ob:=nil;
  end;
  Free;
end;


function TPfadliste.Pfad(Index:Integer):TPfad;
begin
  if (Index<=TList(self).Count-1)and(Index>=0)
  then
    result:=TPfad(TList(self).Items[Index])
   else
    ShowMessage('Fehler Listenindex außerhalb des zulässigen Be
     reichs');
end;


function TPfadliste.Kopie:TPfadliste;
var NeuePfadliste:TPfadliste;
    Index:Integer;
begin
  NeuePfadliste:=TPfadliste.Create;
  for Index:=0 to Anzahl-1 do
    NeuePfadliste.AmEndeanfuegen(TGraph(Pfad(Index)));
  Kopie:=NeuePfadliste;
end;
```

```
constructor TPfad.Create;
begin
  inherited Create;
end;

procedure TPfad.Free;
begin
  inherited Free;
end;

function TPfad.Knotenliste:TKnotenliste;
begin
  Knotenliste:=self.Knotenliste;
end;

function TPfad.Kantenliste:TKantenliste;
begin
  Kantenliste:=self.Kantenliste;
end;

function TPfad.Pfadlaenge:Extended;
var T:TObject;
begin
  T:=self;
  if T is TGraph
  then
    Pfadlaenge:=self.Kantenliste.Anzahl
  else
    Pfadlaenge:=0;
end;


function TPfad.PfadSumme(Wert:TWert):Extended;
var T:TObject;
begin
  T:=self;
  if T is TGraph
  then
    Pfadsumme:=TGraph(self).Kantensumme(Wert)
  else
    Pfadsumme:=0;
end;


function TPfad.Pfadprodukt (Wert:TWert): Extended;
Var T:TObject;
begin
  T:=self;
  if T is TGraph
  then
```

```
    Pfadprodukt:=TGraph(self).Kantenprodukt(Wert)
  else
    Pfadprodukt:=0;
end;


function TPfad.Pfadstring(Sk:TString):string;
begin
  Pfadstring:=TGraph(self).InhaltallerKantenoderKnoten(Sk);
end;

function TPfad.Pfadstringliste(Sk:TString):Tstringlist;
begin
 Pfadstringliste:=TGraph(self).ListemitInhaltKantenoderKnoten(Sk);
end;
constructor TKnotenliste.Create;
begin
  inherited Create;
end;

procedure TKnotenliste.Free;
begin
  inherited Free;
end;

procedure TKnotenliste.Freeall;
var Index:Integer;
    Kno:TKnoten;
begin
  if not Leer then
    for Index:=0 to Anzahl-1 do
    begin
      Kno:=Knoten(Index);
      Kno.Free;
      Kno:=nil;
    end;
  Free;
end;


function TKnotenliste.Knoten(Index:Integer):TKnoten;
begin
  if (Index<=Anzahl-1)and (Index>=0)
  then
    result:=TKnoten(TList(self).Items[Index])
  else
    ShowMessage('Fehler! Listenindex außerhalb des zulässigen
Bereichs');
end;
```

```
function TKnotenliste.Kopie:TKnotenliste;
var NeueKnotenliste:TKnotenliste;
    Index:Integer;
begin
  NeueKnotenliste:=TKnotenliste.Create;
  for Index:=0 to Anzahl-1 do
    NeueKnotenliste.AmEndeanfuegen(Knoten(Index));
  Kopie:=NeueKnotenliste;
end;


function TKnotenliste.Knotenlistealsstring:string;
var Index:Integer;
    Kno:TKnoten;
    Str:string;
begin
  Str:='';
  if not Leer then
  for Index:=0 to Anzahl-1 do
  begin
    Kno:=Knoten(Index);
    if length(Str)+length(Kno.Wert)<254
    then
      Str:=Str+Kno.Wert+' ';
  end;
  KnotenlistealsString:=Str;
end;


constructor TKnoten.Create;
begin
  Graph_:=nil;
  AusgehendeKantenliste_:=TKantenliste.Create;
  EingehendeKantenliste_:=TKantenliste.Create;
  Pfadliste_:=TPfadliste.Create;
  Besucht_:=false;
  Erreicht_:=false;
  WertPosition_:=0;
  Wertliste_:=TStringlist.Create;
end;
procedure TKnoten.Free;
begin
  Wertliste_.Free;
  Wertliste_:=nil;
  AusgehendeKantenliste_.Free;
  AusgehendeKantenliste_:=nil;
  EingehendeKantenliste_.Free;
  EingehendeKantenliste_:=nil;
  Pfadliste_.Free;
  Pfadliste_:=nil;
```

```
  inherited Free;
end;

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;

function TKnoten.WelcherGraph:TGraph;
begin
  WelcherGraph:=Graph_;
end;

procedure TKnoten.SetzeGraph(G:TGraph);
begin
  Graph_:=G;
end;

function TKnoten.WelcheEingehendeKantenliste:TKantenliste;
begin
  WelcheEingehendeKantenliste:=EingehendeKantenliste_;
end;


procedure TKnoten.SetzeEingehendeKantenliste(L:TKantenliste);
begin
  EingehendeKantenliste_:=L;
end;

function TKnoten.WelcheAusgehendeKantenliste:TKantenliste;
begin
```

```
  WelcheAusgehendeKantenliste:=AusgehendeKantenliste_;
end;


procedure TKnoten.SetzeAusgehendeKantenliste(L:TKantenliste);
begin
  AusgehendeKantenliste_:=L;
end;

function TKnoten.WelchePfadliste:TPfadliste;
begin
  WelchePfadliste:=Pfadliste_;
end;

procedure TKnoten.SetzePfadliste(P:TPfadliste);
begin
  Pfadliste_:=P;
end;


procedure TKnoten.SetzeWertposition(P:Integer);
begin
  if P>=0 then Wertposition_:=P;
end;


function TKnoten.WelcheWertposition:Integer;
begin
  WelcheWertposition:=Wertposition_;
end;

function TKnoten.WelcheWertliste:TStringlist;
begin
  WelcheWertliste:=Wertliste_;
end;

procedure TKnoten.SetzeWertliste(W:TStringlist);
begin
  Wertliste_:=W;
end;

function TKnoten.Istbesucht:boolean;
begin
  Istbesucht:=Besucht_;
end;

procedure TKnoten.Setzebesucht(B:boolean);
begin
  Besucht_:=B;
end;
```

```
function TKnoten.Isterreicht:boolean;
begin
  Isterreicht:=Erreicht_;
end;

procedure TKnoten.Setzeerreicht(E:boolean);
begin
  Erreicht_:=E;
end;


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;



procedure TKnoten.FueralleausgehendenKanten(Handlung:THandlung);
var Index:Integer;
begin
  for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
    Handlung(self,AusgehendeKantenliste.Kante(Index));
end;


procedure TKnoten.FueralleeingehendenKanten(Handlung:THandlung);
var Index:Integer;
begin
  for Index:=0 to EingehendeKantenliste.Anzahl-1 do
```

```
    Handlung(self,EingehendeKantenliste.Kante(Index));
end;


procedure TKnoten.FueralleKanten(Handlung:THandlung);
var Index:Integer;
    Ka:TKante;
begin
  if not AusgehendeKantenliste.Leer then
    for   Index:=0 to AusgehendeKantenliste.Anzahl-1 do
    begin
      Ka:=AusgehendeKantenliste.Kante(Index);
      Handlung(self,Ka);
    end;
  if not EingehendeKantenliste.Leer then
    for Index:=0 to EingehendeKantenliste.Anzahl-1 do
    begin
      Ka:=EingehendeKantenliste.Kante(Index);
      if Ka.Gerichtet then
        Handlung(self,Ka);
    end;
end;

procedure TKnoten.FuerjedeausgehendeKante(Vorgang:TVorgang);
var Index:Integer;
begin
  for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
    Vorgang(AusgehendeKantenliste.Kante(Index));
end;


procedure TKnoten.FuerjedeeingehendeKante(Vorgang:TVorgang);
var Index:Integer;
begin
  for Index:=0 to EingehendeKantenliste.Anzahl-1 do
    Vorgang(EingehendeKantenliste.Kante(Index));
end;


procedure TKnoten.FuerjedeKante(Vorgang:TVorgang);
var Index:Integer;
    Ka:TKante;
begin
  if not AusgehendeKantenliste.Leer then
    for Index:=0  to AusgehendeKantenliste.Anzahl-1 do
    begin
      Ka:=AusgehendeKantenliste.Kante(Index);
      Vorgang(Ka);
    end;
  if not EingehendeKantenliste.Leer then
```

```
    for Index:=0 to EingehendeKantenliste.Anzahl-1 do
    begin
      Ka:=EingehendeKantenliste.Kante(Index);
      if Ka.Gerichtet then
        Vorgang(Ka);
    end;
end;

function TKnoten.Kantenzahlausgehend:Integer;
begin
  Kantenzahlausgehend:=AusgehendeKantenliste.Anzahl;
end;

function TKnoten.Kantenzahleingehend:Integer;
begin
  Kantenzahleingehend:=EingehendeKantenliste.Anzahl;
end;


function TKnoten.Kantenzahlungerichtet:Integer;
var Index,Anzahl:Integer;
begin
  Anzahl:=0;
  for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
  if ( not AusgehendeKantenliste.Kante(Index).gerichtet) then
  Anzahl:=Anzahl+1;
  Kantenzahlungerichtet:=Anzahl;
end;


function TKnoten.Kantenzahl:Integer;
begin
  Kantenzahl:=Kantenzahlausgehend+Kantenzahleingehend
  -AnzahlSchlingen-Kantenzahlungerichtet;
end;


function TKnoten.AnzahlSchlingen:Integer;
var Index,Schlingen:Integer;
begin
  Schlingen:=0;
  if not AusgehendeKantenliste.Leer then
    for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
    if AusgehendeKantenliste.Kante(Index).KanteistSchlinge then
      Schlingen:=Schlingen+1;
  AnzahlSchlingen:=Schlingen;
end;

function TKnoten.Grad(Gerichtet:Boolean):Integer;
begin
```

```
  if Gerichtet
  then
    Grad:=Kantenzahlausgehend-Kantenzahleingehend
  else
    Grad:=Kantenzahl;
end;


function TKnoten.IstimPfad:Boolean;
begin
  IstimPfad:=not Pfadliste.Leer;
end;

procedure TKnoten.LoeschePfad;
begin
  Pfadliste.Clear;
end;


procedure TKnoten.ErzeugeallePfade;  {*}
var Index:Integer;
    MomentaneKantenliste:TKantenliste;

  procedure GehezuallenNachbarknoten(Kno:TKnoten;Ka:TKante);
  label Endproc;
  var Ob:TObject;
      Index:Integer;
  begin
    Application.Processmessages;
    if Graph.Abbruch then goto Endproc;
    if not Ka.Zielknoten(Kno).Besucht then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kno);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
        Ka.Zielknoten(Kno).Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
      Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
      if Pfadliste.Anzahl>10000 then
      begin
        ShowMessage('Mehr als 10000 Pfade!Abbruch!');
        Graph.Abbruch:=true;
        goto Endproc;
      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;
    Endproc:
  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;


procedure TKnoten.ErzeugeallePfadeZielknoten(Kno:TKnoten);{*}
var Index:Integer;
    MomentaneKantenliste:TKantenliste;


  procedure GehezuallenNachbarknoten(Kn:TKnoten;Ka:TKante);
  label Endproc;
  var Ob:Tobject;
      Index:Integer;
  begin
    Application.Processmessages;
    if Graph.Abbruch then goto Endproc;
    if not Ka.Zielknoten(Kn).Besucht then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kn);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
      if Ka.Zielknoten(Kn)=Kno then
      begin
    Ka.Zielknoten(Kn).Pfadliste.amEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
        if Kno.Pfadliste.Anzahl>10000 then
        begin
          ShowMessage('Mehr als 10000 Pfade!Abbruch!');
          Graph.Abbruch:=true;
          goto Endproc;
        end;
      end;
      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;
    Endproc:
  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;


function
TKnoten.PfadzumZielknoten(Kno:TKnoten;Ka:TKante):Boolean;
var Index:Integer;
    MomentaneKantenliste:TKantenliste;
     Gefunden:Boolean;

  procedure GehezuallenNachbarknoten(Kn:TKnoten;Kan:TKante);
  label Ende;
  var Ob:Tobject;
      Index:Integer;
  begin
    Application.Processmessages;
    if Graph.Abbruch then goto Ende;
    if Gefunden then goto Ende;
    if Kan=Ka then goto Ende;
    if not Kan.Zielknoten(Kn).Besucht then
    begin
      Kan.Pfadrichtung:=Kan.Zielknoten(Kn);
      MomentaneKantenliste.AmEndeanfuegen(Kan);
      if MomentaneKantenliste.Anzahl>10000 then
      begin
        ShowMessage('Mehr als 10000 Kanten
        durchsucht!Abbruch!');
        Graph.Abbruch:=true;
        goto Ende;
      end;
      if Kan.Zielknoten(Kn)=Kno then
      begin
        PfadzumZielknoten:=true;
        Gefunden:=true;
        goto Ende;
      end;
```

```
      kan.Zielknoten(Kn).Besucht:=true;
      if not Kan.Zielknoten(Kn).AusgehendeKantenliste.Leer then
        for Index:= 0 to
      kan.Zielknoten(Kn).AusgehendeKantenliste.Anzahl-1 do
       GehezuallenNachbarknoten(Kan.Zielknoten(Kn),Kan.Zielknoten(Kn).
          AusgehendeKantenliste.Kante(Index));
      MomentaneKantenliste.AmEndeloeschen(Ob);
      kan.Zielknoten(Kn).Besucht:=false;
    end;
    Ende:
  end;

begin
  Gefunden:=false;
  PfadzumZielknoten:=false;
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.Pfadlistenloeschen;
  LoeschePfad;
  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;
  PfadzumZielknoten:=Gefunden;
end;


procedure TKnoten.ErzeugeTiefeBaumPfade(Preorder:Boolean);{*}
var Index:Integer;
    MomentaneKantenliste:TKantenliste;
    P:TGraph;

  procedure GehezuNachbarknoten(Kno:TKnoten;Ka:TKante);
  label Endproc;
  var Ob:Tobject;
      Index:Integer;
  begin
    Application.Processmessages;
    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
     Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
      if Pfadliste.Anzahl>10000 then
      begin
        ShowMessage('Mehr als 10000 Pfade!Abbruch!');
```

```
        Graph.Abbruch:=true;
        goto Endproc;
      end;
      Ka.Zielknoten(Kno).Pfadliste.AmAnfanganfuegen(MomentaneKantenliste.Kopie.Graph);
      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
      Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
      MomentaneKantenliste.AmEndeloeschen(Ob);
    end;
    Endproc:
  end;


begin
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.Pfadlistenloeschen;
  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;
  P:=TGraph.Create;
  P.Knotenliste.AmEndeanfuegen(self);
  if Preorder
  then
    Pfadliste.AmAnfanganfuegen(P)
  else
    Pfadliste.AmEndeanfuegen(P);
end;


procedure TKnoten.ErzeugeWeiteBaumPfade;{*}
var Ob1,Ob2:TObject;
    Index:Integer;
    Kantenliste:TKantenliste;
    Knotenliste:TKnotenliste;
    MomentaneKantenliste:TKantenliste;
    P:TGraph;


  procedure SpeichereNachbarknoten(Kno:TKnoten;Ka:TKante);
  label Endproc;
```

```
  var Hilfliste:TKantenliste;
      Index:Integer;
  begin
    Application.Processmessages;
    if Graph.Abbruch then goto Endproc;
    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;
      MomentaneKantenliste.AmEndeanfuegen(Ka);
    Ka.Zielknoten(Kno).Pfadliste.AmAnfanganfuegen(MomentaneKantenliste.
      Kopie.UGraph);
     Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.UGraph);
      if Pfadliste.Anzahl>10000 then
      begin
        ShowMessage('Mehr als 10000 Pfade!Abbruch!');
        Graph.Abbruch:=true;
        goto Endproc;
      end;
      Ka.Zielknoten(Kno).Besucht:=true;
      Kantenliste.AmAnfanganfuegen(Ka);
      Knotenliste.AmAnfanganfuegen(Ka.Zielknoten(Kno));
    end;
  Endproc:
end;


begin
  Graph.Pfadlistenloeschen;
  Graph.LoescheKnotenbesucht;
  Besucht:=false;
  Kantenliste:=TKantenliste.Create;
  Knotenliste:=TKnotenliste.Create;
  Besucht:=true;
  P:=TGraph.Create;
  P.Knotenliste.AmEndeanfuegen(self);
  Pfadliste.AmAnfanganfuegen(P);
  if not AusgehendeKantenliste.Leer then
    for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
    SpeichereNachbarknoten(self,AusgehendeKantenliste.Kante(Index));
  while not Knotenliste.Leer do
  begin
    Kantenliste.AmEndeloeschen(Ob1);
    Knotenliste.AmEndeloeschen(Ob2);
   TKante(Ob1).Pfadrichtung:=TKante(Ob1).Zielknoten(TKnoten(Ob2));
```

```
    if not TKnoten(Ob2).AusgehendeKantenliste.Leer then
      for Index:=0 to TKnoten(Ob2).AusgehendeKantenliste.
       Anzahl-1 do
      SpeichereNachbarknoten(TKante(Ob1).Zielknoten(Tknoten(Ob2)),
    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;


procedure TKnoten.ErzeugeKreise;
var Index:Integer;
    MomentaneKantenliste:TKantenliste;

  procedure GehezudenNachbarknoten(Kno:TKnoten;Ka:TKante);
  label Endproc;
  var Ob:Tobject;
      Index:Integer;

  begin
    Application.Processmessages;
    if Graph.Abbruch then goto Endproc;
    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);
        if Pfadliste.Anzahl>10000 then
        begin
          ShowMessage('Mehr als 10000 Kreise!Abbruch!');
          Graph.Abbruch:=true;
          goto Endproc;
        end;
        MomentaneKantenliste.AmEndeloeschen(Ob);
      end;
    Endproc:
  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;


procedure TKnoten.ErzeugeminimalePfade(Wert:TWert);
var Index,Index1:Integer;
    HilfKantenliste:TKantenliste;
    MomentaneKantenliste:TKantenliste;

  procedure GehezuallenNachbarn(Kno:TKnoten;Ka:TKante);
  label Endproc;
  var Ob:TObject;
      Index:Integer;
  begin
    Application.Processmessages;
    if Graph.Abbruch then goto Endproc;
    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;
    Endproc:
  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
  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(Wert)).Kantenliste;
        Pfadliste.AmEndeanfuegen(HilfKantenliste.Kopie.Graph);
        Hilfkantenliste.Free;
        Hilfkantenliste:=nil;
      end;
  end;
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
end;

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;


procedure TKnoten.ErzeugeminimalePfadenachDijkstra(Wert:TWert);
label Endproc;
var WegPfadliste:TPfadliste;
     MomentanerWeg,MomentanerWegneu:TKantenliste;
     Index1,Index2: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 Index1:=0 to AusgehendeKantenliste.Anzahl-1  do
    begin
      Ka:=AusgehendeKantenliste.Kante(Index1);
      MomentanerWeg:=TKantenliste.Create;
      Ka.Pfadrichtung:=Ka.Zielknoten(self);
      MomentanerWeg.AmAnfanganfuegen(Ka);
      WegPfadliste.AmAnfanganfuegen(Momentanerweg.Graph);
    end;
  WegPfadliste.Sortieren(Pfadvergleich,Wert);
  while not Wegpfadliste.Leer  do
  begin
    Application.ProcessMessages;
    if Graph.Abbruch then goto Endproc;
    Wegpfadliste.AmEndeloeschen(Ob);
    Momentanerweg:=TGraph(Ob).Kantenliste;
    Ka1:=MomentanerWeg.Kante(MomentanerWeg.Letztes);
    Kno:=Ka1.Pfadrichtung;
    if not Kno.Besucht
    then
    begin
      Kno.Besucht:=true;
      Kno.Pfadliste.AmEndeanfuegen(MomentanerWeg.Kopie.Graph);
      Pfadliste.AmEndeanfuegen(MomentanerWeg.Kopie.Graph);
      if not Kno.AusgehendeKantenliste.Leer then
        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);
            if not Ka2.Pfadrichtung.Besucht then
            begin
              MomentanerWegneu:=TKantenliste.Create;
              MomentanerWegneu:=MomentanerWeg.Kopie;
              MomentanerWegneu.AmEndeanfuegen(Ka2);
           WegPfadliste.AmAnfanganfuegen(Momentanerwegneu.Graph);
            end;
          end;
        end;
    end
    else
      begin
        MomentanerWeg.Free;
```

```
        MomentanerWeg:=nil;
      end;

    WegPfadliste.Sortieren(Pfadvergleich,Wert);
  end;
  Endproc:
  Wegpfadliste.Freeall;
  Wegpfadliste:=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);
end;


procedure TKnoten.SortierePfadliste(Wert:TWert);
begin
  Pfadliste.Sortieren(KleinererPfad,Wert);
end;


function Wertung(Ob:TObject):Extended;
begin
  Wertung:=1
end;

function TKnoten.AnzahlPfadZielknoten: Integer;
begin
  Graph.Pfadlistenloeschen;
  ErzeugeTiefeBaumpfade(true);
  AnzahlPfadZielknoten:=Pfadliste.Anzahl-1;
end;


function TKnoten.MinimalerPfad(Wert:TWert):TPfad;
var Hilfliste:TPfadliste;
begin
  Hilfliste:=Pfadliste.Kopie;
  if not Hilfliste.Leer
  then
  begin
    Hilfliste.Sortieren(KleinererPfad,Wert);
    MinimalerPfad:=Hilfliste.Pfad(0);
  end
```

```
  else
    MinimalerPfad:=TPfad(TGraph.Create);
end;


function TKnoten.MaximalerPfad(Wert:TWert):TPfad;
var Hilfliste:TPfadliste;
begin
  Hilfliste:=Pfadliste.Kopie;
  if not Hilfliste.Leer
  then
  begin
    Hilfliste.Sortieren(KleinererPfad,Wert);
    MaximalerPfad:=Hilfliste.Pfad(Hilfliste.Anzahl-1);
  end
  else
     MaximalerPfad:=TPfad(TGraph.Create);
end;

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
    Application.Processmessages;
    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 AusgehendeKantenliste.Leer then
  for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
  begin
   GehezudenNachbarknoten(self,AusgehendeKantenliste.Kante(Index));
    if Gefunden then goto Endproc;
  end;
  Endproc:
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
  KnotenistKreisknoten:=Gefunden;
end;


constructor TGraph.Create;
begin
  inherited Create;
  Knotenliste_:=TKnotenliste.Create;
  Kantenliste_:=TKantenliste.Create;
  Wertposition_:=-1;
  Wertliste_:=TStringlist.Create;
end;

procedure TGraph.Free;
begin
  Knotenliste_.Free;
  Knotenliste_:=nil;
  Kantenliste_.Free;
  Kantenliste_:=nil;
  Wertliste_.Free;
  Wertliste_:=nil;
  inherited Free;
end;


procedure TGraph.Freeall;
begin
  Kantenliste_.Freeall;
  Kantenliste_:=nil;
  Knotenliste_.Freeall;
```

```
  Knotenliste_:=nil;
  Wertliste_.Free;
  Wertliste_:=nil;
  inherited Free;
end;


function TGraph.WelcheKnotenliste:TKnotenliste;
begin
   WelcheKnotenliste:=Knotenliste_;
end;

procedure TGraph.SetzeKnotenliste(K:TKnotenliste);
begin
   Knotenliste_:=K;
end;

function TGraph.WelcheKantenliste:TKantenliste;
begin
   WelcheKantenliste:=Kantenliste_;
end;

procedure TGraph.SetzeKantenliste(K:TKantenliste);
begin
   Kantenliste_:=K;
end;

 function TGraph.WelcheWertliste:TStringlist;
 begin
   WelcheWertliste:=Wertliste_;
 end;

 procedure TGraph.SetzeWertliste(W:TStringlist);
 begin
    Wertliste_:=W;
 end;

function TGraph.WelcheWertposition:Integer;
begin
  WelcheWertposition:=Wertposition_;
end;

procedure TGraph.SetzeWertposition(P:Integer);
begin
  if P>=0 then Wertposition_:=P;
end;


function TGraph.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;


procedure TGraph.Wertschreiben(S:string);
begin
  if (Position>=0) and (Position<=Wertliste.count-1)
  then
    Wertliste.Strings[Position]:=S
  else
    ShowMessage('Fehler Wertposition Wertlesen');
  Wertlistelesen;
end;


procedure TGraph.SetzeUnterbrechung(Ub:boolean);
begin
  Unterbrechung_:=Ub;
end;

function TGraph.WelcheUnterbrechung;
begin
  WelcheUnterbrechung:=Unterbrechung_;
end;

procedure TGraph.ImGraphKnotenundKantenloeschen;
begin
  Kantenliste_.Freeall;
  Knotenliste_.Freeall;
  Wertliste_.Free;
  Kantenliste_:=TKantenliste.Create;
  Knotenliste_:=TKnotenliste.Create;
  Wertliste_:=Tstringlist.Create;
end;

function TGraph.Leer:Boolean;
begin
  Leer:=Knotenliste.Leer;
end;
```

```
procedure TGraph.KnotenEinfuegen(Kno:TKnoten);
begin
  Knotenliste.AmEndeanfuegen(Kno);
  Kno.Graph:=self;
end;


procedure TGraph.Knotenloeschen(Kno:TKnoten);
var Index:Integer;
    Ka:TKante;
begin
  if not Kno.AusgehendeKantenliste.Leer then
    for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
    begin
      Ka:=Kno.AusgehendeKantenliste.Kante(Index);
      if not Ka.KanteistSchlinge then
      begin
      Ka.Zielknoten(Kno).EingehendeKantenliste.LoescheElement(Ka);
      if not Ka.gerichtet then Ka.Zielknoten(Kno).
      AusgehendeKantenliste.LoescheElement(Ka);
      end;
    Kantenliste.LoescheElement(Ka);
  end;
  if not Kno.EingehendeKantenliste.Leer then
    for Index:=0 to Kno.EingehendeKantenliste.Anzahl-1 do
    begin
      Ka:=Kno.EingehendeKantenliste.Kante(Index);
      if not Ka.KanteistSchlinge then
      begin
      Ka.Quellknoten(Kno).AusgehendeKantenliste.LoescheElement(Ka);
        if not Ka.gerichtet then
       Ka.Quellknoten(Kno).EingehendeKantenliste.
        loescheElement(Ka);
      end;
      Kantenliste.LoescheElement(Ka);
    end;
  self.Knotenliste.LoescheElement(Kno);
  Kno.Freeall;
  Kno:=nil;
end;


procedure
TGraph.KanteEinfuegen(Ka:TKante;Anfangsknoten,EndKnoten:TKnoten;
  Gerichtet:Boolean);
label Ende;
begin
  if (Anfangsknoten=nil) or (Endknoten=nil)or(Ka=nil) then goto
Ende;
  Ka.Anfangsknoten:=Anfangsknoten;
```

```
  Ka.Endknoten:=Endknoten;
  Anfangsknoten.AusgehendeKantenliste.AmEndeanfuegen(Ka);
  if Gerichtet then
  begin
    EndKnoten.EingehendeKantenliste.AmEndeanfuegen(Ka);
    Ka.Gerichtet:=true;
  end
  else
  begin
    Ka.Gerichtet:=false;
    Anfangsknoten.EingehendeKantenliste.AmEndeanfuegen(Ka);
    if  Anfangsknoten<>Endknoten then
    begin
      EndKnoten.AusgehendeKantenliste.AmEndeanfuegen(Ka);
      Endknoten.EingehendeKantenliste.AmEndeanfuegen(Ka);
    end;
  end;
  Ka.Anfangsknoten:=Anfangsknoten;
  Ka.EndKnoten:=EndKnoten;
  Kantenliste.amEndeanfuegen(Ka);
  Ende:
end;


procedure TGraph.EinfuegenKante(Ka:TKante);
label Ende;
begin
  If (Ka=nil) or (Ka.Anfangsknoten=nil) or (Ka.Endknoten=nil)
then goto Ende;
  Knoteneinfuegen(Ka.Anfangsknoten);
  Knoteneinfuegen(Ka.EndKnoten);
  Ka.Anfangsknoten.AusgehendeKantenliste.AmEndeanfuegen(Ka);
  Ka.Endknoten.EingehendeKantenliste.AmEndeanfuegen(Ka);
  if not Ka.Gerichtet then
  if Ka.Anfangsknoten<>Ka.Endknoten then
  begin
    Ka.Endknoten.AusgehendeKantenliste.AmEndeanfuegen(Ka);
    Ka.Anfangsknoten.EingehendeKantenliste.AmEndeanfuegen(Ka);
  end;
  Kantenliste.AmEndeanfuegen(Ka);
  Ende:
end;


procedure TGraph.Kanteloeschen(Ka:TKante);
begin
  Kantenliste.LoescheElement(Ka);
  Ka.Anfangsknoten.AusgehendeKantenliste.LoescheElement(Ka);
  if Ka.gerichtet
  then
```

```
    Ka.EndKnoten.EingehendeKantenliste.LoescheElement(Ka)
  else
  begin
    if Ka.Anfangsknoten<>Ka.Endknoten then
    begin
      Ka.EndKnoten.AusgehendeKantenliste.LoescheElement(Ka);
      Ka.EndKnoten.EingehendeKantenliste.LoescheElement(Ka);
    end;
      Ka.Anfangsknoten.EingehendeKantenliste.LoescheElement(Ka);
  end;
  Ka.Free;
  Ka:=nil;
end;


procedure TGraph.LoescheKantenbesucht;
var Index:Integer;
begin
  for Index:=0 to Kantenliste.Anzahl-1 do
    Kantenliste.Kante(Index).Besucht:=false;
end;


procedure TGraph.LoescheKantenerreicht;
var Index:Integer;
begin
  for Index:=0 to Kantenliste.Anzahl-1 do
    Kantenliste.Kante(Index).Erreicht:=false;
end;



procedure TGraph.LoescheKnotenbesucht;
var Index:Integer;
begin
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      Knotenliste.Knoten(Index).Besucht:=false;
end;


procedure TGraph.LoescheKnotenerreicht;
var Index:Integer;
begin
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      Knotenliste.Knoten(Index).Erreicht:=false;
end;
```

```
procedure TGraph.FuerjedenKnoten(Vorgang:TVorgang);
begin
  Knotenliste.FuerjedesElement(Vorgang);
end;


procedure TGraph.FuerjedeKante(Vorgang:TVorgang);
begin
  Kantenliste.FuerjedesElement(Vorgang);
end;

function TGraph.Anfangsknoten:TKnoten;
begin
  if not Knotenliste.Leer
  then
  begin
    Anfangsknoten:=Knotenliste.Knoten(0);
    Anfangsknoten.Graph:=self;
  end
  else
    Anfangsknoten:=nil;
end;

function TGraph.Anfangskante:TKante;
begin
  if not Kantenliste.Leer
  then
    Anfangskante:=Kantenliste.Kante(0)
  else
    Anfangskante:=nil;
end;


function TGraph.AnzahlKanten:Integer;
begin
  AnzahlKanten:=Kantenliste.Anzahl-AnzahlSchlingen;
end;


function TGraph.AnzahlSchlingen:Integer;
Var Graphschlingen:Integer;
    Index:Integer;
    Ka:TKante;

begin
  Graphschlingen:=0;
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
    begin
      Ka:=Kantenliste.Kante(Index);
```

```
      if Ka.KanteistSchlinge then
        Graphschlingen:=Graphschlingen+1;
    end;
  AnzahlSchlingen:=Graphschlingen;
end;



function TGraph.AnzahlKantenmitSchlingen:Integer;
begin
  AnzahlKantenmitSchlingen:=AnzahlKanten+AnzahlSchlingen;
end;


function TGraph.AnzahlKnoten:Integer;
begin
  AnzahlKnoten:=Knotenliste.Anzahl;
end;


function TGraph.AnzahlgerichteteKanten:Integer;
var Index,Anzahl:Integer;
begin
  Anzahl:=0;
  if not Kantenliste.Leer then
  for Index:=0 to Kantenliste.Anzahl-1 do
    if Kantenliste.Kante(Index).gerichtet then Anzahl:=Anzahl+1;
  AnzahlgerichteteKanten:=Anzahl;
end;

function TGraph.AnzahlungerichteteKanten:Integer;
begin
  AnzahlungerichteteKanten:=
  AnzahlKantenmitSchlingen-AnzahlgerichteteKanten;
end;


procedure BesucheZielknoten(Y,X:Tobject);
var Zkno:Tknoten;
    Ykno:TKnoten;
begin
  Ykno:=TKnoten(Y);
  Zkno:=TKnoten(TKante(X).Zielknoten(Ykno));
  if not Zkno.Besucht then
  begin
    Zkno.Besucht:=true;
    Zkno.FueralleKanten(BesucheZielknoten);
  end;
end;
```

```
function TGraph.AnzahlKomponenten:Integer;
var Komponentenzahl,Index:Integer;
begin
  Komponentenzahl:=0;
  LoescheKnotenbesucht;
  for Index:=0 to Knotenliste.Anzahl-1 do
    if not Knotenliste.Knoten(Index).Besucht then
    begin
      Komponentenzahl:=Komponentenzahl+1;
      Knotenliste.Knoten(Index).Besucht:=true;
      Knotenliste.Knoten(Index).
      FueralleKanten(BesucheZielknoten);
    end;
  AnzahlKomponenten:=Komponentenzahl;
end;

function TGraph.AnzahlparallelerKanten:Integer;
var Kna1,Kna2:TKante;
    Zaehl1,Zaehl2,Anzahl:Integer;
begin
  Anzahl:=0;
  LoescheKantenbesucht;
  if not Kantenliste.Leer
  then
  begin
    for Zaehl1:=0 to Kantenliste.Anzahl-1 do
    begin
      Kna1:=Kantenliste.Kante(Zaehl1);
      for Zaehl2:=0 to Kantenliste.Anzahl-1 do
      begin
        Kna2:=Kantenliste.Kante(Zaehl2);
        if Kna1.Gerichtet  and Kna2.Gerichtet and (Kna1<>Kna2)
        and (not Kna1.Besucht) and (not Kna2.Besucht)
        then
        begin
          if (Kna1.Anfangsknoten=Kna2.Anfangsknoten) and
          (kna1.Endknoten=Kna2.Endknoten)
          then
          begin
            Anzahl:=Anzahl+1;
            kna1.Besucht:=true;
            Kna2.Besucht:=true;
          end;
        end;
      end;
    end;
  end;
  AnzahlparallelerKanten:=Anzahl;
end;
```

```
function TGraph.AnzahlantiparallelerKanten:Integer;
var Kna1,Kna2:TKante;
    Zaehl1,Zaehl2,Anzahl:Integer;
begin
  Anzahl:=0;
  LoescheKantenbesucht;
  if not Kantenliste.Leer then
  begin
  for Zaehl1:=0 to Kantenliste.Anzahl-1 do
  begin
    Kna1:=Kantenliste.Kante(Zaehl1);
    for Zaehl2:=0 to Kantenliste.Anzahl-1 do
    begin
      Kna2:=Kantenliste.Kante(Zaehl2);
        if Kna1.Gerichtet  and Kna2.Gerichtet and (Kna1<>Kna2)
        and (not Kna1.Besucht) and (not Kna2.Besucht)
        then
        begin
          if (Kna1.Anfangsknoten=Kna2.Endknoten)and
          (Kna1.Endknoten=Kna2.Anfangsknoten)
          then
          begin
            Anzahl:=Anzahl+1;
            Kna1.Besucht:=true;
            Kna2.Besucht:=true;
          end;
        end;
      end;
    end;
  end;
  AnzahlantiparallelerKanten:=Anzahl;
end;


function TGraph.AnzahlparallelerKantenungerichtet:Integer;
var kna1,Kna2:TKante;
    Zaehl1,Zaehl2,Anzahl:Integer;
begin
  Anzahl:=0;
  LoescheKantenbesucht;
  if not Kantenliste.Leer then
  begin
    for Zaehl1:=0 to Kantenliste.Anzahl-1 do
    begin
      Kna1:=Kantenliste.Kante(Zaehl1);
      for Zaehl2:=0 to Kantenliste.Anzahl-1 do
      begin
        Kna2:=Kantenliste.Kante(Zaehl2);
        if (Kna1<>Kna2) and (not Kna1.Besucht) and (not
         Kna2.Besucht)
```

```
        then
        begin
          if ((Kna1.Anfangsknoten=Kna2.Anfangsknoten)and
          (Kna1.Endknoten=Kna2.Endknoten))or
      ((Kna1.Anfangsknoten=Kna2.Endknoten)and(Kna1.Endknoten=Kna2.Anfangsknoten))
          then
          begin
            Kna1.Besucht:=true;
            Kna2.Besucht:=true;
            Anzahl:=Anzahl+1;
          end;
        end;
      end;
    end;
  end;
  AnzahlparallelerKantenungerichtet:=Anzahl;
end;



function TGraph.Kantensumme(Wert:TWert):Extended;
begin
  Kantensumme:=Kantenliste.WertsummederElemente(Wert);
end;

function TGraph.Kantenprodukt(Wert:TWert):Extended;
begin
  Kantenprodukt:=Kantenliste.WertproduktderElemente(Wert);
end;

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;



function TGraph.InhaltallerKnoten(Sk:TString):string;
var Stringliste:TStringlist;
    St:string;
    Zaehl:Integer;
begin
  St:='';
  Stringliste:=ListemitKnotenInhalt(Sk);
  if Stringliste<>nil then
```

```
  for Zaehl:=0 to Stringliste.Count-1 do
  begin
    St:=St+' '+Stringliste.Strings[Zaehl];
  end;
  InhaltallerKnoten:=St;
end;

function TGraph.ListemitInhaltKantenoderKnoten
(Sk:TString):TStringlist;
var Stringliste:TStringlist;
    Index:Integer;
begin
  Stringliste:=TStringlist.Create;
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
      Stringliste.Add(Sk(Kantenliste.Kante(Index)));
  ListemitInhaltKantenoderKnoten:=Stringliste;
end;

function TGraph.InhaltallerKantenoderKnoten(Sk:Tstring):string;
var Stringliste:TStringlist;
    St:String;
    Zaehl:Integer;
begin
  St:='';
  Stringliste:=ListemitInhaltKantenoderKnoten(Sk);
  If Stringliste<>nil then
    for Zaehl:=0 to Stringliste.Count-1 do
    begin
      St:=St+' '+Stringliste.Strings[Zaehl];
    end;
  InhaltallerKantenoderKnoten:=St;
end;

procedure TGraph.Pfadlistenloeschen;
var Index:Integer;
begin
  if not Knotenliste.Leer then
  for Index:=0 to Knotenliste.Anzahl-1 do
    if  not Knotenliste.Knoten(Index).Pfadliste.Leer
    then
      Knotenliste.Knoten(Index).LoeschePfad;
end;


procedure TGraph.SortiereallePfadlisten(Wert:Twert);
var Index:Integer;
begin
  if not Knotenliste.Leer then
    for Index:=1 to Knotenliste.Anzahl-1 do
```

```
      if not Knotenliste.Knoten(Index).Pfadliste.Leer then
        Knotenliste.Knoten(Index).SortierePfadliste(Wert);
end;


function TGraph.
BestimmeminimalenPfad(Kno1,Kno2:TKnoten;Wert:TWert):TPfad;
label Endproc;
var MomentaneKantenliste,KaWeg:TKantenliste;
    Index,Index1:Integer;
    Di,Dj,D:Extended;
    Kno,ZKno:TKnoten;
    Ka:TKante;
    Negativ:boolean;

function BesuchtMarkierung:Boolean;
var Zaehl:Integer;
begin
  BesuchtMarkierung:=false;
  if not Knotenliste.leer then
  for Zaehl:=0 to Knotenliste.Anzahl-1 do
  if Knotenliste.Knoten(Zaehl).Besucht
  then
    BesuchtMarkierung:=true;
end;

begin
  Pfadlistenloeschen;MomentaneKantenliste:=nil;
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    Knotenliste.Knoten(Index).Pfadliste.AmEndeAnfuegen(TGraph.create);
  Negativ:=false;
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
      if StringtoReal(Kantenliste.Kante(Index).Wert)<0 then
Negativ:=true;
  if Negativ and (AnzahlungerichteteKanten>0)
  then
  begin
    ShowMessage('Der Graph enthält ungerichtete und negativ be-
wertete Kanten!Fehler!');MomentaneKantenliste.Free;
    MomentaneKantenliste:=nil;goto Endproc;
  end;
  LoescheKnotenbesucht;
  if not Kno1.AusgehendeKantenliste.Leer
  then
  for Index:=0 to Kno1.AusgehendeKantenliste.Anzahl-1 do
  begin
    Ka:=Kno1.AusgehendeKantenliste.Kante(Index);
    ZKno:=Ka.Zielknoten(Kno);
```

```
  if (not Ka.KanteistSchlinge) and (ZKno<>Kno1)
  then
  begin
    Ka.Pfadrichtung:=ZKno;
    MomentaneKantenliste:=TKantenliste.Create;
    MomentaneKantenliste.AmEndeAnfuegen(Ka);
    ZKno.LoeschePfad;
    ZKno.Pfadliste.AmEndeAnfuegen(MomentaneKantenliste.Graph);
    ZKno.Besucht:=true;
  end;
end;
while Besuchtmarkierung do
begin
  if not Knotenliste.Leer then
  for Index:=0 to Knotenliste.Anzahl-1 do
  begin
    Application.ProcessMessages;
    if Abbruch then goto Endproc;
    Kno:=Knotenliste.Knoten(Index);
    if Kno.Besucht then
    begin
      Di:=Kno.Pfadliste.Pfad(0).Pfadsumme(Wert);
      if not Kno.AusgehendeKantenliste.leer then
      for Index1:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
      begin
    MomentaneKantenliste:=TGraph(Kno.Pfadliste.Pfad(0)).Kantenliste.Kopie;
        if MomentaneKantenliste.Anzahl>=AnzahlKanten then
        begin
          ShowMessage('Pfad länger als Kantenzahl!Negativer
          Kreis!');
          Kno2.LoeschePfad;
          goto Endproc;
        end;
        Ka:=Kno.AusgehendeKantenliste.Kante(Index1);
        ZKno:=Ka.Zielknoten(Kno);
        if (not Ka.KanteistSchlinge) and (ZKno<>Kno1)
        then
        begin
          KaWeg:=TKantenliste.Create;
          KaWeg.AmEndeAnfuegen(Ka);
          D:=KaWeg.WertsummederElemente(Wert);
          KaWeg.Free;
          Ka.Pfadrichtung:=ZKno;
          Dj:=ZKno.Pfadliste.Pfad(0).Pfadsumme(Wert);
          if (Dj>Di+D) or
          (TGraph(ZKno.Pfadliste.Pfad(0)).Leer) then
          begin
            ZKno.Besucht:=true;
            MomentaneKantenliste.AmEndeAnfuegen(Ka);
            ZKno.LoeschePfad;
```

```
          ZKno.Pfadliste.AmEndeAnfuegen(MomentaneKantenliste.Graph.Kopie);
             end
           end;
         end;
       Kno.Besucht:=false;
       end;
    end;
  end;if MomentaneKantenliste<>nil then
 begin MomentaneKantenliste.Free;MomentaneKantenliste:=nil;end;  end:
  Endproc:if not Kno2.Pfadliste.leer
  then
    BestimmeminimalenPfad:=Kno2.Pfadliste.Pfad(0)
  else
    BestimmeminimalenPfad:=TPfad(TGraph.create);
end;

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
    Application.ProcessMessages;
    Gefunden:=Knotenliste.Knoten(Index).KnotenistKreisknoten;
    if Gefunden then goto Endproc;
  end;
  Endproc:
  GraphhatKreise:=Gefunden;
end;


function TGraph.GraphhatgeschlosseneEulerlinie
(Gerichtet:Boolean):Boolean;
var IstEuler:Boolean;
    Index,Kantenzahl:Integer;
begin
  IstEuler:=true;
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kantenzahl:=Knotenliste.Knoten(Index).Kantenzahl
      -Knotenliste.Knoten(Index).AnzahlSchlingen;
      if (not Gerichtet) and odd(Kantenzahl) then
       IstEuler:=false;
      if gerichtet and (Knotenliste.Knoten(Index).Grad(true)<>0)
       then IstEuler:=false;
    end;
```

```
  GraphhatgeschlossenenEulerlinie:=IstEuler;
end;


function TGraph.GraphhatoffeneEulerlinie(var Kno1,Kno2:TKnoten;
  Gerichtet:Boolean):Boolean;
var ErsterKnotengefunden,ZweiterKnotengefunden,IstEuler:Boolean;
    Zaehler,Kantenzahl,Index:Integer;Hilf:TKnoten;
begin
  Zaehler:=0;
  ErsterKnotengefunden:=false;
  ZweiterKnotengefunden:=false;
  if not Knotenliste.Leer then
  begin
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kantenzahl:=Knotenliste.Knoten(Index).Kantenzahl
      -Knotenliste.Knoten(Index).AnzahlSchlingen;
      if (not Gerichtet) and odd(Kantenzahl) then
      Zaehler:=Zaehler+1;
      if Gerichtet and (Knotenliste.Knoten(Index).Grad(true)<>0)
      then Zaehler:=Zaehler+1;
      if (Zaehler=1)and (not ErsterKnotengefunden) then
      begin
        Kno1:=Knotenliste.Knoten(Index);
        ErsterKnotengefunden:=true;
      end;
      if (Zaehler=2)and (not ZweiterKnotengefunden) then
      begin
        Kno2:=Knotenliste.Knoten(Index);
        ZweiterKnotengefunden:=true;
      end;
    end;
  end;
  if Zaehler=2
  then begin if Kno1.Grad(true)<Kno2.Grad(true) then begin Hilf:=Kno1;
    Kno1:=Kno2;Kno2:=Hilf;end;IstEuler:=true;end
  else
  begin
    IstEuler:=false;
    Kno1:=nil;
    Kno2:=nil;
  end;
  GraphhatoffeneEulerlinie:=IstEuler;
end;


function TGraph.Kopie:TGraph;
var HilfKnotenliste:TKnotenliste;
    HilfKantenliste:TKantenliste;
```

```
    Gra:TGraph;
    Index:Integer;
begin
  Gra:=TGraph.Create;
  Hilfknotenliste:=TKnotenliste.Create;
  Hilfkantenliste:=TKantenliste.Create;
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1  do
      Hilfknotenliste.AmEndeanfuegen(Knotenliste.Knoten(Index));
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
      Hilfkantenliste.AmEndeanfuegen(Kantenliste.Kante(Index));
  Gra.Knotenliste:=Hilfknotenliste;
  Gra.Kantenliste:=HilfKantenliste;
  Kopie:=Gra;
end;


function TGraph.KanteverbindetKnotenvonnach
(Kno1,Kno2:TKnoten):Boolean;
var Zaehler:Integer;
    Verbunden:boolean;
begin
  Verbunden:=false;
  if not Kno1.AusgehendeKantenliste.Leer then
    for zaehler:=0 to Kno1.AusgehendeKantenliste.Anzahl-1 do
       if
Kno1.AusgehendeKantenliste.Kante(zaehler).Zielknoten(Kno1)=Kno2
       then
         Verbunden:=true;
  KanteverbindetKnotenvonnach:=Verbunden;
end;


function TGraph.ErsteKantevonKnotenzuKnoten
(Kno1,Kno2:TKnoten):TKante;
var Index:Integer;
     Ka,Kant:TKante;
begin
  Ka:=nil;
  if not Kno1.AusgehendeKantenliste.Leer then
    for Index:=Kno1.AusgehendeKantenliste.Anzahl-1 downto 0 do
    begin
      Kant:=Kno1.AusgehendeKantenliste.Kante(Index);
      if Kant.Endknoten=Kno2 then Ka:=Kant;
    end;
  ErsteKantevonKnotenzuKnoten:=Ka;
end;

function TGraph.ErsteSchlingezuKnoten(Kno:TKnoten):TKante;
```

```
var Index:Integer;
     Ka,Kant:TKante;
begin
  Ka:=Nil;
  if not Kno.AusgehendeKantenliste.Leer then
    for Index:=Kno.AusgehendeKantenliste.Anzahl-1 downto 0 do
    begin
      Kant:=Kno.AusgehendeKantenliste.Kante(Index);
      if Kant.Endknoten=Kno then Ka:=Kant;
    end;
  ErsteSchlingezuKnoten:=Ka;
end;


function TGraph.Graphistpaar:Boolean;
var Index:Integer;
    Kno:TKnoten;
    Paar:Boolean;
begin
  Paar:=true;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=Knotenliste.Knoten(Index);
      if (Kno.AusgehendeKantenliste.Leer)and
      (Kno.EingehendeKantenliste.Leer)
      then
        Paar:=false;
       if (not Kno.AusgehendeKantenliste.Leer)and (not
       Kno.EingehendeKantenliste.Leer)
       then
         Paar:=false;
    end;
  Graphistpaar:=Paar;
end;

function TGraph.GraphistBinaerbaum:Boolean;
var Index,Zahl:Integer;
    Kno:TKnoten;
begin
  GraphistBinaerBaum:=true;
  for Index:=0 to Knotenliste.Anzahl-1 do
  begin
    Kno:=Knotenliste.Knoten(Index);
    Zahl:= Kno.AusgehendeKantenliste.Anzahl;
    if Zahl>2
    then
      GraphistBinaerbaum:=false;
    if (Zahl=1) and (Kno.AusgehendeKantenliste.
    Kante(0).Zielknoten(Kno).AusgehendeKantenliste.Anzahl>0)
```

```
    then
      GraphistBinaerbaum:=false;
    Zahl:=Knotenliste.Knoten(Index).EingehendeKantenliste.Anzahl;
    if Zahl>1
    then
      GraphistBinaerbaum:=false;
  end;
end;


end.
```

Unit UInhgrph

```
unit UInhGrph;
{$F+}
interface

uses

  UList,UGraph,UKante,
  Sysutils,Graphics,Wintypes,WinProcs,Dialogs,Menus,Forms,
  Messages, Classes,StdCtrls, Controls;



type


  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 Setze
   radius;
  property Farbe:TColor read WelcheFarbe write SetzeFarbe;
  property Stil:TPenstyle read WelcherStil write SetzeStil;
  property Typ:char read WelcherTyp write SetzeTyp;
  procedure ZeichneKnoten(Flaeche:TCanvas);
  procedure ZeichneDruckKnoten
  (Flaeche:TCanvas;Faktor:Integer);
  procedure Knotenzeichnen
  (Flaeche:TCanvas;Demo:Boolean;Pausenzeit:Integer);
  procedure AnzeigePfadliste(Flaeche:TCanvas;Ausgabe:TLabel;
  var SListe:TStringList;Zeichnen:Boolean;
  LetzterPfad:Boolean);
  function ErzeugeminmaxKreise
  (Minmax:Boolean):TKantenliste;
  procedure ErzeugeKreisevonfesterLaenge(Laenge:Integer);
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;
  function MausklickaufKante(X,Y:Integer):Boolean;
  procedure ZeichneKante(Flaeche:TCanvas);
  procedure ZeichneDruckKante(Flaeche:TCanvas;Faktor:Integer);
  procedure Kantezeichnen
  (Flaeche:TCanvas;Demo:Boolean;Pausenzeit:Integer);
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 WelcheLiniendicke:Integer;
procedure SetzeLiniendicke(D: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;
procedure Demopause;
procedure FuegeKnotenein(Kno:TInhaltsknoten);
procedure FuegeKanteein
(Kno1,Kno2:TInhaltsknoten;Gerichtet:Boolean;
Ka:TInhaltskante);
procedure EinfuegenKante(Ka:TInhaltskante);
procedure LoescheKante(Kno1,Kno2:TInhaltsknoten);
procedure LoescheInhaltskante(Ka:TInhaltskante);
procedure ZeichneGraph(Flaeche:TCanvas);
procedure ZeichneDruckGraph(Flaeche:TCanvas;Faktor:Integer);
procedure Graphzeichnen(Flaeche:TCanvas;Ausgabe:TLabel;
Wert:TWert;Sliste:TStringlist;
Demo:Boolean;Pausenzeit:Integer;Kantengenauigkeit:Integer);
procedure FaerbeGraph(F:TColor;T:TPenstyle);
function FindezuKoordinatendenKnoten
(var A,B:Integer;var Kno:TInhaltsknoten):Boolean;
function FindedenKnotenzuKoordinaten(var A,B:Integer;
```

```
  var Kno:TInhaltsknoten):Boolean;
  function Graphknoten(Kno:TInhaltsknoten):TInhaltsknoten;
  procedure SpeichereGraph(Dateiname:string);virtual;
  procedure LadeGraph(Dateiname:string);virtual;
  procedure EingabeKante(var Ka:Tinhaltskante;var Aus:Boolean;
  var Abbruch:Boolean);virtual;
  function Kantezeichnen(X,Y:Integer):Boolean;
  function Inhaltskanteloeschen(X,Y:Integer):Boolean;
  function Knoteninhaltzeigen(X,Y:Integer):Boolean;
  function Knotenverschieben(X,Y:Integer):Boolean;
  function Kanteverschieben(X,Y:Integer):Boolean;
  procedure EditiereKnoten(var Kno:TInhaltsknoten;var
  Abbruch:Boolean);virtual;
  function Knoteneditieren(X,Y:Integer):Boolean;
  procedure EditiereKante(var Ka:TInhaltskante;var
  Aus:Boolean;var Abbruch:Boolean);virtual;
  function Kanteeditieren(X,Y:Integer):Boolean;
  function Kanteninhaltzeigen(X,Y:Integer):Boolean;
  procedure EingabeKnoten(var Kno:TInhaltsknoten;var
  Abbruch:Boolean);virtual;
  function Knotenzeichnen(X,Y:Integer):Boolean;
  function ZweiKnotenauswaehlen(X,Y:Integer;var
  Kno1,Kno2:TInhaltsknoten;
  var Gefunden:Boolean):Boolean;
  function Inhaltsknotenloeschen(X,Y:Integer):Boolean;
  function  InhaltsKopiedesGraphen
  (Inhaltsgraphclass:TInhaltsgraphclass;
    Inhaltsknotenclass:TInhaltsknotenclass;Inhaltskanteclass:TInhaltskanteclass;
  UngerichteterGraph:Boolean):TInhaltsgraph;
  function AnzahlTypKanten(Typ:char):Integer;
  function AnzahlTypKnoten(Typ:char):Integer;
  function AnzahlBruecken(var
  SListe:TStringList;Ausgabe:TlabeL;
  Flaeche:TCanvas):Integer;
  function AlleKnotenbestimmen:TStringList;
  function AnzahlKnotenkleinsterKreis(var
  St:string;Flaeche:TCanvas):Integer;
  function AnzahlKnotengroesterKreis(var
  St:string;Flaeche:TCanvas):Integer;
  function KreisefesterLaenge(Laenge:Integer;var
  Sliste:TStringlist;
  Flaeche:TCanvas;Ausgabe:TLabel):Integer;
  function AlleKantenbestimmen:TStringList;
end;


function Bewertung(Ob:TObject):Extended;
function ErzeugeKnotenstring(Ob:TObject):string;
function ErzeugeKantenstring(Ob:TObject):string;
procedure LoescheBild(var G:TInhaltsgraph;var
```

```
Oberflaeche:TForm);



implementation


constructor TInhaltsknoten.Create;
begin
  inherited Create;
  Inhalt_:=’ ‘;
  X_:=0;
  Y_:=0;
  Radius_:=15;
  Farbe_:=clblack;
  Stil_:=pssolid;
  Typ_:=’ ‘;
  Wertlisteschreiben;
end;

procedure TInhaltsknoten.Free;
begin
  inherited Free;
end;

procedure TInhaltsknoten.Freeall;
begin
  inherited Freeall;
end;


function TInhaltsknoten.Wertlisteschreiben:TStringList;
begin
  Wertliste.Clear;
  Wertliste.Add(Inhalt_);
  Wertlisteschreiben:=Wertliste;
end;

procedure TInhaltsknoten.Wertlistelesen;
begin
  Inhalt_:=Wertliste.Strings[0];
end;
procedure TInhaltsknoten.Schreibex(X:Integer);
begin
  X_:=X;
end;

function TInhaltsknoten.Lesex:Integer;
begin
  Lesex:=X_;
```

```
end;


procedure TInhaltsknoten.Schreibey(Y:Integer);
begin
  Y_:=Y;
end;

function TInhaltsknoten.Lesey:Integer;
begin
  Lesey:=Y_;
end;

procedure TInhaltsknoten.Setzeradius(R:Integer);
begin
  Radius_:=R;
end;

function TInhaltsknoten.Welcherradius:Integer;
begin
  Welcherradius:=Radius_;
end;

procedure TInhaltsknoten.SetzeFarbe(F:TColor);
begin
  Farbe_:=F;
end;

function TInhaltsknoten.WelcheFarbe:TColor;
begin
  WelcheFarbe:=Farbe_;
end;

procedure TInhaltsknoten.SetzeStil(T:TPenstyle);
begin
  Stil_:=T;
end;

function   TInhaltsknoten.WelcherStil:TPenstyle;
begin
  WelcherStil:=Stil_;
end;
function TInhaltsknoten.WelcherTyp:Char;
begin
  WelcherTyp:=Typ_;
end;

procedure TInhaltsknoten.SetzeTyp(Typ:char);
begin
  Typ_:=Typ;
```

```
end;


procedure TInhaltsknoten.ZeichneKnoten(Flaeche:TCanvas);
begin
  Flaeche.Pen.Color:=Farbe;
  Flaeche.Pen.Style:=Stil;
  Flaeche.Ellipse(X-Radius,Y-Radius,X+Radius,Y+Radius);
  if Wert<>''
  then
    if Graph<>nil
    then
      Flaeche.Textout(X-1-7*length(Wert) DIV 2,Y-11,
     RundeStringtoString(Wert,TInhaltsgraph(Graph).Knotengenauigkeit))
    else
      Flaeche.Textout(X-1-7*length(Wert) DIV 2,Y-11,Wert);
end;

procedure TInhaltsknoten.ZeichneDruckKnoten
(Flaeche:TCanvas;Faktor:Integer);
begin
  Flaeche.Pen.Color:=Farbe;
  Flaeche.Pen.Style:=Stil;
  Flaeche.Ellipse(X-Radius,Y-Radius,X+Radius,Y+Radius);
  if Wert<>''
  then
    if Graph<>nil
    then
      Flaeche.Textout(X-55-7*length(Wert) DIV 2,Y-11*Faktor,
     RundeStringtoString(Wert,TInhaltsgraph(Graph).Knotengenauigkeit))
    else
      Flaeche.Textout(X-10-7*length(Wert) DIV 2,Y-
11*Faktor,Wert);
end;

procedure TInhaltsknoten.Knotenzeichnen
(Flaeche:TCanvas;Demo:Boolean;Pausenzeit:Integer);
begin
    Farbe:=clred;
    Stil:=psdot;
    ZeichneKnoten(Flaeche);
    if Demo then
    if Pausenzeit>0 then
    begin
      MessageBeep(0);
      Pause(Pausenzeit);
    end;
    Farbe:=clblack;
    Stil:=pssolid;
    ZeichneKnoten(Flaeche);
```

```
  end;

procedure TInhaltsknoten.AnzeigePfadliste
(Flaeche:TCanvas;Ausgabe:TLabel;
var SListe:TStringList;Zeichnen:Boolean;LetzterPfad:Boolean);
label Endschleif;
var  Zaehl:Integer;
     T:TInhaltsgraph;
     S:string;
begin
  if Pfadliste.Leer
  then
    ShowMessage('Keine Pfade')
  else
    for Zaehl:=0 to Pfadliste.Anzahl-1 do
    begin
      Application.Processmessages;
      TObject(T):=Pfadliste.Pfad(Zaehl);
      if not (T is TInhaltsgraph) then goto Endschleif;
      if Zeichnen then T.FaerbeGraph(clred,psdot);
      if Zeichnen then T.ZeichneGraph(Flaeche);
      if not T.Leer then
      begin
        S:=T.InhaltallerKnoten(ErzeugeKnotenstring)+' Summe: '+
        RundeZahltoString(T.Kantensumme(Bewertung),
        TInhaltsgraph(self.Graph).Kantengenauigkeit)+
        ' Produkt: '+
        RundeZahltoString(T.Kantenprodukt(Bewertung),
        TInhaltsgraph(self.Graph).Kantengenauigkeit);
        SListe.Add(S);
        if Zeichnen and (Ausgabe<> nil) then Ausgabe.Caption:=S;
        if Zeichnen and (Ausgabe<>nil) then Ausgabe.Refresh;
        if TInhaltsgraph(self.Graph).abbruch then exit;
        if Zeichnen and TInhaltsgraph(self.Graph).Demo then
         Messagebeep(0);
        if Zeichnen then TInhaltsgraph(self.Graph).Demopause;
        if Zeichnen and (Ausgabe<>nil) then Ausgabe.Caption:='';
        if (TInhaltsgraph(self.Graph).Demo and Zeichnen)
        or (Zaehl<Pfadliste.Anzahl-1) then
        T.FaerbeGraph(clblack,pssolid);
         if Zeichnen and (not LetzterPfad) then
        T.FaerbeGraph(clblack,pssolid);
        if Zeichnen then T.ZeichneGraph(Flaeche);
      end;
      Endschleif:
    end;
end;


function TInhaltsKnoten.ErzeugeminmaxKreise
```

```
(Minmax:Boolean):TKantenliste;
label Endproc;
var Index:Integer;
    MomentaneKantenliste:TKantenliste;
    Startknoten:TKnoten;
    MinMaxliste,Hilf:TKantenliste;
    Gefunden,Ende:Boolean;
    Kantenminzahl:Integer;

  procedure GehezudenNachbarknoten(X,Y:Tobject);
  label Endproc;
  var Z:TObject;
      Ka:TKante;
      Kno:TKnoten;
      Index,Zaehl:Integer;
  begin
    Application.Processmessages;if Ende goto Endproc;
    if TInhaltsgraph(Graph).Abbruch then goto Endproc;
    Ka:=TKante(Y);
    Kno:=TKnoten(X);
    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 Minmax and
      (MomentaneKantenliste.Anzahl>Kantenminzahl)and
          (Kantenminzahl>2) then goto Endproc;
        if not Ka.Zielknoten(Kno).AusgehendeKantenliste.Leer
           then
          for Index:=0 to
        Ka.Zielknoten(Kno).AusgehendeKantenliste.Anzahl-1 do
          begin
            if (Startknoten<>Ka.Zielknoten(Kno)) and
            (not Ka.Zielknoten(Kno).AusgehendeKantenliste.
              Kante(Index).Besucht) then
               GehezudenNachbarknoten(Ka.Zielknoten(Kno),
               Ka.Zielknoten(Kno).AusgehendeKantenliste.
                  Kante(Index));
          end;
        MomentaneKantenliste.AmEndeloeschen(Z);
        Ka.Zielknoten(Kno).Besucht:=false;
      end
      else
      if (Ka.Zielknoten(Kno)=Startknoten) and
      (Ka<>MomentaneKantenliste.Kante(MomentaneKantenliste.Letztes))
      then
      begin
```

```
        Ka.Pfadrichtung:=Ka.Zielknoten(Kno);
        MomentaneKantenliste.AmEndeanfuegen(Ka);
        if Minmax
        then
        begin
          if ((not Gefunden) or
        (MinMaxliste.Anzahl>MomentaneKantenliste.Anzahl))
            and(MomentaneKantenliste.Anzahl>2)
          then
          begin
            Kantenminzahl:=MinMaxliste.Anzahl;
            Hilf:=Minmaxliste;
            Minmaxliste:=MomentaneKantenliste.Kopie;
            Hilf.Free;
            Hilf:=nil;
            Gefunden:=true;
          end;if MinMaxliste.Anzahl=3 then Ende:=true;
      end
      else
      begin
        if (Minmaxliste.Anzahl<MomentaneKantenliste.Anzahl)
          and(MomentaneKantenliste.Anzahl>2)
        then
        begin
          Hilf:=Minmaxliste;
          Minmaxliste:=MomentaneKantenliste.Kopie;
          Hilf.Free;
          Hilf:=nil;
       end;if MinmMaxliste.Anzahl=Graph.AnzahlKanten then Ende:=true;
     end;
    MomentaneKantenliste.AmEndeloeschen(Y);
  end;
  Endproc:
end;


begin
  Kantenminzahl:=Graph.AnzahlKanten+1;
  Gefunden:=false;Ende:=false;
  MinMaxliste:=TKantenliste.Create;
  Startknoten:=self;
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.Pfadlistenloeschen;
  Graph.LoescheKnotenbesucht;
  Besucht:=true;
  if not AusgehendeKantenliste.Leer then
    for Index:=0 to self.AusgehendeKantenliste.Anzahl-1 do
    begin
      if TInhaltsgraph(self.Graph).Abbruch then goto Endproc;
      AusgehendeKantenliste.Kante(Index).Besucht:=true;
```

```
    GehezudenNachbarknoten(self,AusgehendeKantenliste.Kante(Index));
    end;
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
  Endproc:
  ErzeugeMinMaxKreise:=MinMaxliste;
end;

procedure TInhaltsKnoten.ErzeugeKreisevonfesterLaenge
(Laenge:Integer);
label Endproc;
var Index:Integer;
    MomentaneKantenliste:TKantenliste;
    Startknoten:TKnoten;
    Kreisliste,Hilf:TKantenliste;
    Gefunden:Boolean;

  procedure GehezudenNachbarknoten(X,Y:Tobject);
   Label Endproc;
  var Z:TObject;
      Ka:TKante;
      Kno:TKnoten;
      Index,Zaehl:Integer;
  begin
    Application.Processmessages;
    if TInhaltsgraph(Graph).Abbruch then goto Endproc;
    Ka:=TKante(Y);
    Kno:=TKnoten(X);
    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
            begin
            if (Startknoten<>Ka.Zielknoten(Kno)) and
            (not Ka.Zielknoten(Kno).
           AusgehendeKantenliste.Kante(Index).Besucht)
            and  and (MomentaneKantenliste.Anzahl<=Laenge-1)
         then
        GehezudenNachbarknoten(Ka.Zielknoten(Kno),Ka.Zielknoten(Kno).
           AusgehendeKantenliste.Kante(Index));
          end;
          MomentaneKantenliste.AmEndeloeschen(Z);
        Ka.Zielknoten(Kno).Besucht:=false;
```

```
      end
      else
      if (MomentaneKantenliste.Anzahl=Laenge-
      1)and(Ka.Zielknoten(Kno)=Startknoten) and
     Ka<>MomentaneKantenliste.Kante(MomentaneKantenliste.Letztes))
      then
      begin
        Ka.Pfadrichtung:=Ka.Zielknoten(Kno);
        MomentaneKantenliste.AmEndeanfuegen(Ka);
        Hilf:=Kreisliste;
        Kreisliste:=MomentaneKantenliste.Kopie;
      Startknoten.Pfadliste.AmEndeanfuegen(Kreisliste.Kopie.Graph);
        Hilf.Free;
        Hilf:=nil;
        MomentaneKantenliste.AmEndeloeschen(Z);
      end;
  Endproc:
  end;


begin
  Gefunden:=false;
  Kreisliste:=TKantenliste.Create;
  Startknoten:=self;
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.Pfadlistenloeschen;
  Graph.LoescheKnotenbesucht;
  Besucht:=true;
  if not AusgehendeKantenliste.Leer then
    for Index:=0 to self.AusgehendeKantenliste.Anzahl-1 do
    begin
      if TInhaltsgraph(self.Graph).Abbruch then goto Endproc;
      AusgehendeKantenliste.Kante(Index).Besucht:=true;
    GehezudenNachbarknoten(self,AusgehendeKantenliste.Kante(Index));
    end;
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
  Endproc:
end;

constructor TInhaltskante.Create;
begin
  inherited Create;
  Inhalt_:=' ';
  Weite_:=0;
  Farbe_:=clblack;
  Stil_:=pssolid;
  Typ_:=' ';
  Wertlisteschreiben;
end;
```

```
procedure TInhaltskante.Free;
begin
  inherited Free;
end;

procedure TInhaltskante.Freeall;
begin
  inherited Freeall;
end;

function Tinhaltskante.Wertlisteschreiben:TStringList;
begin
  Wertliste.Clear;
  Wertliste.Add(Inhalt_);
  Wertlisteschreiben:=Wertliste;
end;

procedure TInhaltskante.Wertlistelesen;
begin
  Inhalt_:=Wertliste.Strings[0];
end;

function TInhaltskante.Welchertyp:char;
begin
  Welchertyp:=Typ_;
end;

procedure TInhaltskante.SetzeTyp(Typ:char);
begin
  Typ_:=Typ;
end;

function TInhaltskante.Welcheweite:Integer;
begin
  WelcheWeite:=Weite_;
end;

procedure TInhaltskante.SetzeWeite(Weite:Integer);
begin
  Weite_:=Weite;
end;
function TInhaltskante.WelcheFarbe:TColor;
begin
  WelcheFarbe:=Farbe_;
end;

procedure TInhaltskante.SetzeFarbe(F:TColor);
begin
  Farbe_:=F;
```

```
end;

function    TInhaltskante.WelcherStil:TPenstyle;
begin
  WelcherStil:=Stil_;
end;



procedure TInhaltskante.SetzeStil(T:TPenstyle);
begin
  Stil_:=T;
end;


function TInhaltskante.MausklickaufKante(X,Y:Integer):Boolean;
var
Rx,Ry,Nx,Ny,N0x,N0y,Rjx,Rjy,Qjx,Qjy,Sjx,Sjy,Ljx,Ljy,X1,X2,Y1,Y2,
    J,Mx,My,Mkx,Mky,R:Integer;
    B,Sig,Nrx,Nry,L1,L2,U1,U2:Real;
    Gefunden:Boolean;


  procedure BestimmeParameter(X1,X2,Y1,Y2,N0x,N0y:Integer;var
  L,U:real);
  var Nenner:Real;
  begin
    Nenner:= X1*N0y-X2*N0y-N0x*(Y1-Y2);
    if abs(Nenner)>0.00000001 then
    begin
      L:=-(N0y*X-N0x*Y-X1*N0y+Y1*N0x)/Nenner;
      U:=-(X*(Y1-Y2)+Y*(X2-X1)+X1*Y2-X2*Y1)/Nenner;
    end
    else
    begin
      ShowMessage('Fehler KantenMausklickpunktbestimmung');
      Gefunden:=false;
      L:=-1;
      U:=-1;
    end;
  end;

begin
  Gefunden:=false;
  X1:=Tinhaltsknoten(Anfangsknoten).X;
  Y1:=TInhaltsknoten(Anfangsknoten).Y;
  X2:=TInhaltsknoten(Endknoten).X;
  Y2:=TInhaltsknoten(Endknoten).Y;
  J:= Weite;
  if not KanteistSchlinge
```

```
then
begin
  Ry:=Y2-Y1;
  Rx:=X2-X1;
  Nx:=Ry;
  Ny:=-Rx;
  if Ny>0
  then
  begin
    Ny:=-Ny;
    Nx:=-Nx
  end;
  Nrx:=Nx;
  Nry:=Ny;
  B:=sqrt(Nrx*Nrx+Nry*Nry);
  if B<>0
  then
  begin
    N0x:=Round(Nx/B);
    N0y:=Round(Ny/B);
  end
  else
  begin
    ShowMessage(‘Fehler B=o’);
    halt;
  end;
  if J<>0
  then
  begin
    Rjx:=X2+J*N0x;
    Rjy:=Y2+J*N0y;
    Qjx:=X1+J*N0x;
    Qjy:=Y1+J*N0y;
    Ljx:=X1-J*N0x;
    Ljy:=Y1-J*N0x;
    Sjx:=X2-J*N0x;
    Sjy:=Y2-J*N0y;
    Mx:=Round((Rjx+Qjx)/2);
    My:=Round((Rjy+Qjy)/2);
  end
  else
  begin
    Mx:=Round((X2+X1)/2);
    My:=Round((Y2+Y1)/2);
  end;
  BestimmeParameter(X1,Mx,Y1,My,N0x,N0y,L1,U1);
  BestimmeParameter(Mx,X2,My,Y2,N0x,N0y,L2,U2);
  if ((0<=L1)and(L1<=1)and(-2<=U1)and(U1<=2) )
  or((0<=L2)and(L2<1)and(-2<=U2)and(U2<=2))
  then Gefunden:=true;
```

```
  end
  else
  begin
    if J=0 then J:=30;
    Mkx:=X1-5;
    Mky:=Y1+J;
    R:=Round(sqrt(Abs(4*(X-Mkx)*(X-Mkx)+(Y-Mky)*(Y-Mky))));
    if (J>=0) and(abs(J-3)<=R)and (R<=abs(J+3)) then
Gefunden:=true;
    if (J<0) and(R<=abs(J-3))and (abs(J+3)<=R) then
Gefunden:=true
  end;
  MausklickaufKante:=Gefunden;
end;


procedure TInhaltskante.ZeichneKante(Flaeche:TCanvas);
var Radius,J,Xtext,Ytext,Xtext1,Ytext1,Xtext2,Ytext2,
    Hierx,Hiery,Dortx,Dorty:Integer;
    GleicherKnoten:Boolean;


  procedure Pfeilzeichnen(Xk1,Yk1,Xk2,Yk2,X0,Y0:Integer);
  const S=10;
  var B1,B2,Xr,Yr,Xn1,Xn2,Yn1,Yn2,Xw1,Xw2,Yw1,Yw2:Real;
      X3,Y3,X4,Y4:Integer;
  begin
    Xr:=Xk2-Xk1;
    Yr:=Yk2-Yk1;
    Xn1:=Yr;
    Yn1:=-Xr;
    Xn2:=-Yr;
    Yn2:=Xr;
    Xw1:=-Xr+Xn1;
    yw1:=-Yr+Yn1;
    Xw2:=-Xr+Xn2;
    yw2:=-Yr+Yn2;
    B1:=sqrt(Xw1*Xw1+Yw1*Yw1);
    B2:=sqrt(Xw2*Xw2+Yw2*Yw2);
    X3:=x0+Round(Xw1*S/B1);
    Y3:=y0+Round(yw1*S/B1);
    X4:=x0+Round(Xw2*S/B2);
    y4:=y0+Round(Yw2*S/B2);
    Flaeche.Moveto(X0,Y0);
    Flaeche.Lineto(X3,Y3);
    Flaeche.Moveto(X0,Y0);
    Flaeche.Lineto(X4,Y4);
  end;

  procedure GebogeneKantezeichnen(X1,Y1,X2,Y2,J:Integer);
```

```
var
Rx,Ry,Nx,Ny,N0x,N0y,Rjx,Rjy,Qjx,Qjy,Mjx,Mjy,Pfx,Pfy,Ljx,Ljy,Sjx,Sjy,Mjlx,Mjly,
    Pflx,Pfly:Integer;
    B,Sig,Nrx,Nry:real;
    St:string;
    Ungerichtet:Boolean;

begin
  Ungerichtet:=false;
  if not self.gerichtet then Ungerichtet:=true;
  St:=self.Wert;
  Ry:=Y2-Y1;
  Rx:=X2-X1;
  Nx:=Ry;
  Ny:=-Rx;
  if Ny>0 then
  begin
    Ny:=-Ny;
    Nx:=-Nx
  end;
  Nrx:=Nx;
  Nry:=Ny;
  B:=sqrt(Nrx*Nrx+Nry*Nry);
  if B<>0
  then
  begin
    N0x:=Round(Nx/B);
    N0y:=Round(Ny/B);
  end
  else
  begin
    ShowMessage('Fehler B=o');
    halt;
  end;
  Rjx:=X2+J*N0x;
  Rjy:=Y2+J*N0y;
  Qjx:=X1+J*N0x;
  Qjy:=Y1+J*N0y;
  Ljx:=X1-J*N0x;
  Ljy:=Y1-J*N0x;
  Sjx:=X2-J*N0x;
  Sjy:=Y2-J*N0y;
  Mjx:=Round((Rjx+Qjx)/2);
  Mjy:=Round((Rjy+Qjy)/2);
  if J <> 0
  then
  begin
   Flaeche.Polyline([Point(X1,Y1),Point(Mjx,Mjy),Point(X2,Y2)]);
    if St<>' '
    then
```

```
      begin
        if StringtoReal(St)<1E30
        then
        begin
          if self.Anfangsknoten.Graph<>nil
          then
            Flaeche.Textout(Mjx-5-6*length(St)  DIV 2,Mjy-5,
         RundeStringtoString(St,TInhaltsgraph(self.Anfangsknoten.Graph).
              Kantengenauigkeit))
          else Flaeche.Textout(Mjx-56*length(St)  DIV 2,Mjy-
         5,St)
        end
        else Flaeche.Textout(Mjx-56*length(St)  DIV 2,Mjy-5,'i')
      end;
      Pfx:=Round((Mjx+X2)/2);
      Pfy:=Round((Mjy+Y2)/2);
      Pflx:=Round((Mjx+X1)/2);
      Pfly:=Round((Mjy+Y1)/2);
      if not Ungerichtet then
     Pfeilzeichnen(Mjx,Mjy,X2,Y2,Pfx,Pfy);
    end
    else
    begin
      if (abs(X1-X2)>20) and (abs(Y1-Y2)>20)  then
        Flaeche.Arc(2*X1-X2,Y1,X2,2*Y2-Y1,X2,Y2,X1,Y1);
      if abs(X1-X2)<=20 then
        Flaeche.Arc(X2-20,Y2,X1+20,Y1,X2,Y2,X1,Y1);
      if (Abs(X1-X2)>20) and (Abs(Y1-Y2)<=20) then
        Flaeche.Arc(X2,Y2-20,X1,Y1+20,X2,Y2,X1,Y1);
    end;
  end;

begin
  Flaeche.Pen.Color:=self.Farbe;
  Flaeche.Pen.Style:=self.Stil;
  J:=0;
  J:=Weite;
  Radius:=J;
  if Radius=0 then Radius:=30;
  GleicherKnoten:=false;
  Hierx:=TInhaltsknoten(self.Anfangsknoten).X;
  Hiery:=TInhaltsknoten(self.Anfangsknoten).Y;
  Dortx:=TInhaltsknoten(self.Endknoten).X;
  Dorty:=TInhaltsknoten(self.Endknoten).Y;
  Xtext:=Round((Dortx*2+15+Hierx)/3);
  Ytext:=Round((Dorty*2+15+Hiery)/3);
  Xtext2:=Round((Dortx+15+Hierx)/2);
  Ytext2:=Round((Dorty+15+Hiery)/2);
  GleicherKnoten:=((Hierx=Dortx) and (Hiery=Dorty));
  Flaeche.Moveto(Hierx,Hiery);
```

```
if GleicherKnoten then
begin
  Flaeche.arc(Hierx-5-Round(Radius/2),Hiery,Hierx-
  5+round(Radius/2),
  Hiery+2*Radius,Hierx-5,Hiery,Hierx-5,Hiery);
  if gerichtet then Pfeilzeichnen(Hierx-5-Round(Radius/
  2),Hiery+2*Radius,
    Hierx-5-Round(Radius/2),Hiery,Hierx-5-Round(Radius/
  2),Hiery+Radius);
  if Wert<>' '
  then
  begin
    if StringtoReal(self.Wert)<1E30
    then
    begin
        if self.Anfangsknoten.Graph<>nil
        then
          Flaeche.Textout(Hierx-12-6*length(Wert)  DIV
     2,Hiery-7+2*Radius,
      RundeStringtoString(self.Wert,TInhaltsgraph(self.Anfangsknoten.Graph).
              Kantengenauigkeit))
          else
            Flaeche.Textout(Hierx-12-6*length(Wert)  DIV
    2,Hiery-7+2*Radius,Wert)
    end
    else
      Flaeche.Textout(Hierx-12,Hiery-7+2*Radius,'i');
  end;
end
else
begin
  if   J=0
  then
  begin
    Flaeche.Lineto(Dortx,Dorty);
    if self.Gerichtet
    then
    begin
      if self.Wert<>' '
      then
      begin
        if StringtoReal(self.Wert)<1E30
        then
        begin
          if self.Anfangsknoten.Graph<>nil
          then
            Flaeche.Textout(Xtext-6*length(Wert)  DIV 2,Ytext,
       RundeStringtoString(self.Wert,TInhaltsgraph(self.Anfangsknoten.Graph).
              Kantengenauigkeit))
          else
```

```
            Flaeche.Textout(Xtext-6*length(Wert)  DIV
         2,Ytext,self.Wert)
          end
          else
            Flaeche.Textout(Xtext,Ytext,'i');
        end;
        Xtext1:=Round(((Dortx)*4+Hierx)/5);
        Ytext1:=Round(((Dorty)*4+Hiery)/5);
        Pfeilzeichnen(Hierx,Hiery,Dortx,Dorty,Xtext1,Ytext1);
      end
      else
      begin
        if self.Wert<>' '
        then
        begin
          if StringtoReal(self.Wert)<1E30
          then
          begin
            if self.Anfangsknoten.Graph<>nil
            then
              Flaeche.Textout(Xtext2-6*length(Wert)  DIV
           2,Ytext2,
         RundeStringtoString(self.Wert,TInhaltsgraph(self.Anfangsknoten.Graph).
                  Kantengenauigkeit))
            else
              Flaeche.Textout(Xtext2-6*length(Wert)  DIV
           2,Ytext2,self.Wert)
          end
          else
            Flaeche.Textout(Xtext2,Ytext2,'i');
        end;
        if self.Wert<>' ' then
        begin
          if StringtoReal(self.Wert)<1E30
          then
          begin
            if self.Anfangsknoten.Graph<>nil
            then
              Flaeche.Textout(Xtext2-6*length(Wert)  DIV
          2,Ytext2,
         RundeStringtoString(self.Wert,TInhaltsgraph(self.Anfangsknoten.Graph).
                  Kantengenauigkeit))
            else
              Flaeche.Textout(Xtext2-6*length(Wert)  DIV
2,Ytext2,self.Wert)
          end
          else
            Flaeche.Textout(Xtext2-6*length(Wert)  DIV
2,Ytext2,'i');
        end;
```

```
      end;
    end
    else
      GebogeneKantezeichnen(Hierx,Hiery,Dortx,Dorty,J);
  end;
end;


procedure TInhaltskante.ZeichneDruckKante
(Flaeche:TCanvas;Faktor:Integer);
var Radius,J,Xtext,Ytext,Xtext1,Ytext1,Xtext2,Ytext2,
    Hierx,Hiery,Dortx,Dorty:Integer;
    GleicherKnoten:Boolean;


  procedure Pfeilzeichnen(Xk1,Yk1,Xk2,Yk2,X0,Y0:Integer);
  var B1,B2,Xr,Yr,Xn1,Xn2,Yn1,Yn2,Xw1,Xw2,Yw1,Yw2:Real;
      X3,Y3,X4,Y4,S:Integer;
  begin
    S:=Faktor*11;
    Xr:=Xk2-Xk1;
    Yr:=Yk2-Yk1;
    Xn1:=Yr;
    Yn1:=-Xr;
    Xn2:=-Yr;
    Yn2:=Xr;
    Xw1:=-Xr+Xn1;
    yw1:=-Yr+Yn1;
    Xw2:=-Xr+Xn2;
    yw2:=-Yr+Yn2;
    B1:=sqrt(Xw1*Xw1+Yw1*Yw1);
    B2:=sqrt(Xw2*Xw2+Yw2*Yw2);
    X3:=x0+Round(Xw1*S/B1);
    Y3:=y0+Round(yw1*S/B1);
    X4:=x0+Round(Xw2*S/B2);
    y4:=y0+Round(Yw2*S/B2);
    Flaeche.Moveto(X0,Y0);
    Flaeche.Lineto(X3,Y3);
    Flaeche.Moveto(X0,Y0);
    Flaeche.Lineto(X4,Y4);
  end;

  procedure GebogeneKantezeichnen(X1,Y1,X2,Y2,J:Integer);
  var
   Rx,Ry,Nx,Ny,N0x,N0y,Rjx,Rjy,Qjx,Qjy,Mjx,Mjy,Pfx,Pfy,Ljx,Ljy,Sjx,Sjy,Mjlx,Mjly,
      Pflx,Pfly:Integer;
      B,Sig,Nrx,Nry:real;
      St:string;
      Ungerichtet:Boolean;
```

```
begin
  Ungerichtet:=false;
  if not self.gerichtet then Ungerichtet:=true;
  St:=self.Wert;
  Ry:=Y2-Y1;
  Rx:=X2-X1;
  Nx:=Ry;
  Ny:=-Rx;
  if Ny>0 then
  begin
    Ny:=-Ny;
    Nx:=-Nx
  end;
  Nrx:=Nx;
  Nry:=Ny;
  B:=sqrt(Nrx*Nrx+Nry*Nry);
  if B<>0
  then
  begin
    N0x:=Round(Nx/B);
    N0y:=Round(Ny/B);
  end
  else
  begin
    ShowMessage('Fehler B=o');
    halt;
  end;
  Rjx:=X2+J*N0x;
  Rjy:=Y2+J*N0y;
  Qjx:=X1+J*N0x;
  Qjy:=Y1+J*N0y;
  Ljx:=X1-J*N0x;
  Ljy:=Y1-J*N0x;
  Sjx:=X2-J*N0x;
  Sjy:=Y2-J*N0y;
  Mjx:=Round((Rjx+Qjx)/2);
  Mjy:=Round((Rjy+Qjy)/2);
  if J <> 0
  then
  begin
   Flaeche.Polyline([Point(X1,Y1),Point(Mjx,Mjy),Point(X2,Y2)]);
    if St<>' '
    then
    begin
      if StringtoReal(St)<1E30
      then
      begin
        if self.Anfangsknoten.Graph<>nil
        then
          Flaeche.Textout(Mjx-5-6*length(St)  DIV 2,Mjy-5,
```

```
          RundeStringtoString(St,TInhaltsgraph(self.Anfangsknoten.Graph).
                 Kantengenauigkeit))
           else Flaeche.Textout(Mjx-56*length(St)  DIV 2,Mjy-
          5,St)
         end
         else Flaeche.Textout(Mjx-56*length(St)  DIV 2,Mjy-5,'i')
       end;
       Pfx:=Round((Mjx+X2)/2);
       Pfy:=Round((Mjy+Y2)/2);
       Pflx:=Round((Mjx+X1)/2);
       Pfly:=Round((Mjy+Y1)/2);
       if not Ungerichtet then
      Pfeilzeichnen(Mjx,Mjy,X2,Y2,Pfx,Pfy);
     end
     else
     begin
       if (abs(X1-X2)>20) and (abs(Y1-Y2)>20)  then
         Flaeche.Arc(2*X1-X2,Y1,X2,2*Y2-Y1,X2,Y2,X1,Y1);
       if abs(X1-X2)<=20 then
         Flaeche.Arc(X2-20,Y2,X1+20,Y1,X2,Y2,X1,Y1);
       if (Abs(X1-X2)>20) and (Abs(Y1-Y2)<=20) then
         Flaeche.Arc(X2,Y2-20,X1,Y1+20,X2,Y2,X1,Y1);
     end;
   end;

begin
   Flaeche.Pen.Color:=self.Farbe;
   Flaeche.Pen.Style:=self.Stil;
   J:=0;
   J:=Weite;
   Radius:=J;
   if Radius=0 then Radius:=30*7;
   GleicherKnoten:=false;
   Hierx:=TInhaltsknoten(self.Anfangsknoten).X;
   Hiery:=TInhaltsknoten(self.Anfangsknoten).Y;
   Dortx:=TInhaltsknoten(self.Endknoten).X;
   Dorty:=TInhaltsknoten(self.Endknoten).Y;
   Xtext:=Round((Dortx*2+15+Hierx)/3);
   Ytext:=Round((Dorty*2+15+Hiery)/3);
   Xtext2:=Round((Dortx+15+Hierx)/2);
   Ytext2:=Round((Dorty+15+Hiery)/2);
   GleicherKnoten:=((Hierx=Dortx) and (Hiery=Dorty));
   Flaeche.Moveto(Hierx,Hiery);
   if GleicherKnoten then
   begin
     if j>=0
     then
       Flaeche.arc(Hierx-Round(Radius/
       2),Hiery,Hierx+round(Radius/2),
       Hiery+2*Radius,Hierx-10,Hiery-10,Hierx+10,Hiery-10)
```

```
  else
    Flaeche.arc(Hierx-Round(Radius/
    2),Hiery,Hierx+round(Radius/2),
    Hiery+2*Radius,Hierx+10,Hiery-10,Hierx-10,Hiery-10);
  if gerichtet then Pfeilzeichnen(Hierx-5-Round(Radius/
   2),Hiery+2*Radius,
    Hierx-5-Round(Radius/2),Hiery,Hierx-5-Round(Radius/
     2),Hiery+Radius);
  if Wert<>' '
  then
  begin
    if StringtoReal(self.Wert)<1E30
    then
    begin
        if self.Anfangsknoten.Graph<>nil
        then
       Flaeche.Textout(Hierx-12-6*length(Wert)  DIV
        2,Hiery-7+2*Radius,
          RundeStringtoString(self.Wert,TInhaltsgraph(self.Anfangsknoten.Graph).
        Kantengenauigkeit))
          else
         Flaeche.Textout(Hierx-12-6*length(Wert)  DIV
          2,Hiery-7+2*Radius,Wert)
    end
    else
    Flaeche.Textout(Hierx-12,Hiery-7+2*Radius,'i');
  end;
end
else
begin
  if   J=0
  then
  begin
    Flaeche.Lineto(Dortx,Dorty);
    if self.Gerichtet
    then
    begin
      if self.Wert<>' '
      then
      begin
        if StringtoReal(self.Wert)<1E30
        then
        begin
          if self.Anfangsknoten.Graph<>nil
          then
         Flaeche.Textout(Xtext-6*length(Wert)  DIV 2,Ytext,
            RundeStringtoString(self.Wert,TInhaltsgraph(self.Anfangsknoten.Graph).
          Kantengenauigkeit))
          else
         Flaeche.Textout(Xtext-6*length(Wert)  DIV
```

```
      2,Ytext,self.Wert)
    end
    else
      Flaeche.Textout(Xtext,Ytext,'i');
  end;
  Xtext1:=Round(((Dortx)*4+Hierx)/5);
  Ytext1:=Round(((Dorty)*4+Hiery)/5);
  Pfeilzeichnen(Hierx,Hiery,Dortx,Dorty,Xtext1,Ytext1);
end
else
begin
  if self.Wert<>' '
  then
  begin
    if StringtoReal(self.Wert)<1E30
    then
    begin
      if self.Anfangsknoten.Graph<>nil
      then
     Flaeche.Textout(Xtext2-6*length(Wert)  DIV
      2,Ytext2,RundeStringtoString
     (self.Wert,TInhaltsgraph(self.Anfangsknoten.Graph).
       Kantengenauigkeit))
      else
     Flaeche.Textout(Xtext2-6*length(Wert)  DIV
      2,Ytext2,self.Wert)
    end
    else
   Flaeche.Textout(Xtext2,Ytext2,'i');
  end;
  if self.Wert<>' ' then
  begin
    if StringtoReal(self.Wert)<1E30
    then
    begin
      if self.Anfangsknoten.Graph<>nil
      then
        Flaeche.Textout(Xtext2-6*length(Wert)  DIV
         2,Ytext,RundeStringtoString
       (self.Wert,TInhaltsgraph(self.Anfangsknoten.Graph).
            Kantengenauigkeit))
      else
    Flaeche.Textout(Xtext2-6*length(Wert)  DIV
     2,Ytext2,self.Wert)
    end
    else
   Flaeche.Textout(Xtext2-6*length(Wert)  DIV
     2,Ytext2,'i');
  end;
end;
```

```
    end
    else
    GebogeneKantezeichnen(Hierx,Hiery,Dortx,Dorty,J);
  end;
end;



procedure TInhaltskante.Kantezeichnen
(Flaeche:TCanvas;Demo:Boolean;Pausenzeit:Integer);
begin
    Farbe:=clred;
    Stil:=psdot;
    ZeichneKante(Flaeche);
    if Demo then
    if Pausenzeit>0 then
    begin
      MessageBeep(0);
      Pause(Pausenzeit);
    end;
    Farbe:=clblack;
    Stil:=pssolid;
    ZeichneKante(Flaeche);
  end;


constructor TInhaltsgraph.Create;
begin
  inherited Create;
  K1:=nil;
  K2:=nil;
  K3:=nil;
  K4:=nil;
  Knotenwertposition_:=0;
  Kantenwertposition_:=0;
  Demo_:=false;
  Pausenzeit_:=0;
  Abbruch:=false;
  Zustand_:=false;
  Stop_:=false;
  Knotengenauigkeit_:=3;
  Kantengenauigkeit_:=3;
  Radius_:=15;
  Liniendicke_:=1;
  Inhaltsknotenclass_:=TInhaltsknoten;
  Inhaltskanteclass_:=TInhaltskante;
  MomentaneKnotenliste_:=TKnotenliste.Create;
  MomentaneKantenliste_:=TKantenliste.Create;
  Graphistgespeichert_:=true;
end;
```

```
procedure TInhaltsGraph.Free;
begin
  MomentaneKantenliste_.Free;
  MomentaneKantenliste_:=nil;
  MomentaneKnotenliste_.Free;
  MomentaneKnotenliste_:=nil;
  inherited Free;
end;

procedure TInhaltsGraph.Freeall;
begin
  MomentaneKantenliste_.Free;
  MomentaneKantenliste_:=nil;
  MomentaneKnotenliste_.Free;
  MomentaneKnotenliste_:=nil;
  inherited Freeall;
end;



function TInhaltsgraph.Wertlisteschreiben:TStringList;
begin
  Wertliste.Clear;
  Wertlisteschreiben:=Wertliste;
end;

procedure TInhaltsgraph.Wertlistelesen;
begin
end;

function TInhaltsgraph.WelcheKnotenclass:TInhaltsknotenclass;
begin
  WelcheKnotenclass:=Inhaltsknotenclass_;
end;

procedure
TInhaltsgraph.SetzeKnotenclass(Inhaltsclass:TInhaltsknotenclass);
begin
  Inhaltsknotenclass_:=Inhaltsclass;
end;

function TInhaltsgraph.WelcheKantenclass:TInhaltskanteclass;
begin
  WelcheKantenclass:=Inhaltskanteclass_;
end;

procedure
TInhaltsgraph.SetzeKantenclass(Inhaltsclass:TInhaltskanteclass);
begin
```

```
  Inhaltskanteclass_:=Inhaltsclass;
end;


function TInhaltsgraph.WelcheKnotenliste:TKnotenliste;
begin
   WelcheKnotenliste:=MomentaneKnotenliste_;
end;

procedure TInhaltsgraph.SetzeKnotenliste(L:TKnotenliste);
begin
   MomentaneKnotenliste_:=L;
end;

function TInhaltsgraph.WelcheKantenliste:TKantenliste;
begin
   WelcheKantenliste:=MomentaneKantenliste_;
end;

procedure TInhaltsgraph.SetzeKantenliste(L:TKantenliste);
begin
   MomentaneKantenliste_:=L;
end;

function TInhaltsgraph.WelcheKnotenwertposition:Integer;
begin
  WelcheKnotenwertposition:=Knotenwertposition_;
end;


procedure TInhaltsgraph.SetzeKnotenwertposition(P:Integer);
var Index:Integer;
begin
  Knotenwertposition_:=P;
  if not Leer then
  for Index:=0 to Knotenliste.Anzahl-1 do
    Knotenliste.Knoten(Index).Position:=self.Knotenwertposition_;
end;


function TInhaltsgraph.WelcheKantenwertposition:Integer;
begin
  WelcheKantenwertposition:=Kantenwertposition_;
end;


procedure TInhaltsgraph.SetzeKantenwertposition(P:Integer);
var Index:Integer;
begin
  Kantenwertposition_:=P;
```

```
  if not Leer then
  for Index:=0 to Kantenliste.Anzahl-1 do
    Kantenliste.Kante(Index).Position:=self.Kantenwertposition_;
end;

procedure TInhaltsgraph.SetzeWartezeit(Wz:Integer);
begin
  Pausenzeit_:=Wz;
end;

function TInhaltsgraph.WelcheWartezeit:Integer;
begin
  WelcheWartezeit:=Pausenzeit_;
end;


procedure TInhaltsgraph.SetzeDemomodus(D:Boolean);
begin
  Demo_:=D;
end;

function TInhaltsgraph.WelcherDemomodus:Boolean;
begin
  WelcherDemomodus:=Demo_;
end;


procedure TInhaltsgraph.SetzeEingabezustand(Ezsd:Boolean);
begin
  Zustand_:=Ezsd;
end;

function TInhaltsgraph.WelcherEingabezustand:Boolean;
begin
  WelcherEingabezustand:=Zustand_;
end;

procedure TInhaltsgraph.SetzeStopzustand(Stop:Boolean);
begin
  Stop_:=Stop;
end;

function TInhaltsgraph.WelcherStopzustand:Boolean;
begin
  WelcherStopzustand:=Stop_;
end;
procedure TInhaltsgraph.SetzeKnotenGenauigkeit(G:Integer);
begin
  Knotengenauigkeit_:=G;
end;
```

```
function TInhaltsgraph.WelcheKnotengenauigkeit:Integer;
begin
  WelcheKnotengenauigkeit:=Knotengenauigkeit_;
end;

procedure TInhaltsgraph.SetzeKantenGenauigkeit(G:Integer);
begin
  Kantengenauigkeit_:=G;
end;

function TInhaltsgraph.WelcheKantengenauigkeit:Integer;
begin
  WelcheKantengenauigkeit:=Kantengenauigkeit_;
end;

function TInhaltsgraph.WelcherKnotenradius:Integer;
begin
  WelcherKnotenradius:=Radius_;
end;

procedure TInhaltsgraph.SetzeKnotenradius(R:Integer);
begin
  Radius_:=R;
end;

function TInhaltsgraph.WelcheLiniendicke:Integer;
begin
  WelcheLiniendicke:=Liniendicke_;
end;

procedure TInhaltsgraph.SetzeLiniendicke(D:Integer);
begin
  Liniendicke_:=D;
end;

function TInhaltsgraph.IstGraphgespeichert:Boolean;
begin
  IstGraphgespeichert:=Graphistgespeichert_;
end;


procedure TInhaltsgraph.SetzeGraphgespeichert(Gesp:Boolean);
begin
  Graphistgespeichert_:=Gesp;
end;

function TInhaltsgraph.WelcherDateiname:string;
begin
  WelcherDateiname:=Dateiname_;
```

```
end;

procedure TInhaltsgraph.SetzeDateiname(N:string);
begin
  Dateiname_:=N;
end;

procedure
TInhaltsgraph.SetzeletztenMausklickknoten(Kno:TInhaltsknoten);
begin
  K4:=Kno;
end;

function
TInhaltsgraph.WelcherletzteMausklickknoten:TInhaltsknoten;
begin
  if not self.Leer then
  if K4=nil then K4:=TInhaltsknoten(Anfangsknoten);
  WelcherletzteMausklickknoten:=Graphknoten(K4);
end;

function TInhaltsgraph.WelcherK1:TInhaltsknoten;
begin
   WelcherK1:=K1_
end;


procedure TInhaltsgraph.SetzeK1(Kno:TInhaltsknoten);
begin
  K1_:=Kno;
end;


function TInhaltsgraph.WelcherK2:TInhaltsknoten;
begin
   WelcherK2:=K2_
end;

procedure TInhaltsgraph.SetzeK2(Kno:TInhaltsknoten);
begin
  K2_:=Kno;
end;
function TInhaltsgraph.WelcherK3:TInhaltsknoten;
begin
   WelcherK3:=K3_
end;


procedure TInhaltsgraph.SetzeK3(Kno:TInhaltsknoten);
begin
```

```
  K3_:=Kno;
end;

function TInhaltsgraph.WelcherK4:TInhaltsknoten;
begin
   WelcherK4:=K4_
end;


procedure TInhaltsgraph.SetzeK4(Kno:TInhaltsknoten);
begin
  K4_:=Kno;
end;

procedure TInhaltsgraph.Demopause;
begin
  if Demo then
  begin
    if Pausenzeit>0 then
    begin
      MessageBeep(0);
      Pause(Pausenzeit);
    end;
    if Pausenzeit<0 then
    begin
    repeat
    Application.processmessages;
  until Pausenzeit>=0;
  Pausenzeit:=-abs(Pausenzeit);
  end;
  end;
end;

procedure TInhaltsgraph.FuegeKnotenein(Kno:TInhaltsknoten);
begin
  self.Knoteneinfuegen(Kno);
end;

procedure TInhaltsgraph.FuegeKanteein
(Kno1,Kno2:TInhaltsknoten;Gerichtet:Boolean;Ka:TInhaltskante);
label Endschleife1,Endschleife2;
var Richtung:Boolean;
    Pos1,Pos2,Index:Integer;
    Kna,Kne:TKnoten;
    Kno:TInhaltsknoten;
begin
  if Gerichtet then Richtung:=true else Richtung:=false;
  Pos1:=-1;
  if not Leer then
  for Index:=0 to self.Knotenliste.Anzahl-1 do
```

```
  begin
    Kno:=TInhaltsknoten(self.Knotenliste.Knoten(Index));
    if (Kno.X=Kno1.X) and (Kno.Y=Kno1.Y) then
    begin
      Pos1:=Index;
      goto Endschleife1;
    end;
  end;
  Endschleife1:
  Pos2:=-1;
  if not Leer then
  for Index:=0 to self.Knotenliste.Anzahl-1 do
  begin
    Kno:=TInhaltsknoten(self.Knotenliste.Knoten(Index));
    if (Kno.X=Kno2.X) and (Kno.Y=Kno2.Y) then
    begin
      Pos2:=Index;
      goto Endschleife2;
    end;
  end;
  Endschleife2:
  if Pos1>=0 then Kna:=self.Knotenliste.Knoten(Pos1) else
Kna:=K1;
  if Pos2>=0 then Kne:=self.Knotenliste.Knoten(Pos2) else
Kne:=K2;
  if (Kna<>nil)and (Kne<>nil)and
    (Ka<>nil) then self.Kanteeinfuegen(Ka,Kna,Kne,Richtung);
end;

procedure TInhaltsgraph.EinfuegenKante(Ka:TInhaltskante);
begin
  if Graphknoten(TInhaltsknoten(Ka.Anfangsknoten))=nil
  then
    KnotenEinfuegen(Ka.Anfangsknoten);
  if Graphknoten(TInhaltsknoten(Ka.Endknoten))=nil
  then
    KnotenEinfuegen(Ka.Endknoten);
      FuegeKanteein(TInhaltsknoten(Ka.Anfangsknoten),TInhaltsknoten(Ka.Endknoten),
    Ka.gerichtet,Ka);;
end;


procedure TInhaltsgraph.LoescheKante(Kno1,Kno2:TInhaltsknoten);
var Index:Integer;
    Ka:TKante;
begin
  for Index:=0 to self.Kantenliste.Anzahl-1 do
  begin
    Ka:=self.Kantenliste.Kante(Index);
    if ((Ka.Anfangsknoten=Kno1) and (Ka.Endknoten=Kno2))or
```

```
      ((Ka.Anfangsknoten=Kno2) and (Ka.Endknoten=Kno1))
    then
    begin
      self.Kanteloeschen(Ka);
      exit;
    end;
  end;
end;


procedure TInhaltsgraph.LoescheInhaltsKante(Ka:TInhaltskante);
var Index:Integer;
    Kant:TKante;
begin
  for Index:=0 to self.Kantenliste.Anzahl-1 do
  begin
    Kant:=self.Kantenliste.Kante(Index);
    if Ka=Kant
    then
    begin
      self.Kanteloeschen(Ka);
      exit;
    end;
  end;
end;

procedure TInhaltsgraph.ZeichneGraph(Flaeche:TCanvas);
var Index:Integer;
begin
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    TInhaltsknoten(Knotenliste.Knoten(Index)).ZeichneKnoten(Flaeche);
    if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
     TInhaltskante(Kantenliste.Kante(Index)).ZeichneKante(Flaeche);
end;


procedure
TInhaltsgraph.ZeichneDruckGraph(Flaeche:TCanvas;Faktor:Integer);
var Index:Integer;
    Kno:TInhaltsknoten;
    Ka:TInhaltskante;
begin
  if not Leer
  then
  begin
      For Index:=0 to Knotenliste.Anzahl-1 do
      begin
        Kno:=TInhaltsknoten(Knotenliste.Knoten(Index));
```

```
        Kno.X:=Kno.X*Faktor;
        Kno.Y:=Kno.y*Faktor;
        Kno.Radius:=Kno.Radius*Faktor;
      end;
      For Index:=0 to Kantenliste.Anzahl-1 do
      begin
         Ka:=TInhaltskante(Kantenliste.Kante(Index));
         Ka.Weite:=Ka.Weite*Faktor;
      end;
    if not Knotenliste.Leer then
      for Index:=0 to Knotenliste.Anzahl-1 do
    TInhaltsknoten(Knotenliste.Knoten(Index)).ZeichneDruckKnoten(Flaeche,Faktor);
    if not Kantenliste.Leer then
      for Index:=0 to Kantenliste.Anzahl-1 do
    TInhaltskante(Kantenliste.Kante(Index)).ZeichneDruckKante(Flaeche,Faktor);
  end;
end;


procedure TInhaltsgraph.Graphzeichnen(Flaeche:TCanvas;
Ausgabe:TLabel;Wert:Twert;Sliste:TStringlist;Demo:Boolean;Pausenzeit:Integer;Kantengenauigkeit:Integer);
begin
  FaerbeGraph(clred,psdot);
  ZeichneGraph(Flaeche);
  if Demo then
    if Pausenzeit>0 then
    begin
      MessageBeep(0);
      Pause(Pausenzeit);
    end;
  FaerbeGraph(clblack,pssolid);
  ZeichneGraph(Flaeche);
  Ausgabe.Caption:=
    InhaltallerKnoten(ErzeugeKnotenstring)+' Summe: '+
    RundeZahltostring(Kantensumme(Wert),Kantengenauigkeit)+'
      Produkt: '+
    RundeZahltostring(Kantenprodukt(Wert),Kantengenauigkeit);
  Ausgabe.Refresh;
  SListe.Add(Ausgabe.Caption);
  Ausgabe.Caption:='';
  Ausgabe.Refresh;
end;
procedure TInhaltsgraph.FaerbeGraph(F:TColor;T:TPenstyle);
var Index:Integer;
begin
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      TInhaltsknoten(Knotenliste.Knoten(Index)).Farbe:=F;
      TInhaltsknoten(Knotenliste.Knoten(Index)).Stil:=T;
```

```
    end;
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
    begin
      TInhaltskante(Kantenliste.Kante(Index)).Farbe:=F;
      TInhaltskante(Kantenliste.Kante(Index)).Stil:=T;
    end;
end;


function TInhaltsgraph.FindezuKoordinatendenKnoten(var
A,B:Integer;
  Var Kno:TInhaltsknoten):Boolean;
label Ende;
var Hilf,Z,Kl:TKnoten;
    Position,Index:Integer;
    Kn:TInhaltsknoten;

  function
GleicheKoordinaten(A,B:Integer;Kn:TInhaltsknoten):Boolean;
  var  Abstand1,Abstand2:Integer;
  begin
    Abstand1:=abs(A-Kn.X);
    Abstand2:=abs(B-Kn.Y);
    GleicheKoordinaten:= (Abstand1<20*Round(Kn.Radius/15))
      and (Abstand2<20*Round(Kn.Radius/15));
  end;

begin
  Position:=-1;
  if not self.Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kn:=TInhaltsknoten(Knotenliste.Knoten(Index));
      if GleicheKoordinaten(A,B,Kn) then
      begin
        Position:=Index;
        goto Ende;
      end;
    end;
  Ende:
  if Position >=0 then Hilf:=self.Knotenliste.Knoten(Position);
  if Position <0
  then
    FindezuKoordinatendenKnoten:=false
  else
  begin
    FindezuKoordinatendenKnoten:=true;
    Kno:=TInhaltsknoten.Create;
    Kno.Wert:=TInhaltsknoten(Hilf).Wert;
```

```
    Kno.X:=TInhaltsknoten(Hilf).X;
    Kno.Y:=TInhaltsknoten(Hilf).Y;
    Kl:=Kno;
    Kno:=TInhaltsknoten(Graphknoten(Kno));
    Kl.Freeall;
    Kl:=nil;
  end;
end;

function TInhaltsgraph.FindedenKnotenzuKoordinaten(var
A,B:Integer;
  Var Kno:TInhaltsknoten):Boolean;
label Ende;
var Hilf,Z,Kl:TKnoten;
    Position,Index:Integer;
    Kn:TInhaltsknoten;

  function
GleicheKoordinaten(A,B:Integer;Kn:TInhaltsknoten):Boolean;
  var  Abstand1,Abstand2:Integer;
  begin
    Abstand1:=abs(A-Kn.X);
    Abstand2:=abs(B-Kn.Y);
    GleicheKoordinaten:= (Abstand1<20*Round(Kn.Radius/15))
      and (Abstand2<20*Round(Kn.Radius/15));
  end;

begin
  Position:=-1;
  if not self.Knotenliste.Leer then
    for Index:=Knotenliste.Anzahl-1 downto 0 do
    begin
      Kn:=TInhaltsknoten(Knotenliste.Knoten(Index));
      if GleicheKoordinaten(A,B,Kn) then
      begin
        Position:=Index;
        goto Ende;
      end;
    end;
  Ende:
  if Position >=0 then Hilf:=self.Knotenliste.Knoten(Position);
  if Position <0
  then
    FindedenKnotenzuKoordinaten:=false
  else
  begin
    FindedenKnotenzuKoordinaten:=true;
    Kno:=TInhaltsknoten.Create;
    Kno.Wert:=TInhaltsknoten(Hilf).Wert;
    Kno.X:=TInhaltsknoten(Hilf).X;
```

```
    Kno.Y:=TInhaltsknoten(Hilf).Y;
    Kl:=Kno;
    Kno:=TInhaltsknoten(Graphknoten(Kno));
    Kl.Freeall;
    Kl:=nil;
  end;
end;

function TInhaltsgraph.Graphknoten
(Kno:TInhaltsknoten):TInhaltsKnoten;
var Index:Integer;
    Kn:TInhaltsknoten;
begin
  Graphknoten:=nil;
  if not self.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kn:=TInhaltsknoten(self.Knotenliste.Knoten(Index));
      if (Kno.X=Kn.X) and (Kno.Y=Kn.Y)
      then
      begin
        Kn.Graph:=self;
        Graphknoten:=Kn;
        exit;
      end
    end;
end;


procedure TInhaltsgraph.SpeichereGraph(Dateiname:string);
var Index:Integer;
    Datei:Text;

  procedure SpeichereKanten(X,Ke:TObject);
  var Kn:TInhaltskante;
      Zkno:TKnoten;
      Index:Integer;
  begin
    Zkno:=TKnoten(X);
    Kn:=TInhaltskante(Ke);
    if not Kn.Besucht  then
    begin
      Kn.Besucht:=true;
      Kn.Wertlisteschreiben;
      writeln(Datei,'Naechste Kante');
      writeln(Datei,'Zielknoten');
      writeln(Datei,(TInhaltsknoten(Kn.Zielknoten(Zkno)).X));
      writeln(Datei,(TInhaltsknoten(Kn.Zielknoten(Zkno)).Y));
      writeln(Datei,'Kantendaten');
      writeln(Datei,Kn.Typ);
```

```
    writeln(Datei,Inttostr(Kn.Farbe));
    writeln(Datei,Inttostr(ord(Kn.Stil)));
    if  Kn.Wertliste.count>0
    then
    begin
      for Index:=0 to Kn.Wertliste.Count do
      begin
       if Index=Kn.Wertliste.Count
        then
        writeln(Datei,'Feldende')
        else
       writeln(Datei,Kn.Wertliste.Strings[Index]);
      end;
    end
    else
    writeln(Datei,'Leere Wertliste');
    writeln(Datei,Kn.Weite);
    if Kn.Gerichtet
    then
    writeln(Datei,'gerichtet')
    else
    writeln(Datei,'ungerichtet');
  end;
end;


procedure SpeicheredieKantenzueinemKnoten(X:TObject);
var Index:Integer;
begin
  writeln(Datei,'Naechster Knoten:');
  writeln(Datei,TInhaltsknoten(X).X);
  writeln(Datei,TInhaltsknoten(X).Y);
  if not TInhaltsknoten(X).AusgehendeKantenliste.Leer then
    for Index:=0 to
   TInhaltsknoten(X).AusgehendeKantenliste.Anzahl-1 do
   SpeichereKanten(X,TInhaltsknoten(X).AusgehendeKantenliste.Kante(Index));
end;


procedure SpeicherealleKnoten(X:TObject);
var Index:Integer;
begin
  writeln(Datei,'Knoten');
  writeln(Datei,TInhaltsknoten(X).X);
  writeln(Datei,TInhaltsknoten(X).Y);
  writeln(Datei,TInhaltsknoten(X).Typ);
  writeln(Datei,Inttostr(TInhaltsknoten(X).Farbe));
  writeln(Datei,Inttostr(ord(TInhaltsknoten(X).Stil)));
  if TInhaltsknoten(X).Wertliste.Count>0 then
  begin
```

```
      for Index:=0 to TInhaltsknoten(X).Wertliste.Count do
      begin
        TInhaltsknoten(X).Wertlisteschreiben;
        if Index=TInhaltsknoten(X).Wertliste.Count
        then
          writeln(Datei,'Feldende')
        else
       writeln(Datei,TInhaltsknoten(X).Wertliste.Strings[Index]);
      end;
    end
    else
      writeln(Datei,'Leere Wertliste');
    writeln(Datei,'radius');
    writeln(Datei,TInhaltsknoten(X).Radius);
  end;


begin
  try
    self.loescheKantenbesucht;
    AssignFile(Datei,Dateiname);
    Rewrite(Datei);
    if not Leer then
    begin
      writeln(Datei,'Knotenlistenanfang');
      if not Knotenliste.Leer then
        for Index:=0 to Knotenliste.Anzahl-1 do
          SpeicherealleKnoten(Knotenliste.Knoten(Index));
      writeln(Datei,'Knotenlistenende');
      writeln(Datei,'Kantenlistenanfang');
      if not Knotenliste.Leer then
        for Index:=0 to Knotenliste.Anzahl-1 do
       SpeicheredieKantenzueinemKnoten(Knotenliste.Knoten(Index));
      writeln(Datei,'Kantenlistenende');
      writeln(Datei,'GraphWertliste');
      if (self.Wertliste<>nil) and (self.Wertliste.Count>0) then
      begin
        self.Wertlisteschreiben;
        for Index:=0 to self.Wertliste.Count do
        begin
          if Index=Wertliste.Count
          then
          writeln(Datei,'Feldende')
          else
         writeln(Datei,self.Wertliste.Strings[Index]);
        end;
      end
      else
        writeln(Datei,'Leere Wertliste');
      writeln(Datei,IntegertoString(Knotengenauigkeit));
```

```
      writeln(Datei,IntegertoString(Kantengenauigkeit));
      writeln(Datei,'Liniendicke');
      writeln(Datei,IntegertoString(Liniendicke));
      writeln(Datei,'Dateiende');
      CloseFile(Datei);
    end;
  except
    CloseFile(Datei);
    ShowMessage('Fehler beim Speichern des Graphen!');
  end;
end;


procedure TInhaltsgraph.LadeGraph(Dateiname:string);
var Knoten,Knoten1,Knoten2:TInhaltsknoten;
    Zeile,Typ,Inhalt:string;
    X,Y,R,Dortx,Dorty,Weite:Integer;
    Kante:TInhaltskante;
    Datei:Text;


  function ErmittleStil(i:Integer):TPenstyle;
  begin
    case i of
      0:ErmittleStil:=pssolid;
      1:ErmittleStil:=psDash;
      2:ErmittleStil:=psDot;
      3:ErmittleStil:=psDashDot;
      4:ErmittleStil:=psDashDotDot;
      5:ErmittleStil:=psclear;
      6:ErmittleStil:=psInsideFrame;
      else ErmittleStil:=pssolid;
    end;
  end;

begin
  try
    Assignfile(Datei,Dateiname);
    Reset(Datei);
    while not eof(Datei) do
    begin
      readln(Datei,Zeile);
      readln(Datei,Zeile);
      while Zeile='Knoten' do
       begin
        Knoten:=self.Inhaltsknotenclass.Create;
        Knoten.Wertliste.Clear;
        readln(Datei,X);
        readln(Datei,Y);
        readln(Datei,Zeile);
```

```
  Knoten.Typ:=Zeile[1];
  readln(Datei,Zeile);
  Knoten.Farbe:=StrToInt(Zeile);
  readln(Datei,Zeile);
  Knoten.Stil:=ErmittleStil(StrToInt(Zeile));
  repeat
    readln(Datei,Zeile);
   if ((Zeile<>'Leere Wertliste')and (Zeile<>'Feldende'))
    then
    Knoten.Wertliste.Add(Zeile);
  until ((Zeile='Leere Wertliste') or (Zeile='Feldende'));
   Knoten.Wertlistelesen;
  readln(Datei,Zeile);
  if Zeile='radius' then
  begin
    readln(Datei,R);
    Knoten.Radius:=R;
    readln(Datei,Zeile);
   end;
  Knoten.X:=X;
  Knoten.Y:=Y;
 FuegeKnotenein(Knoten);
end;
readln(Datei,Zeile);
readln(Datei,Zeile);
while Zeile='Naechster Knoten:' do
begin
  Knoten1:=self.Inhaltsknotenclass.Create;
  readln(Datei,X);
  readln(Datei,Y);
  TInhaltsknoten(Knoten1).X:=X;
  TInhaltsknoten(Knoten1).Y:=Y;
  readln(Datei,Zeile);
  while Zeile='Naechste Kante' do
  begin
  Knoten2:=self.Inhaltsknotenclass.Create;
    readln(Datei,Zeile);
    readln(Datei,Dortx);
    readln(Datei,Dorty);
   TInhaltsknoten(Knoten2).X:=Dortx;
   TInhaltsknoten(Knoten2).Y:=Dorty;
    readln(Datei,Zeile);
    readln(Datei,Zeile);
    Kante:=self.Inhaltskanteclass.Create;
    Kante.Wertliste.Clear;
    Kante.Typ:=Zeile[1];
    readln(Datei,Zeile);
    Kante.Farbe:=StrToInt(Zeile);
    readln(Datei,Zeile);
    Kante.Stil:=ErmittleStil(StrToInt(Zeile));
```

```
          repeat
            readln(Datei,Zeile);
            if ((Zeile<>'Leere Wertliste')and
            (Zeile<>'Feldende')) then
            Kante.Wertliste.Add(Zeile);
          until ((Zeile='Leere Wertliste') or
          (Zeile='Feldende'));
          Kante.Wertlistelesen;
          readln(Datei,Weite);
          Kante.Weite:=Weite;
          readln(Datei,Zeile);
          if (Zeile='gerichtet') or (Zeile='gerichted')
          then
           Kante.gerichtet:=true
          else
          Kante.gerichtet:=false;
          readln(Datei,Zeile);
       self.FuegeKanteein(Knoten1,Knoten2,Kante.gerichtet,Kante);
        end;
      end;
      if not eof(Datei) then
      begin
        readln(Datei,Zeile);
        self.Wertliste.Clear;
        repeat
          readln(Datei,Zeile);
         if ((Zeile<>'Leere Wertliste')and (Zeile<>'Feldende'))
          then
          self.Wertliste.Add(Zeile);
        until ((Zeile='Leere Wertliste') or (Zeile='Feldende'));
        self.Wertlistelesen;
        Readln(Datei,Zeile);
       self.Knotengenauigkeit:=StringtoInteger(Zeile);
        Readln(Datei,Zeile);
       self.Kantengenauigkeit:=StringtoInteger(Zeile);
        if not Eof(Datei) then Readln(Datei,Zeile);
        if not Eof(Datei) then Readln(Datei,Zeile);
        Liniendicke:=StringtoInteger(Zeile);
      end;
    end;
    Closefile(Datei);
  except
    Closefile(Datei);
    ShowMessage('Fehler beim Laden des Graphen! Möglicherweise
falscher Graphtyp!!');
  end;
end;


procedure TInhaltsgraph.EingabeKante(var Ka:Tinhaltskante;var
```

```
Aus:Boolean;
var Abbruch:Boolean);
var Typ:char;
begin
  Kantenform.Kanteninhalt:=' ';
  Kantenform.Kantenweite:=0;
  Kantenform.Showmodal;
  if Kantenform.Kantenreal= true
  then
    Typ:='r'
  else
    if Kantenform.KantenInteger=true then Typ:='i' else
  Typ:='s';
  Ka.Typ:=Typ;
  Ka.Position:=self.Kantenwertposition;
  if Abs(StringtoReal(Kantenform.Kanteninhalt))<1.0E30
  then
     Ka.Wert:=Kantenform.Kanteninhalt
  else
  begin
   ShowMessage('Fehler! Eingabe nicht im zulässigen numerischen
   Bereich!');
    exit;
  end;
  Ka.Weite:=Kantenform.Kantenweite;
  Ka.gerichtet:=false;
  if (Kantenform.Kanteein=false) and (Kantenform.Kanteaus=true)
  then
  begin
    Ka.gerichtet:=true;
    Aus:=true;
  end;
  if (Kantenform.Kanteein=true) and (Kantenform.Kanteaus=false)
  then
  begin
    Ka.gerichtet:=true;
    Aus:=false;
  end;
  Abbruch:=Kantenform.Kantenabbruch;
end;


function TInhaltsgraph.Kantezeichnen(X,Y:Integer):Boolean;
var Ka:TInhaltskante;
    Aus,abbruch:Boolean;
    K:TInhaltsknoten;
begin
  result:=false;
  if K1<>nil  then
  begin
```

```
    if FindezuKoordinatendenKnoten(X,Y,K)=false then
    begin
      ShowMessage('Kein Knoten!');
      K2:=nil;
      K1:=nil;
      result:=true;
    end
    else
    begin
      K2:=K;
      Ka:=self.Inhaltskanteclass.Create;
      self.EingabeKante(Ka,Aus,abbruch);
      if not Abbruch then
      begin
        if Ka.Gerichtet then
        begin
          if not Aus
          then
            FuegeKanteein(K2,K1,Ka.Gerichtet,Ka)
          else
            if Aus  then
              FuegeKanteein(K1,K2,Ka.Gerichtet,Ka)
        end
        else
          FuegeKanteein(K1,K2,Ka.Gerichtet,Ka);
      end;
      result:=true;
      K1:=nil;
      K2:=nil;
    end
  end
  else
  begin
    if FindezuKoordinatendenKnoten(X,Y,K) =false then
    begin
      ShowMessage('Kein Knoten!');
      K1:=nil;
      K2:=nil;
      result:=true;
    end
    else
    begin
       K1:=K;
      result:=false;
    end;
  end;
end;

function TInhaltsgraph.InhaltskanteloeschenX,Y:Integer):Boolean;
```

```
label Endproc;
var Kante:TInhaltskante;
    Typ:char;
    Index:Integer;
    Gefunden:Boolean;
begin
  Gefunden:=false;
  if not self.Kantenliste.Leer then
  for Index:=0 to self.Kantenliste.Anzahl-1 do
  begin
    Kante:=TInhaltskante(self.Kantenliste.Kante(Index));
    if Kante.MausklickaufKante(X,Y) then
    begin
      LoescheInhaltsKante(Kante);
      Gefunden:=true;
      goto Endproc;
    end;
  end;
  Endproc:
  if not Gefunden then ShowMessage('Keine Kante!');
  InhaltsKanteloeschen:=Gefunden;
end;

function TInhaltsgraph.Knoteninhaltzeigen(X,Y:Integer):Boolean;
var K:TInhaltsknoten;
    S:string;
begin
  Knoteninhaltzeigen:=false;
  if not self.Leer then
  begin
    if FindezuKoordinatendenKnoten(X,Y,K)=true then
    begin
      S:='Inhalt: '+K.Wert+chr(13)+'X= '+InttoStr(K.X)+' '+'Y=
'+Inttostr(K.Y);Knotenwertposition:=0;
      ShowMessage(S);
      Knoteninhaltzeigen:=true;
    end;
  end;
end;

function TInhaltsgraph.Knotenverschieben(X,Y:Integer):Boolean;
var K:TInhaltsknoten;
begin
  Knotenverschieben:=false;
  K:=K3;
  if FindezuKoordinatendenKnoten(X,Y,K) then
  begin
    K3:=K;
    if K3.Besucht then
    begin
```

```
      K3.X:=X;
      K3.Y:=Y;
      Knotenverschieben:=true;
    end
    else
    begin
      K:=K3;
      if FindedenKnotenzuKoordinaten(X,Y,K) then
      begin
        K3:=K;
        if K3.Besucht then
        begin
          K3.X:=X;
          K3.Y:=Y;
          Knotenverschieben:=true;
        end;
      end;
    end;
  end;
end;


function TInhaltsgraph.Kanteverschieben(X,Y:Integer):Boolean;
label Endproc;
var Rx,Ry,Nx,Ny,N0x,N0y,Rjx,Rjy,Qjx,Qjy,Mjx,Mjy,Mjlx,Mjly,
    X1,X2,Y1,Y2,Deltaj,Deltal,Deltax,Deltay,J,Dist,Index:Integer;
    B,Sig,Nrx,Nry:real;
    Ka:TInhaltskante;
begin
  Kanteverschieben:=false;
  if not self.Kantenliste.Leer then
    for Index:=0 to self.Kantenliste.Anzahl-1 do
    begin
      Ka:=TInhaltskante(Kantenliste.Kante(Index));
      if Ka.Besucht then
      begin
        X1:=TInhaltsknoten(Ka.Anfangsknoten).X;
        Y1:=TInhaltsknoten(Ka.Anfangsknoten).Y;
        X2:=TInhaltsknoten(Ka.Endknoten).X;
        Y2:=TInhaltsknoten(Ka.Endknoten).Y;
        J:=Ka.Weite;
        if not Ka.KanteistSchlinge then
        begin
          Ry:=Y2-Y1;
          Rx:=X2-X1;
          Nx:=Ry;
          Ny:=-Rx;
          if Ny>0 then
          begin
            Ny:=-Ny;
```

```
          Nx:=-Nx
        end;
        Nrx:=Nx;
        Nry:=Ny;
      if (Nrx*Nrx+Nry*Nry)<0 then ShowMessage('Fehler! Radi
        kand<0');
      B:=sqrt(Nrx*Nrx+Nry*Nry);
        if B<>0
        then
        begin
          N0x:=Round(Nx/B);
          N0y:=Round(Ny/B);
        end
        else
       ShowMessage('Fehler! B=0');
        Rjx:=X2+J*N0x;
        Rjy:=Y2+J*N0y;
        Qjx:=X1+J*N0x;
        Qjy:=Y1+J*N0y;
       Mjx:=Round((Rjx+Qjx)/2);
       Mjy:=Round((Rjy+Qjy)/2);
      if (((X-Mjx)*(X-Mjx)+(Y-Mjy)*(Y-Mjy)) <0 )then goto
        Endproc;;
      Dist:=Round(sqrt((X-Mjx)*(X-Mjx)+(Y-Mjy)*(Y-Mjy)));
        if Dist<20 then
        begin
          Deltax:=X-Mjx;
          Deltay:=Y-Mjy;
          Deltaj:=N0x*Deltax+N0y*Deltay;
          J:=J+Deltaj;
          Ka.Weite:=J;
          Kanteverschieben:=true;
        end;
      end
      else
      begin
      if J=0 then J:=30;
      Mjx:=X1-5;
      Mjy:=Y1+2*J;
      N0x:=0;
      N0y:=1;
      if (( (X-Mjx)*(X-Mjx)+(Y-Mjy)*(Y-Mjy)) <0 )then goto
Endproc;;
      Dist:=Round(sqrt((X-Mjx)*(X-Mjx)+(Y-Mjy)*(Y-Mjy)));
      if Dist<20 then
      begin
        Deltax:=X-Mjx;
        Deltay:=Y-Mjy;
        Deltaj:=N0x*Deltax+N0y*Deltay;
        J:=J+Deltaj;
```

```
          Ka.Weite:=J;
          Kanteverschieben:=true;
        end;
      end;
    end;
  end;
  Endproc:
end;



procedure TInhaltsgraph.EditiereKnoten(var
Kno:TInhaltsknoten;var Abbruch:Boolean);
var S:string;
begin
  Abbruch:=false;
  S:=Inputbox('Inhalt '+K1.Wert+' ändern','Eingabe neuer Inhalt:
  ',Kno.Wert);
  if (not StringistRealZahl(S)) or (StringistRealZahl(S) and
  (Abs(StringtoReal(S))<1.0E30))
  then
    Kno.Wert:=S
  else
    begin
      ShowMessage('Fehler! Eingabe nicht im zulässigen numeri
       schen Bereich!');
      Abbruch:=true;
    end;
end;


function TInhaltsgraph.Knoteneditieren(X,Y:Integer):Boolean;
var S:string;
    Abbruch:Boolean;
    K:TInhaltsknoten;
begin
  Knoteneditieren:=false;
  if  FindezuKoordinatendenKnoten(X,Y,K)=true
  then
  begin
    K1:=K;
    S:=K1.Wert;
   self.EditiereKnoten(K,Abbruch);
    K1:=K;
    if abbruch then K1.Wert:=S;
    K1:=nil;
    Knoteneditieren:=true;
  end
  else
    begin
```

```
      K1:=K;
      ShowMessage('Kein Knoten!');
    end;
end;


procedure TInhaltsgraph.EditiereKante(var Ka:TInhaltskante;var
Aus:Boolean;
var Abbruch:Boolean);
var Typ:char;
begin
  Kantenform.Kanteninhalt:=Ka.Wert;
  Kantenform.Kantenweite:=Ka.Weite;
  if Ka.gerichtet then
  begin
    Kantenform.Kanteein:=false;
    Kantenform.Kanteaus:=true;
  end
  else
  begin
    Kantenform.Kanteein:=false;
    Kantenform.Kanteaus:=false;
  end;
  Kantenform.Kantenreal:=false;
  Kantenform.KantenInteger:=false;
  if Ka.Typ='r' then Kantenform.Kantenreal:=true;
  if Ka.Typ='i' then Kantenform.Kanteninteger:=true;
  KantenForm.Showmodal;
  if KantenForm.Kantenreal= true
  then
    Typ:='r'
  else
    if Kantenform.Kanteninteger=true
    then
      Typ:='i'
    else
      Typ:='s';
  if Abs(StringtoReal(Kantenform.Kanteninhalt))<1.0E30
  then
     TInhaltskante(Ka).Wert:=Kantenform.Kanteninhalt
  else
  begin
   ShowMessage('Fehler! Eingabe nicht im zulässigen numerischen
   Bereich!');
   Abbruch:=true;
  end;
  TInhaltskante(Ka).Typ:=Typ;
  TInhaltskante(Ka).Weite:=Kantenform.Kantenweite;
  if ((Kantenform.Kanteein=false)and(Kantenform.Kanteaus=false))
   or
```

```
  ((Kantenform.Kanteein=true)and(Kantenform.Kanteaus=true))
  then
    Ka.gerichtet:=false;
  if ((Kantenform.Kanteein=false)and(Kantenform.Kanteaus=true))
  then
  begin
    Ka.gerichtet:=true;
    Aus:=true;
  end;
  if ((Kantenform.Kanteein=true)and(Kantenform.Kanteaus=false))
  then
  begin
    Ka.gerichtet:=true;
    Aus:=false;
  end;
  Abbruch:=Kantenform.Kantenabbruch;
end;



function TInhaltsgraph.Kanteeditieren(X,Y:Integer):Boolean;
label Endproc;
var Zaehl:Integer;
    Ka:TInhaltskante;
    Hilf:TKnoten;
    Index:Integer;
    Gefunden,Aus,Abbruch:Boolean;
begin
  Gefunden:=false;
  if not self.Kantenliste.Leer then
    for Index:=0 to self.Kantenliste.Anzahl-1 do
    begin
      Ka:=TInhaltskante(self.Kantenliste.Kante(Index));
      if Ka.MausklickaufKante(X,Y) then
      begin
        self.EditiereKante(Ka,Aus,Abbruch);
        if  not Abbruch then
        begin
          if Ka.gerichtet and (not Aus) then
          begin
            Ka.Gerichtet:=true;
         Ka.Anfangsknoten.AusgehendeKantenListe.LoescheElement(Ka);
           Ka.Endknoten.EingehendeKantenliste.LoescheElement(Ka);
            self.Kantenliste.LoescheElement(Ka);
            Hilf:=Ka.AnfangsKnoten;
            Ka.Anfangsknoten:=Ka.Endknoten;
            Ka.Endknoten:=Hilf;
             FuegeKanteein(TInhaltsknoten(Ka.Anfangsknoten),TInhaltsknoten(Ka.Endknoten),
            Ka.Gerichtet,Ka);
          end;
```

```
        end;
      Gefunden:=true;
      goto Endproc;
    end;
  end;
  Endproc:
  if not Gefunden then ShowMessage('Keine Kante!');
  Kanteeditieren:=Gefunden;
end;

function TInhaltsgraph.Kanteninhaltzeigen(X,Y:Integer):Boolean;
label Endproc;
var Zaehl:Integer;
    Ka:TInhaltskante;
    Index:Integer;S:String;
begin
  Kanteninhaltzeigen:=false;
  if not self.Kantenliste.Leer then
    for Index:=0 to self.Kantenliste.Anzahl-1 do
    begin
      Ka:=TInhaltskante(self.Kantenliste.Kante(Index));
      if Ka.MausklickaufKante(X,Y) then
      begin
        Kanteninhaltzeigen:=true;
        S:='Inhalt: '+Ka.Wert+chr(13)+
        'Kante: '+Ka.Anfangsknoten.Wert+'-'+Ka.Endknoten.Wert;
        Kantenwertposition:=0;ShowMessage(S);goto Endproc;
      end;
    end;
    Endproc:
end;


procedure TInhaltsgraph.EingabeKnoten(Var Kno:TInhaltsknoten;Var
abbruch:Boolean);
var S:string;
    Eingabe:Boolean;
begin
  Abbruch:=true;
  S:='';
  Eingabe:=InputQuery('Zeichneneingabe:','Eingabe eines
Buchstabes:',S);
  if Eingabe then Abbruch:=false;
  if (not StringistRealZahl(S)) or (StringistRealZahl(S) and
  (Abs(StringtoReal(S))<1.0E30))
  then
    Kno.Wert:=S
  else
  begin
    ShowMessage('Fehler! Eingabe nicht im zulässigen numerischen
```

```
    Bereich!');
    Abbruch:=true;
  end;
end;

function TInhaltsgraph.ZweiKnotenauswaehlen(X,Y:Integer;var
Kno1,Kno2:TInhaltsKnoten;
var Gefunden:Boolean):Boolean;
label Endproc;
var K:TInhaltsknoten;
begin
  result:=false;
  Gefunden:=false;
  Kno1:=nil;
  Kno2:=nil;
  if K1<>nil then
  begin
    if not Leer  then
    begin
      if FindezuKoordinatendenKnoten(X,Y,K)=false then
      begin
        Gefunden:=false;
        ShowMessage('Kein Knoten');
        result:=true;
        K1:=nil;
        K2:=nil;
      end
      else
      begin
        K2:=K;
        K2.Graph:=self;
          if Abbruch then
          begin
            K1:=nil;
            K2:=nil;
            Gefunden:=false;
            result:=true;
            goto Endproc;
          end;
          if Abbruch then goto Endproc;
      Kno1:=K1;
      Kno2:=K2;
      Gefunden:=true;
      result:=true;
      K1:=nil;
      K2:=nil;
    end;
  end;
  end
  else
```

```
  begin
    if not Leer  then
    begin
      if FindezuKoordinatendenKnoten(X,Y,K)=false then
      begin
        Gefunden:=false;
        ShowMessage('Kein Knoten');
        K1:=nil;
        result:=true;
      end
      else
      begin
        K1:=K;
        K1.Graph:=self;
        result:=false;
       ShowMessage('2.Knoten wählen');
      end
    end;
  end;
  if Leer then result:=true;
  Endproc:
end;


function TInhaltsgraph.Knotenzeichnen(X,Y:Integer):Boolean;
var Abbruch:Boolean;
    K:TInhaltsknoten;
begin
  Knotenzeichnen:=false;
  K1:=self.Inhaltsknotenclass.Create;
  K1.Position:=self.Knotenwertposition;
  K:=K1;
  EingabeKnoten(K,Abbruch);
  K1:=K;
  if not Abbruch then
  begin
    K1.X:=X;
    K1.Y:=Y;
    K1.Radius:=Radius;
    FuegeKnotenein(K1);
    Knotenzeichnen:=true;
  end;
    K1:=nil;
end;


function TInhaltsgraph.Inhaltsknotenloeschen
(X,Y:Integer):Boolean;
var K:TInhaltsknoten;
begin
```

```
  Inhaltsknotenloeschen:=false;
  K:=TInhaltsknoten.Create;
  K.Wert:=' ';
  K.X:=X;
  K.Y:=Y;
  if FindezuKoordinatendenKnoten(X,Y,K) then
  begin
    K1:=K;
    self.Knotenloeschen(K1);
    Inhaltsknotenloeschen:=true;
  end
  else
    begin
      K1:=K;
      ShowMessage('Kein Knoten!');
    end;
  K1:=nil;
end;

function TInhaltsgraph.InhaltsKopiedesGraphen
(Inhaltsgraphclass:TInhaltsgraphclass;Inhaltsknotenclass:Tinhaltsknotenclass;
Inhaltskanteclass:TInhaltskanteclass;UngerichteterGraph:Boolean):TInhaltsgraph;
var Gra:TInhaltsgraph;
    Kna:TInhaltskante;
    Kno:Tinhaltsknoten;
    Stringliste:TStringList;
    Index,index1:Integer;
    Min:Integer;
begin
  Gra:=Inhaltsgraphclass.Create;
  Gra.Knotenwertposition:=Knotenwertposition;
  Gra.Kantenwertposition:=Kantenwertposition;
  Gra.Demo:=Demo;
  Gra.Pausenzeit:=Pausenzeit;
  Gra.Abbruch:=Abbruch;
  Gra.Zustand:=Zustand;
  Gra.Stop:=Stop;
  Gra.Liniendicke:=Liniendicke;
  Gra.MomentaneKnotenliste:=TKnotenliste.Create;
  Gra.MomentaneKantenliste:=TKantenliste.Create;
  Gra.Position:=Position;
  Gra.Knotengenauigkeit:=Knotengenauigkeit;
  Gra.Kantengenauigkeit:=Kantengenauigkeit;
  Gra.Graphistgespeichert:=Graphistgespeichert;
  Wertlisteschreiben;
  Gra.Wertliste.Clear;
  Gra.Wertlisteschreiben;
  Min:=Trunc(Minimum(Wertliste.Count,Gra.Wertliste.Count));
  for Index:=0 to Min-1 do
    Gra.Wertliste.Strings[Index]:=Wertliste.Strings[Index];
```

```
if Gra.Wertliste.Count>0 then Gra.Wertlistelesen;
Gra.Wertlisteschreiben;
if not self.Knotenliste.Leer then
  for Index:=0 to self.Knotenliste.Anzahl-1 do
    with TInhaltsknoten(self.Knotenliste.Knoten(Index)) do
    begin
      Kno:=Inhaltsknotenclass.Create;
      Kno.Graph:=Gra;
      Kno.X:=X;
      Kno.Y:=Y;
      Kno.Farbe:=Farbe;
      Kno.Stil:=Stil;
      Kno.Typ:=Typ;
      Kno.Radius:=Radius;
     Kno.Pfadliste:=TPfadliste.Create;
      Wertlisteschreiben;
      Kno.Wertliste.Clear;
      Kno.Wertlisteschreiben;
      Min:=Trunc(Minimum(Wertliste.Count,Kno.Wertliste.Count));
      for index1:=0 to Min-1 do
      Kno.Wertliste.Strings[index1]:=Wertliste.Strings[index1];
      if Kno.Wertliste.count>0 then Kno.Wertlistelesen;
      Kno.Wertlisteschreiben;
      Gra.Knotenliste.AmEndeanfuegen(Kno);
      end;
    if not self.Kantenliste.Leer then
      for Index:=0 to Kantenliste.Anzahl-1 do
       with TInhaltskante(Kantenliste.kante(Index)) do
        begin
        Kna:=Inhaltskanteclass.Create;
           Kna.Anfangsknoten:=Gra.Graphknoten(TInhaltsknoten(Anfangsknoten));
       Kna.Endknoten:=Gra.Graphknoten(TInhaltsknoten(Endknoten));
          Kna.Typ:=Typ;
          Kna.Wert:=Wert;
          Kna.Weite:=Weite;
          Kna.Farbe:=Farbe;
          if UngerichteterGraph
          then
            Kna.Gerichtet:=false
          else
            Kna.Gerichtet:=Gerichtet;
          Kna.Stil:=Stil;
          Wertlisteschreiben;
          Kna.Wertliste.Clear;
          Kna.Wertlisteschreiben;
       Min:=Trunc(Minimum(Wertliste.Count,Kna.Wertliste.Count));
          for Index1:=0 to Min-1 do
        Kna.Wertliste.Strings[Index1]:=Wertliste.Strings[Index1];
        if Kna.Wertliste.Count>0 then Kna.Wertlistelesen;
          Kna.Wertlisteschreiben;
```

```
            Gra.FuegeKanteein(self.Graphknoten(TInhaltsknoten(Kna.AnfangsKnoten)),
            self.Graphknoten(TInhaltsknoten(Kna.Endknoten)),Kna.Gerichtet,Kna);
          end;
    If LetzterMausklickknoten<>nil then
    Gra.LetzterMausklickknoten:=Graphknoten(LetzterMausklickknoten);
  InhaltsKopiedesGraphen:=Gra;
end;


function TInhaltsgraph.AnzahlTypknoten(Typ:char):Integer;
var Index,Typknoten:Integer;
begin
  Typknoten:=0;
  if not self.Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    if TInhaltsknoten(Knotenliste.Knoten(Index)).Typ=Typ then
    Typknoten:=Typknoten+1;
    AnzahlTypKnoten:=Typknoten;
end;

function TInhaltsgraph.AnzahlTypkanten(Typ:char):Integer;
var Index,Typkanten:Integer;
begin
  Typkanten:=0;
  if not self.Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
      if TInhaltskante(Kantenliste.Kante(Index)).Typ=Typ then
      Typkanten:=Typkanten+1;
  AnzahlTypKanten:=Typkanten;
end;


function TInhaltsgraph.AnzahlBruecken(var
SListe:TStringList;Ausgabe:TLabel;
  Flaeche:TCanvas):Integer;
label Endproc;
var Zaehl,Anzahl:Integer;
    Ka:TInhaltsKante;
begin
  Anzahl:=0;
  if not Kantenliste.Leer then
    for Zaehl:=0 to Kantenliste.Anzahl-1 do
    begin
      Application.ProcessMessages;
      if self.Abbruch then goto Endproc;
      TKante(Ka):=Kantenliste.Kante(Zaehl);
      Ausgabe.Caption:='Untersuchte Kante:
      '+Ka.Anfangsknoten.Wert+'-'+
      Ka.Endknoten.Wert;
      if (Ka.Anfangsknoten.AusgehendeKantenliste.Anzahl=1)or
```

```
      (Ka.Endknoten.AusgehendeKantenliste.Anzahl=1)or((not
       Ka.KanteistSchlinge) and
     (Ka.Anfangsknoten.PfadzumZielknoten(Ka.Endknoten,Ka)=false))
        then
        begin
         Anzahl:=Anzahl+1;
         Ka.Farbe:=clred;
         Ka.Stil:=psdot;
        Ka.ZeichneKante(Flaeche);
       SListe.Add(Tinhaltsknoten(Ka.Anfangsknoten).
       Wert+' '+TInhaltsknoten(Ka.Endknoten).Wert);
        end;
    end;
  Endproc:
  AnzahlBruecken:=Anzahl;
end;


function TInhaltsgraph.AlleKnotenbestimmen:TStringList;
var Index:Integer;
    Gstliste:TStringList;

  procedure Knotenangeben(X:TObject);
  begin
    Gstliste.Add(TInhaltsknoten(X).Wert+' Koordinaten:  X='+
    InttoStr(TInhaltsknoten(X).X)+'
    Y='+InttoStr(TInhaltsknoten(X).Y)
    +' Knotengrad (gerichtet/ungerichtet):
    '+IntToStr(TKnoten(X).Grad(true))+'/'
    +IntToStr(TKnoten(X).Grad(false))+' Typ:
    '+Tinhaltsknoten(X).Typ);
  end;

begin
  Gstliste:=TStringList.Create;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      Knotenangeben(Knotenliste.Knoten(Index));
  AlleKnotenbestimmen:=Gstliste;
end;


function TInhaltsgraph.AlleKantenbestimmen:TStringList;
var Index:Integer;
    Gstliste:TStringList;

  procedure Kantenangeben(X:TObject);
  var Str:string;
      Kn:TinhaltsKante;
  begin
```

```
    Kn:=TInhaltsKante(X);
    Str:='Kante: '+TInhaltsknoten(Kn.Anfangsknoten).Wert+'
    '+TInhaltsknoten(Kn.Endknoten).
      Wert;
    Str:=Str+' Inhalt: '+TInhaltskante(Kn).Wert+' ';
    Str:=Str+'Typ: '+Kn.Typ;
    if Kn.gerichtet then Str:=Str+' gerichtet';
    if not Kn.gerichtet then str:=str+' ungerichtet';
    if Kn.KanteistSchlinge and Kn.gerichtet then str:=str+' ge
    richtete Schlinge';
    if Kn.KanteistSchlinge and not Kn.gerichtet then Str:=Str+'
    ungerichtete Schlinge';
    Gstliste.Add(Str);
  end;

begin
  GStliste:=TStringList.Create;
  if not self.Kantenliste.Leer then
  for Index:=0 to self.Kantenliste.Anzahl-1 do
    Kantenangeben(self.Kantenliste.Kante(Index));
  AlleKantenbestimmen:=Gstliste;
end;

function WerteinerKanteimKreis(X:TObject):Extended;
begin
  WerteinerKanteimKreis:=1;
end;


function TInhaltsgraph.AnzahlKnotenkleinsterKreis(var
St:string;Flaeche:TCanvas):Integer;
label Endproc;
var Index,Anzahl:Integer;
    Kno:TKnoten;
    S,T:TGraph;
    Kant:TKantenliste;
begin
  LoescheKantenbesucht;
  AnzahlKnotenkleinsterKreis:=0;
  Anzahl:=AnzahlKanten+1;
  St:='';
  S:=nil;
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Application.Processmessages;
      if Abbruch then
      begin
        Anzahl:=0;
        goto Endproc;
```

```
      end;
      Kno:=Knotenliste.Knoten(Index);
      Kant:=TInhaltsknoten(Kno).ErzeugeMinmaxKreise(true);
      T:=TGraph(Kant.Graph);
      if not T.Leer then
      If (T.AnzahlKanten<Anzahl)and (T.AnzahlKanten>2) then
      begin
        Anzahl:=T.AnzahlKanten;
        S:=T;
        if Anzahl=3 then goto Endproc;
      end;
    end;
  Endproc:
  if  Anzahl=AnzahlKanten+1
    then
    begin
      Anzahl:=0;
      St:='';
    end
    else
      if not S.Kantenliste.Leer then
      begin
        St:='';
        for Index:=0 to S.Kantenliste.Anzahl-1  do
         St:=St+TInhaltskante(S.Kantenliste.Kante(Index)).Wert+'
           ';
        TInhaltsgraph(S).FaerbeGraph(clred,psdot);
        TInhaltsgraph(S).ZeichneGraph(Flaeche);
        Showmessage('Kanten kleinster Kreis: '+St+' Kantenzahl:
        '+Integertostring(Anzahl));
        TInhaltsgraph(S).FaerbeGraph(clblack,pssolid);
        TInhaltsgraph(S).ZeichneGRaph(Flaeche);
      end;
  FaerbeGraph(clblack,pssolid);
  AnzahlKnotenkleinsterKreis:=Anzahl;
end;

function TInhaltsgraph.AnzahlKnotengroesterKreis(var
St:string;Flaeche:TCanvas):Integer;
label Endproc;
var Index,Anzahl:Integer;
    Kno:TKnoten;
    S,T:TGraph;
    Kant:TKantenliste;
begin
  LoescheKantenbesucht;
  Anzahl:=0;
  S:=nil;
  St:='';
  if not Knotenliste.Leer then
```

```
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Application.ProcessMessages;
      if Abbruch then
      begin
        Anzahl:=0;
        goto Endproc;
      end;
      Kno:=Knotenliste.Knoten(Index);
      Kant:=TInhaltsknoten(Kno).ErzeugeminmaxKreise(false);
      T:=TGraph(Kant.Graph);
      if not T.Leer then
      If T.AnzahlKanten>Anzahl then
      begin
        Anzahl:=T.AnzahlKanten;
        S:=T;
        if Anzahl=AnzahlKanten then goto Endproc;
      end;
    end;
    if  Anzahl=0
    then
      St:=''
    else
      if not S.Kantenliste.Leer then
      begin
        St:='';
        for Index:=0 to S.Kantenliste.Anzahl-1  do
        St:=St+TInhaltskante(S.Kantenliste.Kante(Index)).Wert+'
           ';
        TInhaltsgraph(S).FaerbeGraph(clred,psdot);
        TInhaltsgraph(S).ZeichneGRaph(Flaeche);
        Showmessage('Kanten größter Kreis:'+St+' Kantenzahl:
        '+Integertostring(Anzahl));
        TInhaltsgraph(S).FaerbeGraph(clblack,pssolid);
        TInhaltsgraph(S).ZeichneGRaph(Flaeche);
      end;
  Endproc:
  FaerbeGraph(clblack,pssolid);
  AnzahlKnotengroesterKreis:=Anzahl;
end;


function TInhaltsgraph.KreisefesterLaenge(Laenge:Integer;var
Sliste:TStringlist;
Flaeche:TCanvas;Ausgabe:TLabel):Integer;
label Endproc;
var Index,Anzahl:Integer;
    Kno:TKnoten;
    T:TInhaltsgraph;
begin
```

```
  T:=nil;
  LoescheKantenbesucht;
  Anzahl:=0;
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Application.ProcessMessages;
      if Abbruch then
      begin
        Anzahl:=0;
        goto Endproc;
      end;
      Kno:=Knotenliste.Knoten(Index);
     TInhaltsknoten(Kno).ErzeugeKreisevonfesterLaenge(Laenge);
      if not TInhaltsknoten(Kno).Pfadliste.leer then
      begin
     TInhaltsknoten(Kno).AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
        Anzahl:=Anzahl+TInhaltsknoten(Kno).Pfadliste.Anzahl;
        TObject(T):=Kno.Pfadliste.Pfad(Kno.Pfadliste.Letztes);
      end;
    end;
 if T<>nil
   then
   begin
     T.FaerbeGraph(clred,psdot);
     T.ZeichneGraph(Flaeche);
   end;
   Endproc:
   KreisefesterLaenge:=Anzahl;
  end;

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;
```

```
function ErzeugeKnotenstring(Ob:TObject):string;
begin
  ErzeugeKnotenstring:=TInhaltsknoten(Ob).Wert
end;

function ErzeugeKantenstring(Ob:TObject):string;
begin
  ErzeugeKantenstring:=TInhaltskante(Ob).Wert;
end;

procedure LoescheBild(var G:TInhaltsgraph;var
Oberflaeche:TForm);
var P:TInhaltsgraph;
begin
  P:=G;
  G:=TInhaltsgraph.Create;
  Oberflaeche.Refresh;
  G:=P;
end;

end.
```

Unit UKante:

```
unit UKante;
{$F+}
interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
Controls,
  Forms, Dialogs, StdCtrls;

type
  TKantenform = class(TForm)
    Checkausgehend: TCheckBox;
    Checkeingehend: TCheckBox;
    CheckInteger: TCheckBox;
    CheckReal: TCheckBox;
    Inhalt: TEdit;
    Weite: TEdit;
    ScrollBar: TScrollBar;
    OK: TButton;
    Abbruch: TButton;
    Textausgehend: TLabel;
    TextEingehend: TLabel;
    TextInteger: TLabel;
    TextReal: TLabel;
    TextWeite: TLabel;
    TextInhalt: TLabel;
```

```
    procedure OKClick(Sender: TObject);
    procedure InhaltChange(Sender: TObject);
    procedure CheckIntegerClick(Sender: TObject);
    procedure CheckRealClick(Sender: TObject);
    procedure InhaltKeyPress(Sender: TObject;var Key:char);
    procedure FormActivate(Sender: TObject);
    procedure AbbruchClick(Sender: TObject);
    procedure WeiteKeyPress(Sender: TObject;var Key:char);
    procedure ScrollBarChange(Sender: TObject);
    procedure WeiteChange(Sender: TObject);
  private
    Kanteein_,Kanteaus_:Boolean;
    KantenInteger_,Kantenreal_:Boolean;
    Kanteninhalt_:string;
    Kantenweite_:Integer;
    Kantenabbruch_:Boolean;
    function WelchesKantenein:Boolean;
    procedure SetzeKantenein(Ke:Boolean);
    function WelchesKantenaus:Boolean;
    procedure SetzeKantenaus(Ka:Boolean);
    function WelchesKantenInteger:Boolean;
    procedure SetzeKantenInteger(Ki:Boolean);
    function WelchesKantenreal:Boolean;
    procedure SetzeKantenreal(Kr:Boolean);
    function WelcherKanteninhalt:string;
    procedure SetzeKanteninhalt(Ki:string);
    function WelcheKantenweite:Integer;
    procedure SetzeKantenweite(Kw:Integer);
    function WelcherKantenabbruch:Boolean;
    procedure SetzeKantenabbruch(Ab:Boolean);
  public
    constructor Create(AOwner: TComponent);
    property Kanteein:Boolean read WelchesKantenein write
    SetzeKantenein;
    property Kanteaus:Boolean read WelchesKantenaus write
    SetzeKantenaus;
    property KantenInteger:Boolean read WelchesKanteninteger
    write SetzeKantenInteger;
    property Kantenreal:Boolean read WelchesKantenreal write
    SetzeKantenreal;
    property Kanteninhalt:string read WelcherKanteninhalt write
    SetzeKanteninhalt;
    property Kantenweite:Integer read WelcheKantenweite write
setzeKantenweite;
    property Kantenabbruch:Boolean read WelcherKantenabbruch
write setzeKantenabbruch;
  end;

var
  Kantenform: TKantenform;
```

```
implementation

{$R *.DFM}

procedure TKantenform.OKClick(Sender: TObject);
label Marke;
var J,Code:Integer;
    R:Extended;
    S:string;
    W:Integer;
begin
  if CheckInteger.Checked and CheckReal.Checked then
  begin
    CheckInteger.Checked:=False;
    CheckReal.Checked:=False;
  end;
  Kanteninhalt:=' ';
  Kanteninhalt:=Inhalt.Text;
  if (Kanteninhalt=' ') or (Kanteninhalt='') then goto Marke;
  while Kanteninhalt[1]=' ' do
  begin
    Application.Processmessages;
    S:=Kanteninhalt;
    Delete(S,1,1);
    Kanteninhalt:=S;
  end;
  while Kanteninhalt[length(Kanteninhalt)]=' ' do
  begin
    Application.ProcessMessages;
    S:=Kanteninhalt;
    Delete(S,length(Kanteninhalt),1);
    Kanteninhalt:=S;
  end;
  val(Kanteninhalt,J,Code);
  if (CheckInteger.Checked=true) and (Code<>0) then
  begin
    ShowMessage('Fehler! Eingabe nicht im zuläsigen
    numerischen'+chr(13)+
      'Bereich oder falsche Zeichen!');
    exit;
  end;
  Marke:
  if (Kanteninhalt = ' ')or  (Kanteninhalt='') then
   if CheckInteger.Checked=true  then Kanteninhalt:='1.0' else
   Kanteninhalt:=' ';
  val(Kanteninhalt,R,Code);
  if (CheckReal.Checked=true) and (Code<>0) then
  begin
    ShowMessage('Fehler! Eingabe nicht im zuläsigen
```

```
    numerischen’+chr(13)+
      ‘Bereich oder falsche Zeichen!’);
    exit;
  end;
  if Kanteninhalt = ‘ ‘ then
    if CheckReal.Checked=true  then Kanteninhalt:=’1.0' else
    Kanteninhalt:=’ ‘;
  Kanteaus:=Checkausgehend.Checked;
  Kanteein:=Checkeingehend.Checked;
  if (Kanteein) and (Kanteaus) then
  begin
    Kanteein:=false;
    Kanteaus:=false;
  end;
  Kantenweite:=0;
  W:=Kantenweite;
  val(Weite.Text,W,Code);
  Kantenweite:=W;
  if Code=0 then
    Kantenweite:=strtoint(Weite.Text);
  Kantenabbruch:=false;
  KantenInteger:=CheckInteger.Checked;
  KantenReal:=CheckReal.Checked;
  Kantenform.close
end;


procedure TKantenform.InhaltChange(Sender: TObject);
begin
  Kanteninhalt:=Inhalt.Text;
end;


procedure TKantenform.CheckIntegerClick(Sender: TObject);
begin
  CheckReal.Checked:=false;
end;

procedure TKantenform.CheckRealClick(Sender: TObject);
begin
 CheckInteger.Checked:=false;
end;
procedure TKantenform.InhaltKeyPress(Sender: TObject;var Key:
Char);
begin
  if ord(key)=13 then OKClick(Sender);
end;

procedure TKantenform.FormActivate(Sender: TObject);
begin
```

```
  Inhalt.Text:=Kanteninhalt;
  Weite.Text:=IntToStr(Kantenweite);
  Checkeingehend.Checked:=Kanteein;
  Checkausgehend.Checked:=Kanteaus;
  CheckReal.Checked:=KantenReal;
  CheckInteger.Checked:=KantenInteger;
end;

procedure TKantenform.AbbruchClick(Sender: TObject);
begin
  Kantenabbruch:=true;
  Kantenform.Close;
end;

procedure TKantenform.WeiteKeyPress(Sender: TObject; var Key:
Char);
begin
  if ord(key)=13 then OKClick(Sender);
end;
procedure TKantenform.ScrollBarChange(Sender: TObject);
begin
  Weite.Text:=InttoStr(Scrollbar.Position);
end;

procedure TKantenform.WeiteChange(Sender: TObject);
var Fehler,Zahl:Integer;
begin
  val(Weite.Text,Zahl,Fehler);
  if Fehler<>0 then Zahl:=0;
  if (Zahl>=-500)and(Zahl<=500) then
  Scrollbar.Position:=Zahl;
end;

constructor TKantenform.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Kanteein_:=false;
  Kanteaus_:=false;
  KantenInteger_:=false;
  Kantenreal_:=false;
  Kanteninhalt_:=' ';
  Kantenweite_:=0;
  Kantenabbruch_:=false;
end;


function TKantenform.WelchesKantenein:Boolean;
begin
  WelchesKantenein:=Kanteein_;
end;
```

```
procedure TKantenform.SetzeKantenein(Ke:Boolean);
begin
  Kanteein_:=Ke;
end;

function TKantenform.WelchesKantenaus:Boolean;
begin
  WelchesKantenaus:=Kanteaus_;
end;

procedure TKantenform.SetzeKantenaus(Ka:Boolean);
begin
  Kanteaus_:=Ka;
end;

function TKantenform.WelchesKantenInteger:Boolean;
begin
  WelchesKanteninteger:=Kanteninteger_;
end;

procedure TKantenform.SetzeKantenInteger(Ki:Boolean);
begin
  KantenInteger_:=Ki;
end;

function TKantenform.WelchesKantenreal:Boolean;
begin
  WelchesKantenreal:=Kantenreal_;
end;

procedure TKantenform.SetzeKantenreal(Kr:Boolean);
begin
  Kantenreal_:=Kr;
end;

function TKantenform.WelcherKanteninhalt:string;
begin
  WelcherKanteninhalt:=Kanteninhalt_;
end;


procedure TKantenform.SetzeKanteninhalt(Ki:string);
begin
  Kanteninhalt_:=Ki;
end;

function TKantenform.WelcheKantenweite:Integer;
begin
  WelcheKantenweite:=Kantenweite_;
```

```
end;

procedure TKantenform.SetzeKantenweite(Kw:Integer);
begin
  Kantenweite_:=Kw;
end;


function TKantenform.WelcherKantenabbruch:Boolean;
begin
  WelcherKantenabbruch:=Kantenabbruch_;
end;

procedure TKantenform.SetzeKantenabbruch(Ab:Boolean);
begin
  Kantenabbruch_:=Ab;
end;


end.
```

Unit UAusgabe:

```
unit UAusgabe;
{$X+}
{$F+}
interface

uses

  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls,
  Forms, Dialogs, Menus,Stdctrls, Grids,Clipbrd,Printers;

type
  TAusgabeform = class(TForm)
    Gitternetz: TStringGrid;
    MainMenu: TMainMenu;
    Ende: TMenuItem;
    Kopieren: TMenuItem;
    Drucken: TMenuItem;
    PrintDialog: TPrintDialog;
    procedure EndeClick(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure KopierenClick(Sender: TObject);
    procedure DruckenClick(Sender: TObject);
  private
    { Private-Deklarationen }
    Listenbox_:TListbox;
```

```
    function WelcheListbox:TListbox;
    procedure SetzeListbox(Lb:TListbox);
  public
    { Public-Deklarationen }
    constructor Create(AOwner:TComponent);
    property Listenbox:TListbox read WelcheListbox write
SetzeListbox;
  end;

var
  Ausgabeform: TAusgabeform;


implementation

{$R *.DFM}

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

procedure TAusgabeform.KopierenClick(Sender: TObject);
label Endproc;
var Index:Integer;
    S:string;
    Stliste:TStringlist;
    Grenze:Integer;
begin
  Stliste:=TStringlist.Create;
  if Listenbox.Items.Count-1<254
  then
    Grenze:=Listenbox.Items.Count-1
  else
    Grenze:=254;
  if Grenze>-1 then
  for Index:=0 to Grenze do
  begin
    S:=Listenbox.Items[Index];
    stliste.Add(S);
  end
    else if Grenze=-1 then Stliste.Add('leere Ausgabe');
  Clipboard.SettextBuf(Stliste.Gettext);
  if Listenbox.Items.Count-1>254 then ShowMessage
  ('Nur die ersten 255 Zeilen wurden kopiert!');
  Stliste.Free;
  Stliste:=niL;
end;

procedure TAusgabeform.DruckenClick(Sender: TObject);
```

```
label Endproc;
var Index:Integer;
    S:string;
    Stliste:TStringlist;
    Grenze:Integer;
    MyFile: TextFile;
begin
  Stliste:=TStringlist.Create;
  if Listenbox.Items.Count-1<254
  then
    Grenze:=Listenbox.Items.Count-1
  else
    Grenze:=254;
  if Grenze>-1 then
    if PrintDialog.Execute then
      for Index:=0 to Grenze do
      begin
        S:=Listenbox.Items[Index];
        Stliste.Add(S);
      end
      else if Grenze=-1 then Stliste.Add('leere Ausgabe');
  AssignPrn(MyFile);
  Rewrite(MyFile);
  Writeln(MyFile,Stliste.Gettext);
  System.CloseFile(MyFile);
  if Listenbox.Items.Count-1>254 then ShowMessage
  ('Nur die ersten 255 Zeilen wurden gedruckt!');
  Stliste.Free;
  Stliste:=nil;
end;


procedure TAusgabeform.FormPaint(Sender: TObject);
var Index:Integer;
     S:string;
begin
  for Index:=1 to Gitternetz.Rowcount do
   Gitternetz.cells[0,index]:='';
  if Listenbox.Items.Count>15 then
   Gitternetz.Rowcount:=Listenbox.Items.Count;
  if Listenbox.Items.Count=0 then
  Ausgabeform.Gitternetz.Cells[0,0]:='leere Ausgabe';
  for Index:=0 to Listenbox.Items.Count-1 do
  begin
    Ausgabeform.Canvas.Pen.Color:=clblack;
    S:=Listenbox.Items[Index];
    Ausgabeform.Gitternetz.Cells[0,Index]:=S;
  end;
end;
```

```
function TAusgabeform.WelcheListbox:TListbox;
begin
  WelcheListbox:=Listenbox_;
end;

procedure TAusgabeform.SetzeListbox(Lb:Tlistbox);
begin
  Listenbox_:=Lb;
end;

constructor TAusgabeform.Create(AOwner:TComponent);
begin
  inherited Create(AOwner);
  Listenbox_:=nil;
end;

end.
```

Unit UPfad:

```
unit UPfad;(nur für DWK)
{$F+}

interface

uses

  UList,UInhGrph,Ugraph,UKante,
  StdCtrls, Controls, ExtCtrls, Dialogs,
  Menus, Classes,
  SysUtils, WinTypes, WinProcs, Messages,Graphics,
  Forms,ClipBrd,Printers;

type

  TPfadknoten = class(TInhaltsknoten)
  public
    constructor create;override;
    procedure Free;
    procedure Freeall;
    procedure ErzeugeallePfade;
    procedure BinaererBaumInorder;
    procedure ErzeugeKreise;
    procedure ErzeugeminimalePfadenachDijkstra(Flaeche:TCanvas);
    procedure ErzeugeTiefeBaumPfade(Preorder:Boolean);
    function AnzahlPfadZielknoten: Integer;
    procedure ErzeugeWeiteBaumPfade;
    procedure BinaeresSuchen(Kno:TKnoten);
    procedure ErzeugeallePfadeZielknoten(Kno:TKnoten);
```

```
    procedure ErzeugeTiefeBaumPfadeeinfach
    (Preorder:Boolean;Flaeche:TCanvas;Ausgabe:TLabel;
    var SListe:TStringlist);
    procedure ErzeugeminimalePfade;
    function KnotenistKreisknoten:Boolean;
    procedure ErzeugeAllePfadeundMinimalenPfad
    (ZKno:TPfadKnoten;var Minlist:TKantenliste);
  end;

  TPfadgraph = class(TInhaltsgraph)
  public
    constructor Create;override;
    procedure Free;
    procedure Freeall;
    procedure AllePfadevoneinemKnotenbestimmen(X,Y:Integer;
    Ausgabe:Tlabel;var SListe:TStringList;Flaeche:TCanvas);
    procedure AlleKreisevoneinemKnotenbestimmen(X,Y:Integer;
    Ausgabe:TLabel;var SListe:TStringList;Flaeche:TCanvas);
    procedure AlleminimalenPfadevoneinenKnotenbestimmen
    (X,Y:Integer;Ausgabe:Tlabel;var
    SListe:TStringList;Flaeche:TCanvas);
    procedure AlletiefenBaumpfadevoneinemKnotenbestimmen
    (X,Y:Integer;Ausgabe:TLabel;var
    SListe:TStringList;Flaeche:TCanvas);
    procedure AlleweitenBaumpfadevoneinemKnotenbestimmen
    (X,Y:Integer;Ausgabe:TLabel;var
    SListe:TStringList;Flaeche:TCanvas);
    function BestimmeminimalenPfad(Kno1,Kno2:TKnoten;Wert:TWert;
    Flaeche:TCanvas):TPfad;
    function MinimalenPfadzwischenzweiKnotenbestimmen
    (X,Y:Integer;Ausgabe:TLabel;var SListe:TStringList;
    Flaeche:TCanvas):Boolean;
    function AllePfadezwischenzweiKnotenbestimmen(X,Y:Integer;
    Ausgabe:TLabel;var SListe:TStringList;
    Flaeche:TCanvas):Boolean;
    procedure Kruskal(Ausgabe:TLabel;var
    SListe:TStringList;Flaeche:TCanvas);
    procedure AlletiefenBaumpfadevoneinemKnotenbestimmeneinfach
    (X,Y:Integer;Ausgabe:TLabel;var
    SListe:TStringList;Flaeche:TCanvas);
    function GraphhatKreise:Boolean;
    function AllePfadeundminimalenPfadzwischenzwei
   Knotenbestimmen
   (X,Y:Integer;Ausgabe:TLabel;var
   SListe:TStringList;Flaeche:TCanvas):Boolean;
  end;


implementation
```

```
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;

function Groesser(Ob1,Ob2:TObject;Wert:TWert):Boolean;
begin
  Groesser:=Wert(OB2)>Wert(Ob1);
end;

constructor TPfadknoten.Create;
begin
  inherited Create;
end;

procedure TPfadknoten.Free;
begin
  inherited Free;
end;

procedure TPfadknoten.Freeall;
begin
  inherited Freeall;
end;

procedure TPfadgraph.Free;
begin
  inherited Free;
end;

procedure TPfadgraph.Freeall;
begin
  inherited Freeall;
end;

constructor TPfadgraph.Create;
begin
  inherited Create;
end;
{Menü: Alle Pfade}


procedure TPfadknoten.BinaererBaumInorder;
label Endproc;
var MomentaneKantenliste:TKantenliste;
    P:TGraph;
    Ob:TObject;
```

```
  procedure GehezuNachbarknoten(Kn:TKnoten;Ka:TKante);
  Label Endproc;
  var Ob:TObject;
  begin
    Application.Processmessages;
    if Graph.Abbruch then goto Endproc;
    if Ka.Zielknoten(Kn).AusgehendeKantenliste.Anzahl>0
    then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kn);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
      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);
    MomentaneKantenliste.AmEndeloeschen(Ob);
    if Ka.Zielknoten(Kn).AusgehendeKantenliste.Anzahl>1 then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kn);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
      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));
    P:=TGraph.Create;
    P.Knotenliste.AmEndeanfuegen(self);
    Pfadliste.AmEndeanfuegen(P);
  if AusgehendeKantenliste.Anzahl>1
  then
    GehezuNachbarknoten(self,AusgehendeKantenliste.Kante(1));
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
end;


procedure TPfadknoten.ErzeugeallePfade;
```

```
var Index:Integer;
    MomentaneKantenliste:TKantenliste;

  procedure GehezuallenNachbarknoten(Kno:TKnoten;Ka:TKante);
  label Endproc;
  var Ob:TObject;
      Index:Integer;
  begin Application.Processmessages;
   if TInhaltsgraph(Graph).Abbruch then goto Endproc;
    if TInhaltsgraph(Graph).Stop then goto Endproc;
    if not Ka.Zielknoten(Kno).Besucht then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kno);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
      Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
      if Pfadliste.Anzahl>10000 then
      begin
        ShowMessage('Mehr als 10000 Pfade!Abbruch!');
        TInhaltsgraph(Graph).Stop:=true;
        goto Endproc;
      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;
    Endproc:
  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;



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(self.Anfangsknoten);
   if GraphistBinaerbaum and (MessageDlg('Inorder-Durchlauf in
   einem geordnetem Binärbaum?',mtConfirmation, [mbYes, mbNo],
    0) = mrYes)
    then
      Kno.BinaererBaumInorder
    else
    Kno.ErzeugeallePfade;
    if Kno.Pfadliste.Leer
    then
      ShowMessage('Keine Pfade')
    else
      Kno.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
  end;

  self.Pfadlistenloeschen;
end;

{Menü: Alle Kreise}

procedure TPfadknoten.ErzeugeKreise;
var Index:Integer;
    MomentaneKantenliste:TKantenliste;

  procedure GehezudenNachbarknoten(Kno:TKnoten;Ka:TKante);
  label Endproc;
  var Ob:TObject;
      Index:Integer;

  begin  Application.Processmessages;
    if TInaltsgraph(Graph).Abbruch then goto Endproc;
    if TInaltsgraph(Graph).Stop then goto Endproc;
    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);
        if Pfadliste.Anzahl>10000 then
        begin
          ShowMessage('Mehr als 10000 Kreise!Abbruch!');
          TInhaltsgraph(Graph).Stop:=true;
          goto Endproc;
        end;
        MomentaneKantenliste.AmEndeloeschen({Y}Ob);
      end;
    Endproc:
  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;



procedure TPfadgraph.AlleKreisevoneinemKnotenbestimmen
(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.ErzeugeKreie;
    if Kno.Pfadliste.Leer
    then
      ShowMessage('Keine Pfade')
    else
      Kno.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
  end;

  Pfadlistenloeschen;
end;

{Menü: Minimale Pfade}

procedure
TPfadknoten.ErzeugeminimalePfadenachDijkstra(Flaeche:TCanvas);
label Endproc;
var WegPfadliste:TPfadliste;
     MomentanerWeg,MomentanerWegneu:TKantenliste;
     Index:Integer;
     Ka: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;
    if TInhaltsgraph(Graph).Demo then
    begin
      TInhaltsgraph(TKantenliste(MomentanerWeg).Graph).FaerbeGraph(clgreen,psdot);
```

```
      TInhaltsgraph(TKantenliste(MomentanerWeg).Graph).ZeichneGraph(Flaeche);
    TInhaltsgraph(Graph).Demopause;
     TInhaltsgraph(TKantenliste(MomentanerWeg).Graph).FaerbeGraph(clblack,pssolid);
     TInhaltsgraph(TKantenliste(MomentanerWeg).Graph).ZeichneGraph(Flaeche);
    end;
    Ka:=MomentanerWeg.Kante(MomentanerWeg.Letztes);
    Kno:=Ka.Pfadrichtung;
    if not Kno.Besucht
    then
    begin
      Kno.Besucht:=true;
      Kno.Pfadliste.AmEndeanfuegen(MomentanerWeg.Kopie.Graph);
      Pfadliste.AmEndeanfuegen(MomentanerWeg.Kopie.Graph);
      if TInhaltsgraph(Graph).Demo then
      begin
        TInhaltsgraph(TKantenliste(MomentanerWeg).Graph).FaerbeGraph(clred,psdot);
        TInhaltsgraph(TKantenliste(MomentanerWeg).Graph).ZeichneGraph(Flaeche);
      TInhaltsgraph(Graph).Demopause;
        TInhaltsgraph(TKantenliste(MomentanerWeg).Graph).FaerbeGraph(clblack,pssolid);
        TInhaltsgraph(TKantenliste(MomentanerWeg).Graph).ZeichneGraph(Flaeche);
      end;
      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);
          MomentanerWegneu:=MomentanerWeg.Kopie;
          MomentanerWegneu.AmEndeanfuegen(Ka);
         WegPfadliste.AmAnfanganfuegen(Momentanerwegneu.Graph);
          end;
        end;
    end
    else
    begin
      MomentanerWeg.Free;
      MomentanerWeg:=nil;
    end;
    WegPfadliste.Sortieren(Pfadvergleich,Bewertung);
  end;
  Endproc:
  Wegpfadliste.Freeall;
  Wegpfadliste:=nil;
end;


procedure TPfadgraph.AlleminimalenPfadevoneinenKnotenbestimmen
(X,Y:Integer;Ausgabe:Tlabel;varSListe:TStringList;Flaeche:TCanvas);
```

```
var Zaehl:Integer;
    T:TInhaltsgraph;
    Kno,K:TPfadknoten;
begin
  if not Leer  then
  begin
    Kno:=TPfadknoten.Create;
    K:=Kno;
    if FindezuKoordinatendenKnoten
   (X,Y,TInhaltsknoten(Kno))=false
    then
      Kno:=TPfadknoten(Anfangsknoten);
    Kno.ErzeugeminimalePfadenachDijkstra(Flaeche);
    if Demo then Showmessage('Ausgabe der Ergebnisse');
    if Kno.Pfadliste.Leer
    then
     ShowMessage('Keine Pfade')
    else
    Kno.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
  end;

  Pfadlistenloeschen;
  K.Free;
  K:=nil;
end;

{Menü: Tiefe Baumpfade}

procedure TPfadknoten.ErzeugeTiefeBaumPfade(Preorder:Boolean);
var Index:Integer;
    MomentaneKantenliste:TKantenliste;
    P:TGraph;

  procedure GehezuNachbarknoten(Kno:TKnoten;Ka:TKante);
  label Endproc;
  var Ob:TObject;
      Index:Integer;
  begin Application.Processmessages;
    if TInaltsgraph(Graph).Abbruch then goto Endproc;
    if Tinhaltsgraph(Graph).Stop then goto Endproc;
    if not Ka.Zielknoten(Kno).Besucht then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kno);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
      if preorder then Pfadliste.AmEndeanfuegen
     (MomentaneKantenliste.Kopie.Graph);
      if Pfadliste.Anzahl>10000 then
      begin
        ShowMessage('Mehr als 10000 Pfade!Abbruch!');
        TInhaltsgraph(Graph).Stop:=true;
```

```
        goto Endproc;
      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
       GehezuNachbarknoten(Ka.Zielknoten(Kno),Ka.Zielknoten(Kno).
          AusgehendeKantenliste.Kante(Index));
      if not preorder then
      Pfadliste.AmEndeanfuegen(MomentaneKantenliste.
       Kopie.Graph);
      MomentaneKantenliste.AmEndeloeschen(Ob);
    end;
    Endproc:
  end;


begin
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.Pfadlistenloeschen;
  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;
  P:=TGraph.Create;
  P.Knotenliste.AmEndeanfuegen(self);
  if Preorder
  then
    Pfadliste.AmAnfanganfuegen(P)
  else
    Pfadliste.AmEndeanfuegen(P);
end;



procedure TPfadgraph.AlletiefenBaumpfadevoneinemKnotenbestimmen
(X,Y:Integer;Ausgabe:TLabel;var
SListe:TStringList;Flaeche:TCanvas);

var T:TInhaltsgraph;
    Zaehl:Integer;
    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.ErzeugetiefeBaumpfade(false)
    else
      Kno.ErzeugetiefeBaumpfade(true);
    if Kno.Pfadliste.Leer
    then
      ShowMessage('Keine Pfade')
    else
      Kno.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
  end;

  Pfadlistenloeschen;
end;

{Menü: Anzahl Zielknoten}

function TPfadknoten.AnzahlPfadZielknoten: Integer;
begin
  Graph.Pfadlistenloeschen;
  ErzeugeTiefeBaumpfade(true);
  AnzahlPfadZielknoten:=Pfadliste.Anzahl-1;
end;


{Menü: Weiter Baum}


procedure TPfadknoten.ErzeugeWeiteBaumPfade;
var Ob1,Ob2:TObject;
    Index:Integer;
    Kantenliste:TKantenliste;
    Knotenliste:TKnotenliste;
    MomentaneKantenliste:TKantenliste;
    P:TGraph;


  procedure SpeichereNachbarknoten(Kno:TKnoten;Ka:TKante);
  label Endproc;
  var Hilfliste:TKantenliste;
      Index:Integer;
  begin Application.Processmessages;
    if Tinhaltsgraph(Graph).Abbruch then goto Endproc;
    if Tinhaltsgraph(Graph).Stop then goto Endproc;
    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;
      MomentaneKantenliste.AmEndeanfuegen(Ka);
        Ka.Zielknoten(Kno).Pfadliste.AmAnfanganfuegen(MomentaneKantenliste.
       Kopie.UGraph);
     Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.UGraph);
      if Pfadliste.Anzahl>10000 then
      begin
        ShowMessage('Mehr als 10000 Pfade!Abbruch!');
        Tinhaltsgraph(Graph).Stop:=true;
        goto Endproc;
      end;
      Ka.Zielknoten(Kno).Besucht:=true;
      Kantenliste.AmAnfanganfuegen(Ka);
      Knotenliste.AmAnfanganfuegen(Ka.Zielknoten(Kno));
    end;
  Endproc:
end;


begin
  Graph.Pfadlistenloeschen;
  Graph.LoescheKnotenbesucht;
  Besucht:=false;
  Kantenliste:=TKantenliste.Create;
  Knotenliste:=TKnotenliste.Create;
  Besucht:=true;
  P:=TGraph.Create;
  P.Knotenliste.AmEndeanfuegen(self);
  Pfadliste.AmAnfanganfuegen(P);
  if not AusgehendeKantenliste.Leer then
    for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
    SpeichereNachbarknoten(self,AusgehendeKantenliste.Kante(Index));
  while not Knotenliste.Leer do
  begin
    Kantenliste.AmEndeloeschen(Ob1);
    Knotenliste.AmEndeloeschen(Ob2);
   TKante(Ob1).Pfadrichtung:=TKante(Ob1).Zielknoten(TKnoten(Ob2));
    if not TKnoten(Ob2).AusgehendeKantenliste.Leer then
      for Index:=0 to TKnoten(Ob2).AusgehendeKantenliste.
       Anzahl-1 do
      SpeichereNachbarknoten(TKante(Ob1).Zielknoten(TKnoten(Ob2)),
    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;


procedure TPfadgraph.AlleweitenBaumpfadevoneinemKnotenbestimmen
(X,Y:Integer;Ausgabe:TLabel;var
SListe:TStringList;Flaeche:TCanvas);

var T:TInhaltsgraph;
    Zaehl:Integer;
    Kno,K:TPfadknoten;
begin
  if not Leer  then
  begin
    Kno:=TPfadknoten.Create;
    K:=Kno;
    if
   FindezuKoordinatendenKnoten(X,Y,TInhaltsknoten(Kno))=false
    then
      Kno:=TPfadknoten(Anfangsknoten);
    Kno.ErzeugeweiteBaumpfade;
    if Kno.Pfadliste.Leer
    then
      ShowMessage('Keine Pfade')
    else
      Kno.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
  end;

  Pfadlistenloeschen;
  K.Free;
  K:=nil;
end;


{Menü: Abstand von zwei Knoten}

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
    Application.Processmessages;
    if Graph.Abbruch then goto Endproc;
    Ka.Pfadrichtung:=Ka.Zielknoten(Kn);
    MomentaneKantenliste.AmEndeanfuegen(Ka);
   Kno.Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
    if Ka.Zielknoten(Kn).Wert=Kno.Wert
    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
      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
      GehezuNachbarknoten(Ka.Zielknoten(Kn),Ka.Zielknoten(Kn).
       AusgehendeKantenliste.Kante(0));
    MomentaneKantenliste.AmEndeloeschen(Ob);
  Endproc:
  end;

begin
  MomentaneKantenliste:=TKantenliste.Create;
  Graph.Pfadlistenloeschen;
  Gefunden:=false;
  if Wert=Kno.Wert then
  begin
    P:=TGraph.Create;
    P.Knotenliste.AmEndeanfuegen(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))
    else
      GehezuNachbarknoten(self,AusgehendeKantenliste.Kante(1));
  if AusgehendeKantenliste.Anzahl=1
  then
    GehezuNachbarknoten(self,AusgehendeKantenliste.Kante(0));
  if not gefunden then Graph.Pfadlistenloeschen;
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
end;

function TPfadgraph.BestimmeminimalenPfad
(Kno1,Kno2:TKnoten;Wert:TWert;
Flaeche:TCanvas):TPfad;
label Endproc;
var WegPfadliste:TPfadliste;
     MomentanerWeg,MomentanerWegneu:TKantenliste;
     Index1,Index2:Integer;
     Ka,Ka1,Ka2:TKante;
     Kno:TKnoten;
     Ob:TObject;
begin
  BestimmeminimalenPfad:=TPfad.Create;
  WegPfadliste:=TPfadliste.Create;
  LoescheKnotenbesucht;
  Kno1.Besucht:=true;
  if not Kno1.AusgehendeKantenliste.Leer then
    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,Wert);
  while not Wegpfadliste.Leer  do
  begin
    Wegpfadliste.AmEndeloeschen(Ob);
    Momentanerweg:=TGraph(Ob).Kantenliste;
    if Demo then
      begin
        TInhaltsgraph(TKantenliste(MomentanerWeg).Graph).
         FaerbeGraph(clgreen,psdot);
        TInhaltsgraph(TKantenliste(MomentanerWeg).Graph).
         ZeichneGraph(Flaeche);
        Demopause;
```

```
      TInhaltsgraph(TKantenliste(MomentanerWeg).Graph).
       FaerbeGraph(clblack,pssolid);
      TInhaltsgraph(TKantenliste(MomentanerWeg).
      Graph).ZeichneGraph(Flaeche);
    end;
  Ka1:=MomentanerWeg.Kante(MomentanerWeg.Letztes);
  Kno:=Ka1.Pfadrichtung;
  if Kno=Kno2 then
  begin
    Bestimmeminimalenpfad:=TPfad(MomentanerWeg.Kopie.Graph);
    if Demo then
    begin
      TInhaltsgraph(TKantenliste(MomentanerWeg).
      Graph).FaerbeGraph(clred,psdot);
      TInhaltsgraph(TKantenliste(MomentanerWeg).
       Graph).ZeichneGraph(Flaeche);
      Demopause;
       TInhaltsgraph(TKantenliste(MomentanerWeg).Graph).FaerbeGraph(clblack,pssolid);
       TInhaltsgraph(TKantenliste(MomentanerWeg).Graph).ZeichneGraph(Flaeche);
    end;
    goto Endproc;
  end;
  if not Kno.Besucht
  then
  begin
    Kno.Besucht:=true;
    if not Kno.AusgehendeKantenliste.Leer then
      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,Wert);
end;
Endproc:
while not Wegpfadliste.Leer do
begin
```

```
    Wegpfadliste.AmEndeloeschen(Ob);
    TPfad(Ob).Free;
    Ob:=nil;
  end;
  Wegpfadliste.Free;
  Wegpfadliste:=nil;
end;



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
    Application.ProcessMessages;
    result:=true;
    if Kno1=Kno2
    then
    begin
      Showmessage('Die beiden Knoten sind identisch!');
      goto Endproc;
    end;
    SListe:=TStringList.Create;
    if GraphistBinaerbaum and (MessageDlg('Binaeres Suchen in
    einem geordnetem Binärbaum?',
      mtConfirmation, [mbYes, mbNo], 0) = mrYes)
    then
    begin
      Kno1.BinaeresSuchen(Kno2);
      if Kno2.Pfadliste.Leer
      then
      begin
       Ausgabe.Caption:='';
       Ausgabe.Refresh;
      ShowMessage('Knotenwert wurde nicht gefunden  oder der
       untersuchte '+chr(13)+
      '(Teil-)Graph ist kein geordneter Binärbaum!');
      end
      else
      Kno2.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
```

```
    end
    else
    begin
       if (MessageDlg('Algorithmus nach Dijkstra (sonst Algo-
        rithmus nach Ford: bei gerichtetem'+chr(13)+
      'Graphen auch mit negativer Kantenbewertung ohne negative
       Kreise)?',
      mtConfirmation, [mbYes, mbNo], 0) = mrYes)
      then
      T:=TInhaltsgraph(self.BestimmeMinimalenPfad(Kno1,Kno2,Bewertung,Flaeche))
      else
       T:=TInhaltsgraph(TInhaltsgraph(self).BestimmeMinimalenPfad(Kno1,Kno2,Bewertung));
      if Demo then Showmessage('Ausgabe des Ergebnisses');
      if not T.Kantenliste.Leer then
      begin
        T.FaerbeGraph(clred,psdot);
        T.ZeichneGraph(Flaeche);
        Ausgabe.Caption:=
        T.InhaltallerKnoten(ErzeugeKnotenstring)+' Summe: '+
        RundeZahltostring(T.Kantensumme(Bewertung),
        Kantengenauigkeit);
        SListe.Add(Ausgabe.Caption);
        Ausgabe.Refresh;
        if Abbruch then
        begin
          FaerbeGraph(clblack,pssolid);
          ZeichneGraph(Flaeche);
          Ausgabe.Caption:='';
          Ausgabe.Refresh;
          goto Endproc;
        end;
        Messagebeep(0);
        ShowMessage(Ausgabe.Caption);
        Ausgabe.Caption:='';
        Ausgabe.Refresh;
        if Demo then T.FaerbeGraph(clblack,pssolid);
        T.ZeichneGraph(Flaeche);
        result:=true;
      end
    else
     ShowMessage('Kein minimaler Pfad zwischen den Knoten');
  end
  end
  else
    result:=false;
  Endproc:
end;


{Menü Alle Pfade zwischen zwei Knoten}
```

```
procedure TPfadknoten.ErzeugeallePfadeZielknoten(Kno:TKnoten);
var Index:Integer;
    MomentaneKantenliste:TKantenliste;


  procedure GehezuallenNachbarknoten(Kn:TKnoten;Ka:TKante);
  label Endproc;
  var Ob:TObject;
      Index:Integer;
  begin
    Application.Processmessages;
    if TInhaltsgraph(Graph).Stop then goto Endproc;
    if not Ka.Zielknoten(Kn).Besucht then
    begin
      Ka.Pfadrichtung:=Ka.Zielknoten(Kn);
      MomentaneKantenliste.AmEndeanfuegen(Ka);
      if Ka.Zielknoten(Kn)=Kno then
      begin
    Ka.Zielknoten(Kn).Pfadliste.AmEndeanfuegen(MomentaneKantenliste.Kopie.Graph);
        if Kno.Pfadliste.Anzahl>10000 then
        begin
          ShowMessage('Mehr als 10000 Pfade!Abbruch!');
          Tinhaltsgraph(Graph).Stop:=true;
          goto Endproc;
        end;
      end;
      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;
    Endproc:
  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;

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
  then
  begin
    Application.ProcessMessages;
    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
    begin
      Ausgabe.Caption:='';
      Ausgabe.Refresh;
      ShowMessage('Keine Pfade zwischen den Knoten');
    end
    else
      Kno2.AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
    end
    else
      result:=false;
  Endproc:
  if Abbruch then ShowMessage('Abbruch!');
end;



{Menü: Minimales Gerüst des Graphen}

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;
    ListederKanten.Sortieren(Groesser,Bewertung);
    while not ListederKanten.Leer do
    begin
      Ka:=TInhaltskante(ListederKanten.Kante(0));
      if Demo then
      begin
        Ka.Farbe:=clgreen;
        Ka.Stil:=psdot;
        Ka.zeichneKante(Flaeche);
        Demopause;
        Ka.Farbe:=clblack;
        Ka.Stil:=pssolid;
        Ka.zeichneKante(Flaeche);
      end;
      Knoa:=TInhaltsknoten.Create;
      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);
      if T.GraphhatKreise
      then
        T.LoescheInhaltskante(Kb)
      else
      begin
        Ausgabe.Caption:=’GerüstKante: ‘+Ka.Wert;
        if not Ka.KanteistSchlinge then
        begin
          Ka.Farbe:=clred;
          Ka.Stil:=psdot;
```

```
          Ka.ZeichneKante(Flaeche);
          Demopause;
        end;
      end;
      if not ListederKanten.Leer then
      ListederKanten.AmAnfangLoeschen(Ob);
      if not ListederKanten.Leer then
      ListederKanten.Sortieren(Groesser,Bewertung);
    end;
    Ausgabe.Caption:='Kanten: '+
    T.InhaltallerKantenoderKnoten(ErzeugeKantenstring)+
    ' Summe: '+
   RundeZahltostring(T.Kantensumme(Bewertung),Kantengenauigkeit);
    SListe.Add(Ausgabe.Caption);
    Demopause;
    Ausgabe.Refresh;
    T.Freeall;
    T:=nil;
  end;
end;


{Ohne Menü: Tiefer Baum vereinfacht}

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(self.Graph).Demo,TPfadgraph(self.Graph).
        Pausenzeit,
       TPfadgraph(self.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(self.Graph).Demo,TPfadgraph(self.Graph).Pausenzeit,
        TPfadgraph(self.Graph).Kantengenauigkeit);
      MomentaneKantenliste.AmEndeloeschen(Ob);
    end;
    Endproc:
  end;


begin
  if Preorder then
  begin
  Knotenzeichnen(Flaeche,TPfadgraph(self.Graph).Demo,TPfadgraph(self.Graph).Pausenzeit);
    SListe.Add(' '+self.Wert);
  end;
  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;
  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;


{Ohne Menü: Erzeuge minimale Pfade}

procedure TPfadknoten.ErzeugeminimalePfade;
var Index,Index1:Integer;
    HilfKantenliste:TKantenliste;
    MomentaneKantenliste:TKantenliste;

  procedure GehezuallenNachbarn(Kno:TKnoten;Ka:TKante);
  label Endproc;
  var Ob:TObject;
      Index:Integer;
  begin
    Application.Processmessages;
    if Graph.Abbruch then goto Endproc;
    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;
    Endproc:
  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
   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;
        Pfadliste.AmEndeanfuegen(HilfKantenliste.Kopie.Graph);
        Hilfkantenliste.Free;
        Hilfkantenliste:=nil;
      end;
  end;
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
end;


{Ohne Menü: Knoten ist Kreisknoten und GraphhatKreise}

function TPfadknoten.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
    Application.Processmessages;
    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 AusgehendeKantenliste.Leer then
  for Index:=0 to AusgehendeKantenliste.Anzahl-1 do
  begin
   GehezudenNachbarknoten(self,AusgehendeKantenliste.Kante(Index));
    if Gefunden then goto Endproc;
  end;
  Endproc:
  MomentaneKantenliste.Free;
  MomentaneKantenliste:=nil;
  KnotenistKreisknoten:=Gefunden;
end;

function TPfadgraph.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
    Application.ProcessMessages;
    Gefunden:=Knotenliste.Knoten(Index).KnotenistKreisknoten;
    if Gefunden then goto Endproc;
  end;
  Endproc:
  GraphhatKreise:=Gefunden;
end;


{Ohne Menü: Alle Pfade und minimale Pfade}

procedure TPfadknoten.ErzeugeAllePfadeundMinimalenPfad
  (ZKno:TPfadKnoten;var Minlist:TKantenliste);
var Index:Integer;
```

```
    MomentaneKantenliste:TKantenliste;

  procedure GehezuallenNachbarknoten(Kno:TKnoten;Ka:TKante);
  label Endproc;
  var Ob:TObject;
      Index:Integer;
  begin
    Application.Processmessages;
    if Graph.Abbruch then goto Endproc;
    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:=MentaneKantenliste.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;
    Endproc:
  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;



function TPfadgraph.
```

```
AllePfadeundminimalenPfadzwischenzweiKnotenbestimmen
(X,Y:Integer;Ausgabe:TLabel;var
SListe:TStringList;Flaeche:TCanvas):Boolean;

var Zaehl:Integer;
    T:TInhaltsgraph;
    Kno1,Kno2:TPfadknoten;
    Gefunden:Boolean;
    Minlist:TKantenliste;
begin
  result:=false;
  if self.ZweiKnotenauswaehlen(X,Y,TInhaltsknoten(Kno1),
  TInhaltsknoten(Kno2),
  Gefunden)and Gefunden
  then
  begin
    Application.ProcessMessages;
    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
      Kno1.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;
```

```
end.

Unit UMath1:

unit UMath1;(nur für DWK)
{$F+}

interface

uses

  ULIst,UInhGrph,Ugraph,UKante,
  StdCtrls, Controls, ExtCtrls, Dialogs,
  Menus, Classes,
  SysUtils, WinTypes, WinProcs, Messages,Graphics,
  Forms,ClipBrd,Printers;

type

  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;override;
    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;override;
  property Ergebnis:string read WelchesErgebnis write
  SetzeErgebnis;
  function Wertlisteschreiben:TStringlist;override;
  procedure Wertlistelesen;override;
end;

TNetzgraph = class(TInhaltsgraph)
  constructor Create;override;
  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;


THamiltongraph = class(TInhaltsgraph)
  constructor Create;override;
  procedure Hamilton(Stufe:Integer;
  Kno,Zielknoten:TInhaltsknoten;var Kliste:TKantenliste;
  Flaeche:TCanvas);
  procedure Hamiltonkreise(Flaeche:TCanvas;Ausgabe:TLabel;var
  SListe:TStringlist);
end;


TEulergraph   = class(TInhaltsgraph)
  constructor Create;override;
  procedure Euler(Stufe:Integer;Kno,Zielknoten:TInhaltsknoten;var
  Kliste:TKantenliste;Flaeche:TCanvas;EineLoesung:Boolean;var
  Gefunden:Boolean;Ausgabe:TLabel);
  procedure EulerfixKant:TKante;Kno,Zielknoten:TInhaltsknoten;
  ;var KListe:TKantenliste;Flaeche:TCanvas;Ausgabe:TLabel);
  procedure Eulerlinie(Flaeche:TCanvas;Ausgabe:TLabel;
  var SListe:TStringlist;
  Anfangsknoten,Endknoten:TInhaltsknoten);
end;


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;override;
  property Knotenfarbe:Integer read WelcheFarbzahl
   write SetzeFarbzahl;
  property Ergebnis:string read WelchesErgebnis
   write SetzeErgebnis;
  function Wertlisteschreiben:TStringlist;override;
  procedure Wertlistelesen;override;
end;

TFarbgraph = class(TInhaltsgraph)
  constructor Create;override;
  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;

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;
  procedure Wertlistelesen;override;
  procedure SetzeAnfangsWertKnotenart;
  function BestimmeAnfangsknoten:TAutomatenknoten;
end;

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;override;
  property Ordnung:Integer read WelcheOrdnung
  write setzeOrdnung;
  property Ergebnis:string read WelchesErgebnis
   write SetzeErgebnis;
  function Wertlisteschreiben:TStringlist;override;
  procedure Wertlistelesen;override;
end;

TRelationsgraph = class(TInhaltsgraph)
  constructor Create;override;
  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;


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;override;
  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;override;
  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;override;
  property Distanz:Extended read WelcheDistanz
   write SetzeDistanz;
  function Wertlisteschreiben:TStringlist;override;
  procedure Wertlistelesen;override;
  procedure LoescheFluss;
  procedure SetzeKnotenDistanz;
  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;


TMatchknoten = class(TInhaltsknoten)
private
  Matchkante_:Integer;
```

```
    VorigeKante_:Integer;
    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;override;
    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;

  TMatchgraph = class(TInhaltsgraph)
    constructor Create;override;
    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;



implementation

constructor TNetzknoten.Create;
begin
  inherited Create;
  Anfang_:=0;
  Ende_:=0;
  Puffer_:=0;
  Ergebnis_:='';
  Wertlisteschreiben;
end;
procedure TNetzknoten.SetzeAnfangszeit(Z:Extended);
```

```
begin
  Anfang_:=Z;
end;

function TNetzknoten.WelcheAnfangszeit:Extended;
begin
  WelcheAnfangszeit:=Anfang_;
end;


procedure TNetzknoten.SetzeEndzeit(Z:Extended);
begin
  Ende_:=Z;
end;

function TNetzknoten.WelcheEndzeit:Extended;
begin
  WelcheEndzeit:=Ende_;
end;

procedure TNetzknoten.SetzePuffer(Z:Extended);
begin
  Puffer_:=Z;
end;


function TNetzknoten.WelcherPuffer:Extended;
begin
  WelcherPuffer:=Puffer_;
end;

function TNetzknoten.WelchesErgebnis:string;
begin
  WelchesErgebnis:=Ergebnis_;
end;

procedure TNetzknoten.SetzeErgebnis(S:string);
begin
  Ergebnis_:=S;
end;


function TNetzKnoten.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;
  Wertliste.Add(Realtostring(Anfang));
  Wertliste.Add(Realtostring(Ende));
  Wertliste.Add(Realtostring(Puffer));
  Wertliste.Add(Ergebnis);
  Wertlisteschreiben:=Wertliste;
```

```
end;

procedure TNetzKnoten.Wertlistelesen;
begin
  inherited Wertlistelesen;
  Anfang:=StringtoReal(Wertliste.Strings[Wertliste.Count-4]);
  Ende:=StringtoReal(Wertliste.Strings[Wertliste.Count-3]);
  Puffer:=StringtoReal(Wertliste.Strings[Wertliste.Count-2]);
  Ergebnis:=Wertliste.Strings[Wertliste.Count-1];
end;

constructor TNetzkante.Create;
begin
  inherited Create;
  Ergebnis_:='';
  Wertlisteschreiben;
end;



procedure TNetzkante.SetzeErgebnis(S:string);
begin
  Ergebnis_:=S;
end;

function TNetzkante.WelchesErgebnis:string;
begin
  WelchesErgebnis:=Ergebnis_;
end;

function TNetzkante.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;
  Wertliste.Add(Ergebnis);
  Wertlisteschreiben:=Wertliste;
end;

procedure TNetzkante.Wertlistelesen;
begin
  inherited Wertlistelesen;
  Ergebnis:=Wertliste.Strings[Wertliste.Count-1];
end;

constructor TNetzgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TNetzknoten;
  InhaltsKanteclass:=TNetzkante;
end;
procedure TNetzgraph.BestimmeAnfangszeit
```

```
(KnoStart:TNetzknoten;Flaeche:TCanvas);
label Endproc;
var Index:Integer;
    Speicherliste:TKnotenliste;
    Kno:TNetzknoten;
    Berechnet:Boolean;
    StartknotenZeit,NeueZeit:Extended;
begin
  LoescheKnotenbesucht;
  Speicherliste:=TKnotenliste.Create;
  KnoStart.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
      if Abbruch then goto Endproc;
      Speicherliste.AmEndeloeschen(TObject(Kno));
      if not Kno.Besucht
      then
      begin
        Berechnet:=true;
        if not Kno.EingehendeKantenliste.Leer then
          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
            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;
          Kno.Anfang:=Startknotenzeit;
          if Demo
          then
            begin
              Knotenwertposition:=1;
              Kno.Farbe:=clblue;
            ZeichneGraph(Flaeche);
```

```
              Demopause;
             Kno.Farbe:=clblack;
            Knotenwertposition:=0;
            end;
        end
        else
       Speicherliste.AmAnfanganfuegen(Kno);
      end;
    end;
  Endproc:
  LoescheKnotenbesucht;
  Speicherliste.Free;
  Speicherliste:=nil;
end;

procedure TNetzgraph.BestimmeEndzeit
(KnoZiel:TNetzknoten;Flaeche:TCanvas);
label Endproc;
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
    if Abbruch then goto Endproc;
    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:=
            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;
        if Demo then
        begin
          Knotenwertposition:=2;
          Kno.Farbe:=clblue;
          ZeichneGraph(Flaeche);
          Demopause;
          Kno.Farbe:=clblack;
          Knotenwertposition:=0;
        end;
      end
      else
        Speicherliste.AmEndeanfuegen(Kno);
    end;
  end;
  Endproc:
  LoescheKnotenbesucht;
  Speicherliste.Free;
  Speicherliste:=nil;
end;

procedure TNetzgraph.BestimmeErgebnisse(var SListe:TStringlist);
label Endproc;
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
    begin
      if Abbruch then goto Endproc;
      Kno:=TNetzknoten(Knotenliste.Knoten(Index));
      Kno.Puffer:=Kno.Ende-Kno.Anfang;
       S:=Kno.Wert+': '
        +' Anfang: ' +
      RundeStringtoString(Kno.Wertliste[1],Knotengenauigkeit)
        +' Ende:
      '+Rundestringtostring(Kno.Wertliste[2],Knotengenauigkeit)+
        ' Puffer: '+
```

```
      RundeStringtoString(Kno.Wertliste[3],Knotengenauigkeit);
    if Kno.Puffer=0 then
    begin
      Kno.Farbe:=clred;
      Kno.Stil:=psdot;
      S:=S+' kritisch';
    end;
    SListe.Add(S);
    with Kno do
    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
  begin
    if Abbruch then goto Endproc;
    Ka:=TNetzkante(Kantenliste.Kante(Index));
    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
     begin
       S:=S+' kritisch ';
       Ka.Farbe:=clred;
       Ka.Stil:=psdot;
     end;
     SListe.Add(S);
     with Ka do
     begin
       Q:= Wert
         +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;


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
  begin
    ShowMessage('Der Graph enthält ungerichtete Kanten oder
    Kreise.');
    exit;
  end;
  if not Leer then
   if (not GraphhatKreise)and (AnzahlungerichteteKanten=0)
    then
    begin
      Zaehl:=0;
      for Index:=0 to Knotenliste.Anzahl-1 do
        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);
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 numeri
  schen Bereich!’);
  Eingabe:=false;
end;
if Eingabe=false then goto Endproc;
Zaehl:=0;
for Index:=0 to Knotenliste.Anzahl-1 do
  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);
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 numeri-
schen 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...';
        if Demo then Ausgabe.Caption:='Bestimme Anfangszeit...';
        BestimmeAnfangszeit(Kno1,Flaeche);
         if Abbruch then goto Endproc;
        LoescheBild(G,TForm(Oberflaeche));
        if Demo then Ausgabe.Caption:='Bestimme Endzeit...';
        BestimmeEndzeit(Kno2,Flaeche);
         if Abbruch then goto Endproc;
        LoescheBild(G,TForm(Oberflaeche));
        BestimmeErgebnisse(SListe);
         if Abbruch then goto Endproc;
        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;

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
  Application.Processmessages;
  if Abbruch then exit;
  if not Kno.AusgehendeKantenliste.leer then
    for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
    begin
      if Abbruch then exit;
     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;
        if Abbruch then exit;
        Zkno.Besucht:=true;
        Kliste.AmEndeanfuegen(Ka);
        Stufe:=Stufe+1;
        Application.Processmessages;
        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;
procedure THamiltongraph.Hamiltonkreise
```

```
(Flaeche:TCanvas;Ausgabe:TLabel;
var SListe: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);
    Ausgabe.Refresh;
    T.FaerbeGraph(clred,psdot);
    T.ZeichneGraph(Flaeche);
    ShowMessage(Ausgabe.Caption);
    T.FaerbeGraph(clred,psdot);
    T.ZeichneGraph(Flaeche);
    MomentaneKantenliste.Free;
    MomentaneKantenliste:=nil;
    if Abbruch then exit
  end;
  Ausgabe.Caption:=’’;
  Ausgabe.Refresh;
end;

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;Ausgabe:TLabel);
var Index:Integer;
    Zkno:TInhaltsknoten;
    Ka:TInhaltskante;
begin
  Application.Processmessages;
  if EineLoesung and Gefunden then exit;
  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;
        if Abbruch then exit;
        Kliste.AmEndeanfuegen(Ka);
        if Demo then
      Ausgabe.Caption:=Kliste.Kantenlistealsstring;
       Stufe:=Stufe+1;
       if (Zkno=Zielknoten)and
      (Stufe=AnzahlKantenmitSchlingen+1)  then
        begin
         Zielknoten.Pfadliste.AmEndeanfuegen(Kliste.Kopie.Graph);
          Gefunden:=true;
          if EineLoesung then exit;
        end
        else
     Euler(Stufe,Zkno,Zielknoten,Kliste,Flaeche,EineLoesung,Gefunden,Ausgabe);
        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);
    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
  EineLoesung:=true;
  ZeichneGraph(Flaeche);
  if Leer then exit;
  MomentaneKantenliste:=TKantenliste.Create;
  LoescheKantenbesucht;
  Pfadlistenloeschen;
  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,Ausgabe);
  end
  else
    Euler(1,Anfangsknoten,Endknoten,MomentaneKantenliste,Flaeche,EineLoesung,Gefunden,Ausgabe);
 Endknoten.AnzeigePfadliste(Flaeche,Ausgabe,
 SListe,true,true);
 if Abbruch then exit;
 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 TEulergraph.Eulerfix(Kant:TInhaltskante;
Kno,Zielknoten:
  TInhaltsknoten;var Kliste: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
    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);
   end;
   if Kant<>nil then
   begin if Demo then
     begin
      Kant.Farbe:=clblue;Kant.Stil:=psdot;
      Kant.ZeichneKante(Flaeche);Demopause;end;Kliste.
       AmAnfanganfuegen(Kant);
     end;
    if Demo then Ausgabe.Caption:=KListe.Kantenlistealsstring;
   end
end;
constructor TFarbknoten.Create;
```

```
begin
  inherited Create;
  Knotenfarbe_:=0;
  Ergebnis_:='';
  Wertlisteschreiben;
end;

procedure TFarbknoten.SetzeFarbzahl(Fa:Integer);
begin
  Knotenfarbe_:=Fa;
end;

function TFarbknoten.WelcheFarbzahl:Integer;
begin
  WelcheFarbzahl:=Knotenfarbe_;
end;

procedure TFarbknoten.SetzeErgebnis(S:string);
begin
  Ergebnis_:=S;
end;

function TFarbknoten.WelchesErgebnis:string;
begin
  WelchesErgebnis:=Ergebnis_;
end;

function TFarbKnoten.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;
  Wertliste.Add(Integertostring(Knotenfarbe));
  Wertliste.Add(Ergebnis);
  Wertlisteschreiben:=Wertliste;
end;

procedure TFarbKnoten.Wertlistelesen;
begin
  inherited Wertlistelesen;
  Knotenfarbe:=StringtoInteger(Wertliste.Strings[Wertliste.Count-
   2]);
  Ergebnis:=Wertliste.Strings[Wertliste.Count-1];
end;

constructor TFarbgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TFarbknoten;
  Inhaltskanteclass:=TInhaltskante;
end;
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;


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
    then
      NachbarknotenhabengleicheFarbe:=false
    else
    NachbarknotenhabengleicheFarbe:=(Kno1.Knotenfarbe=Kno2.Knotenfarbe);
  end;


begin
  Kno:=TFarbknoten(Knotenliste.Knoten(Index));
  repeat
    Kno.Knotenfarbe:=(Kno.Knotenfarbe+1) mod (AnzahlFarben+1);
    if Kno.Knotenfarbe=0
    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;
        if not Kno.EingehendeKantenliste.leer  then
        for Zaehl:=0 to Kno.EingehendeKantenliste.Anzahl-1 do
        if NachbarknotenhabengleicheFarbe
        (Kno.EingehendeKantenliste.Kante(Zaehl)) then
         GleicheFarbe:=true;
  until (Not GleicheFarbe) ;
  KnotenistzuFaerben:=true;
```

```
end;

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
    Application.Processmessages;
    if Abbruch then  exit;
    Knotenzufaerben:=Knotenistzufaerben(Index,AnzahlFarben);
    if (Knotenzufaerben) and (Index<AnzahlKnoten-1)
    then
    Farbverteilung(Index+1,AnzahlFarben,Gefunden,EineLoesung,Flaeche,
        Ausgabe,Ausgabeliste)
    else
      if (Index=Anzahlknoten-1) and Knotenzufaerben then
      begin
        Gefunden:=true;
        ErzeugeErgebnis;
        S:='';
        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;
        Ausgabeliste.Add(S);
        Ausgabe.Caption:=S;
      end
  until (not Knotenzufaerben) or (Gefunden and EineLoesung);
  Endproc:
end;

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));
  end;
end;

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 Far-
ben:','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;



constructor TAutomatenknoten.Create;
begin
  inherited Create;
  Knotenart_:=0;
  Wertlisteschreiben;
end;

procedure TAutomatenknoten.SetzeKnotenart(Ka:TKnotenart);
begin
  Knotenart_:=Ka;
end;

function TAutomatenknoten.WelcheKnotenart:TKnotenart;
begin
  WelcheKnotenart:=Knotenart_;
end;

function TAutomatenknoten.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;
  Wertliste.Add(IntegertoString(Knotenart));
  Wertlisteschreiben:=Wertliste;
end;

procedure TAutomatenknoten.Wertlistelesen;
begin
  inherited Wertlistelesen;
  Knotenart:=StringtoInteger(Wertliste.Strings[Wertliste.Count-
1]);
end;

constructor TAutomatengraph.Create;
begin
  inherited Create;
  MomentanerKnoten_:=nil;
end;

procedure TAutomatengraph.SetzeMomentanenKnoten
(Kno:TAutomatenknoten);
begin
  MomentanerKnoten_:=Kno;
end;


function TAutomatengraph.
```

```
WelcherMomentaneKnoten:TAutomatenknoten;
begin
  WelchermomentaneKnoten:=MomentanerKnoten_;
end;

function TAutomatengraph.Wertlisteschreiben:TStringList;
begin
  inherited Wertlisteschreiben;
 Wertliste.Add(Integertostring(Knotenliste.Position(MomentanerKnoten)));
  Wertlisteschreiben:=Wertliste;
end;

procedure TAutomatengraph.Wertlistelesen;
begin
  inherited Wertlistelesen;
  if not Knotenliste.leer then
    Momentanerknoten:=
    TAutomatenknoten(Knotenliste.Knoten(StringtoInteger(Wertliste.Strings
      [Wertliste.Count-1])));
end;



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;

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;

constructor TRelationsknoten.Create;
begin
  inherited Create;
```

```
  Ordnung_:=0;
  Ergebnis_:=’’;
  Wertlisteschreiben;
end;


procedure TRelationsknoten.SetzeOrdnung(O:Integer);
begin
  Ordnung_:=O;
end;

function TRelationsknoten.WelcheOrdnung:Integer;
begin
  WelcheOrdnung:=Ordnung_;
end;

procedure TRelationsknoten.SetzeErgebnis(S:string);
begin
  Ergebnis_:=S;
end;

function TRelationsknoten.WelchesErgebnis:string;
begin
   WelchesErgebnis:=Ergebnis_;
end;

function TRelationsKnoten.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;
  Wertliste.Add(Integertostring(Ordnung));
  Wertliste.Add(Ergebnis);
  Wertlisteschreiben:=Wertliste;
end;

procedure TRelationsKnoten.Wertlistelesen;
begin
  inherited Wertlistelesen;
  Ordnung:=StringtoInteger(Wertliste.Strings[Wertliste.Count-
   2]);
  Ergebnis:=Wertliste.Strings[Wertliste.Count-1];
end;

constructor TRelationsgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TRelationsknoten;
  InhaltsKanteclass:=TInhaltskante;
end;

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;


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
      begin
        Ka:=TInhaltskante.Create;
        Ka.Weite:=30;
        Ka.Wert:='r';
        Ka.Farbe:=clgreen;
        Ka.Stil:=psdot;
        FuegeKanteein(Kno,Kno,true,Ka);
        Graphistreflexiv:=false;
      end;
    end;
  if Graphistreflexiv
  then
    SListe.Add('Die ursprüngliche Relation ist reflexiv!')
  else
    SListe.Add('Die ursprüngliche Relation ist nicht refle
     xiv!');
end;

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
     Application.processmessages;
```

```
    if Abbruch then exit;
   K:=TRelationsknoten(Ka.Zielknoten(Kno));
   TInhaltsknoten(K).Farbe:=clblack;
   TInhaltsknoten(K).Zeichneknoten(Flaeche);
   if not K.Besucht then
    begin
      if Demo then
      begin
        if Abbruch then exit;
        TInhaltskante(Ka).Farbe:=clblue;
        TInhaltskante(Ka).Stil:=psdot;
        TInhaltskante(Ka).ZeichneKante(Flaeche);
        TInhaltsknoten(K).Farbe:=clblack;
      end;
      Topsort(K);
    end
    else
      if K.Ordnung=0 then  Kreis:=true;
    if Demo then
    begin
      TInhaltskante(Ka).Farbe:=clblue;
      TInhaltskante(Ka).Stil:=psdot;
      TInhaltskante(Ka).ZeichneKante(Flaeche);
      Application.Processmessages;
      if Abbruch then exit;
    end;
  end;



begin
  if not Kreis and (not Kno.Besucht) then
  begin
    Kno.Besucht:=true;
    if not Kno.AusgehendeKantenliste.Leer then
      for  Zaehl:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
      begin
        if not
     Kno.AusgehendeKantenliste.Kante(Zaehl).KanteistSchlinge
        then
          TopsortNachbarknoten(Kno,Kno.ausgehendeKantenliste.Kante(Zaehl));
       if Abbruch then exit;
      end;
    AktuelleOrdnung:=AktuelleOrdnung+1;
    Kno.Ordnung:=AnzahlKnoten-AktuelleOrdnung+1;
    if Demo then
    begin
      Kno.Position:=1;
      Kno.Farbe:=clred;
     Kno.ZeichneKnoten(Flaeche);
```

```
          Demopause;
        Kno.Position:=0;
        Kno.Farbe:=clblack;
        Kno.ZeichneKnoten(Flaeche);
        Kno.Position:=0;
        if Abbruch then exit;
      end;
    end;
  end;


begin
  if not Leer then
  begin
    LoescheKnotenbesucht;
    SetzebeiallenKnotenAnfangsOrdnung;
    AktuelleOrdnung:=0;
    Kreis:=false;
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      if Abbruch then exit;
      Topsort(TRelationsknoten(Knotenliste.Knoten(Index)));
    end;
  end;
end;


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
      begin
        Ka:=Tinhaltskante.Create;
        Ka.Wert:='t';
        Ka.Farbe:=clred;
        Ka.Stil:=psdot;
      FuegeKanteein(TInhaltsknoten(Kno1),TInhaltsknoten(Kno2),true,Ka);
        Graphisttransitiv:=false;
      end;
    end;

begin
  Graphisttransitiv:=true;
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
```

```
    begin
      Kno:=Knotenliste.Knoten(Index);
      if not Kno.EingehendeKantenliste.leer then
        for Index1:=0 to Kno.EingehendeKantenliste.Anzahl-1 do
        begin
       Kno1:=Kno.EingehendeKantenliste.Kante(index1).Anfangsknoten;
          if not Kno.AusgehendeKantenliste.leer then
            for Index2:=0 to Kno.AusgehendeKantenliste.Anzahl-1
            do
            begin
          Kno2:=Kno.AusgehendeKantenliste.Kante(Index2).Endknoten;
              ErzeugeTransitiveKante(Kno1,Kno2);
            end;
        end;
    end;
  if not Graphisttransitiv
  then
    SListe.Add('Die ursprüngliche Relation ist nicht transi
     tiv!')
  else
    SListe.Add('Die ursprüngliche Relation ist transitiv!');
end;

procedure TRelationsgraph.ErzeugeErgebnis(var
SListe:TStringlist);
var Index:Integer;
    Kno:TRelationsknoten;
begin
  SListe.Add('Ordnung:');
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TRelationsknoten(Knotenliste.Knoten(Index));
      Kno.Ergebnis:=Kno.Wert+':'+Integertostring(Kno.Ordnung);
      SListe.Add(Kno.Ergebnis);
    end;
end;

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
    begin
      Ka:=TInhaltskante(Kantenliste.Kante(Index));
      Knoa:=TRelationsknoten(Ka.Anfangsknoten);
```

```
      Knoe:=TRelationsknoten(Ka.Endknoten);
      if not KanteverbindetKnotenvonnach(Knoe,Knoa)
      then
        Symmetrisch:=false;
    end;
  Relationistsymmetrisch:=Symmetrisch;
end;


constructor TMaxflussknoten.Create;
begin
  inherited Create;
  Distanz_:=0;
  Ergebnis_:='';
  Wertlisteschreiben;
end;

procedure TMaxflussknoten.SetzeDistanz(Di:Extended);
begin
  Distanz_:=Di;
end;

function TMaxflussknoten.WelcheDistanz:Extended;
begin
  WelcheDistanz:=Distanz_;
end;

procedure TMaxflussknoten.SetzeErgebnis(S:string);
begin
  Ergebnis_:=S;
end;

function TMaxflussknoten.WelchesErgebnis:string;
begin
  WelchesErgebnis:=Ergebnis_;
end;

function TMaxflussknoten.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;;
  Wertliste.Add(Realtostring(Distanz));
  Wertliste.Add(Ergebnis);
  Wertlisteschreiben:=Wertliste;
end;

procedure TMaxflussknoten.Wertlistelesen;
begin
  inherited Wertlistelesen;
  Distanz:=StringtoReal(Wertliste.Strings[Wertliste.Count-2]);
  Ergebnis:=Wertliste.Strings[Wertliste.Count-1];
```

```
end;

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;

constructor TMaxflusskante.Create;
begin
  inherited Create;
  Fluss_:=0;
  Ergebnis_:='';
  Wertlisteschreiben;
end;

procedure TMaxflusskante.SetzeFluss(Fl:Extended);
begin
  Fluss_:=Fl;
end;
function TMaxflusskante.WelcherFluss:Extended;
begin
  WelcherFluss:=Fluss_;
end;

procedure TMaxflusskante.SetzeErgebnis(S:string);
begin
  Ergebnis_:=S;
end;

function TMaxflusskante.WelchesErgebnis:string;
begin
  WelchesErgebnis:=Ergebnis_;
end;


function TMaxflusskante.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;
  Wertliste.Add(Realtostring(Fluss));
  Wertliste.Add(Ergebnis);
  Wertlisteschreiben:=Wertliste;
```

```
end;

procedure TMaxflusskante.Wertlistelesen;
begin
  inherited Wertlistelesen;
  Fluss:=StringtoReal(Wertliste.Strings[Wertliste.Count-2]);
  Ergebnis:=Wertliste.Strings[Wertliste.Count-1];
end;

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;

constructor TMaxflussgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TMaxflussknoten;
  InhaltsKanteclass:=TMaxflusskante;
  Distanz_:=0;
end;


procedure TMaxflussgraph.SetzeDistanz(Di:Extended);
begin
  Distanz_:=Di;
end;

function TMaxflussgraph.WelcheDistanz:Extended;
begin
  WelcheDistanz:=Distanz_;
end;

function TMaxflussgraph.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;
  Wertliste.Add(Realtostring(Distanz));
  Wertlisteschreiben:=Wertliste;
end;


procedure TMaxflussgraph.Wertlistelesen;
begin
  inherited Wertlistelesen;
```

```
  Distanz:=StringtoReal(Wertliste.Strings[Wertliste.Count-1]);
end;

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;

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;

procedure TMaxflussgraph.Fluss(Kno,Endknoten:TKnoten;var
Gefunden:Boolean;
  Var Gesamtfluss:Extended;Flaeche:TCanvas;Oberflaeche:TForm);
var Index1,Index2:Integer;
    Kaeingehend,Kaausgehend:TKante;
     G:TInhaltsgraphH;

  procedure Graphzeichnen;
  begin
    Knotenwertposition:=2;
    Kantenwertposition:=2;
   LoescheBikld(G,Oberflaeche);
    ZeichneGraph(Flaeche);
    Knotenwertposition:=0;
    Kantenwertposition:=0;
  end;



  GraphH:=TInhaltsgraph(MaxFlussgraph);
    Paintbox.OnMouseDown:=PanelDownMouse;
    Paintbox.OnDblClick:=nil;
    Paintbox.ONmousemove:=nil;
     MaxFlussgraph.StartFluss(Paintbox.Canvas,Gesamtfluss,Knotenformular);
    if Gesamtfluss>0 then
     begin
      Sliste:=TStringList.Create;
      MaxFlussgraph.BestimmeErgebnis(Sliste);
```

```
     Sliste.Insert(0,'Kanten,Schranken und Fluss:');
      Sliste.Add('maximaler Gesamtfluss: '+
      RundeZahltoString(Gesamtfluss,Graph.Kantengenauigkeit));
      if not Graph.Abbruch then
       begin
        StringlistnachListbox(Sliste,Listbox);
        ShowMessage('Der maximale Gesamtfluss beträgt '+
        RundeZahltoString(Gesamtfluss,Graph.Kantengenauigkeit));
        end;
       Bildloeschen;
      MaxFlussgraph.Knotenwertposition:=0;
      MaxFlussgraph.Kantenwertposition:=2;
      MaxFlussgraph.ZeichneGraph(Paintbox.Canvas);
      GraphH:=TInhaltsgraph(MaxFlussgraph);
      GraphH.Knotenwertposition:=0;
      GraphH.Kantenwertposition:=2;
       Aktiv:=false;
      GraphH:=MaxFlussgraph.
      InhaltskopiedesGraphen(TInhaltsgraph,TMaxflussknoten,
      TMaxflusskante,false);
      GraphH.Kantenwertposition:=2;
       Aktiv:=false;
       Sliste.Free;
       Sliste:=nil;
      MaxFlussgraph.Freeall;
      MaxFlussgraph:=nil;
    end;
    Menuenabled(true);
    if GraphH.Abbruch then
    begin
      Aktiv:=true;
      Bildloeschen;
      Graph.ZeichneGraph(Paintbox.Canvas);
      ShowMessage('Abbruch!');
    end;
  except
    AbbruchClick(Sender);
    Fehler;
  end;
end;


procedure BesucheeingehendeKante(Ka:TKante);
  var Kno1,Kno2:TMaxflussknoten;
      Schranke,Fluss:Extended;
      Kante:TMaxFlusskante;
  begin
    Application.Processmessages;
    if Abbruch then exit;
    if Gefunden then exit;
```

```
Kno1:=TMaxflussknoten(Ka.Endknoten);
Kno2:=TMaxflussknoten(Ka.Anfangsknoten);
Kante:=TMaxflusskante(Ka);
Schranke:=StringtoReal(Ka.Wert);
Fluss:=Kante.Fluss;
if not Kante.Besucht then
begin
  Kante.ErzeugeErgebnis;
  if Demo then
  begin
    Kante.Farbe:=clred;
    Kante.Stil:=psdot;
    Kante.Position:=2;
    Flaeche.Refresh;
    Graphzeichnen;
    Demopause;
    Kante.Position:=0;
  end;
  if Fluss>0 then
  begin
    Kante.Besucht:=true;
    Kno2.Distanz:=Minimum(Kno1.Distanz,Fluss);
    Kno2.ErzeugeErgebnis;
    if Demo then
    begin
      Kno2.Farbe:=clred;
      Kno2.Stil:=psdot;
      Kno2.Position:=2;
      Flaeche.Refresh;
      Graphzeichnen;
      Kno2.Position:=0;
      Demopause;
    end;
    self.Fluss(Kno2,Endknoten,Gefunden,Gesamtfluss,Flaeche);
  end;
  Kante.Besucht:=false;
  if Gefunden then Kante.Fluss:=Kante.Fluss-Distanz;
  Kno2.ErzeugeErgebnis;
  Kante.ErzeugeErgebnis;
  if Demo then
  begin
    Kante.Farbe:=clblack;
    Kante.Stil:=pssolid;
    Kno2.Farbe:=clblack;
    Kno2.Stil:=pssolid;
    Kante.Position:=2;
    Kno2.Position:=2;
    Flaeche.Refresh;
    Graphzeichnen;
    Kante.Position:=0;
```

```
      Kno2.Position:=0;
       Demopause;
    end;
  end;
end;

procedure BesucheausgehendeKante(Ka:TKante);
var Kno1,Kno2:TMaxflussknoten;
    Schranke,Fluss:Extended;
    Kante:TMaxFlusskante;
begin
  Application.Processmessages;
  if Abbruch then exit;
  if Gefunden then exit;
  Kno1:=TMaxflussknoten(Ka.Anfangsknoten);
  Kno2:=TMaxflussknoten(Ka.Endknoten);
  Kante:=TMaxflusskante(Ka);
  Schranke:=StringtoReal(Kante.Wert);
  Fluss:=Kante.Fluss;
  if not Kante.Besucht then
  begin
    Kante.ErzeugeErgebnis;
    if Demo then
    begin
      Kante.Farbe:=clred;
      Kante.Stil:=psdot;
      Kante.Position:=2;
      Flaeche.Refresh;
      Graphzeichnen;
      Demopause;
      Kante.Position:=0;
    end;
    if Fluss<Schranke then
    begin
      Kante.Besucht:=true;
      Kno2.Distanz:=Minimum(Kno1.Distanz,Schranke-Fluss);
      Kno2.ErzeugeErgebnis;
      if Demo then
      begin
        Kno2.Farbe:=clred;
        Kno2.Stil:=psdot;
        Kno2.Position:=2;
        Flaeche.Refresh;
        Graphzeichnen;
        Demopause;
        Kno2.Position:=0;
      end;
      if Kno2=Endknoten then
      begin
        Gefunden:=true;
```

```
          Distanz:=Kno2.Distanz;
        end
        else
       self.Fluss(Kno2,Endknoten,Gefunden,Gesamtfluss,Flaeche);
      end;
      Kante.Besucht:=false;
      if Gefunden then Kante.Fluss:=Kante.Fluss+Self.Distanz;
      Kante.ErzeugeErgebnis;
      Kno2.ErzeugeErgebnis;
      if Demo then
      begin
        Kante.Farbe:=clblack;
        Kante.Stil:=pssolid;
        Kno2.Farbe:=clblack;
        Kno2.Stil:=pssolid;
        Kante.Position:=2;
        Kno2.Position:=2;
        Flaeche.Refresh;
        Graphzeichnen;
        Demopause;
        Kante.Position:=0;
        Kno2.Position:=0;
      end;
    end;
  end;

begin
  if not Gefunden then
  begin
    if not Kno.AusgehendeKantenliste.Leer then
      for Index1:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
      begin
        if Abbruch then exit;
        Kaausgehend:=Kno.AusgehendeKantenliste.Kante(Index1);
        if Kaausgehend.gerichtet then
       BesucheausgehendeKante(Kaausgehend);
      end;
      if not Kno.EingehendeKantenliste.Leer then
        for Index2:=0 to Kno.EingehendeKantenliste.Anzahl-1 do
        begin
          if Abbruch then exit;
          Kaeingehend:=Kno.EingehendeKantenliste.Kante(Index2);
          if Kaeingehend.Gerichtet then
BesucheeingehendeKante(Kaeingehend);
        end;
  end;
end;
procedure TMaxflussGraph.StartFluss(Flaeche:TCanvas;var
Gesamtfluss:Extended,Oberflaeche:TForm);
```

```
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;
  LoescheFluss;
  Ergebniserzeugen;
  Zaehle:=0;
  for Index:=0 to Knotenliste.Anzahl-1 do
    if Knotenliste.Knoten(Index).EingehendeKantenliste.Leer then
    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
    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
    SetzeKnotenDistanz;
    LoescheKantenbesucht;
    Gefunden:=false;
    Distanz:=0;
    ZeichneGraph(Flaeche);
    Fluss(Startknoten,Zielknoten,Gefunden,Gesamtfluss,Flaeche)
  until not Gefunden;
  TInhaltsknoten(Zielknoten).Farbe:=clgreen;
  TInhaltsknoten(Zielknoten).Stil:=psdash;
  ZeichneGraph(Flaeche);
end;


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;

constructor TMatchknoten.Create;
begin
   inherited Create;
   Matchkante_:=-1;
   VorigeKante_:=-1;
   Wertlisteschreiben;
end;

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;

procedure TMatchknoten.SetzeMindex(M:Integer);
```

```
begin
   Matchkante_:=M;
end;

function TMatchknoten.WelcherMindex:Integer;
begin
  WelcherMindex:=Matchkante_;
end;

procedure TMatchknoten.SetzeVindex(V:Integer);
begin
   VorigeKante_:=V;
end;

function TMatchknoten.WelcherVindex:Integer;
begin
   WelcherVindex:=VorigeKante_;
end;

function TMatchKnoten.Wertlisteschreiben:TStringList;
begin
  inherited Wertlisteschreiben;
  Wertliste.Add(Integertostring(Matchkantenindex));
  Wertliste.Add(Integertostring(VorigerKantenindex));
  Wertlisteschreiben:=Wertliste;
end;

procedure TMatchKnoten.Wertlistelesen;
begin
  inherited Wertlistelesen;
 Matchkantenindex:=StringtoInteger(Wertliste.Strings[Wertliste.Count-
  2]);
 VorigerKantenindex:=StringtoInteger(Wertliste.Strings[Wertliste.Count-
  1]);
end;

function TMatchknoten.Knotenistexponiert:Boolean;
begin
  Knotenistexponiert:=(MatchKante=nil);
end;


constructor TMatchgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TMatchknoten;
  InhaltsKanteclass:=TInhaltskante;
end;

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;

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;


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
        if Abbruch then exit;
      Ka:=TInhaltskante(Kno1.AusgehendeKantenliste.Kante(Index2));
        Kno2:=TMatchknoten(Ka.Zielknoten(Kno1));
        if (Kno2.Matchkante=nil) and (Kno1.Matchkante=nil) and
         (not Ka.KanteistSchlinge) then
        begin
          Kno1.Matchkante:=Ka;
          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
          if abbruch then exit;
       Ka:=TInhaltskante(Kno1.EingehendeKantenliste.Kante(index2));
          Kno2:=TMatchknoten(Ka.Zielknoten(Kno1));
        if (Kno2.Matchkante=nil) and (Kno1.Matchkante=nil) and
```

```
          (not Ka.KanteistSchlinge) then
           begin
             Kno1.Matchkante:=Ka;
             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;



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;


procedure TMatchgraph.VergroessereMatching(G:TInhaltsgraph);
var Index:Integer;

  procedure FaerbeKanteum(Ka:TInhaltskante);
  begin
    if Ka.Wert='L    '
    then
    begin
       Ka.Wert:=TInhaltskante(G.Kantenliste.Kante(Kantenliste.Position(Ka))).Wert;
      Ka.Farbe:=clblack;
      Ka.Stil:=pssolid;
    end;
    if Ka.Wert='A    '
    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;


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
    Application.ProcessMessages;
    if Abbruch then goto NaechsteKante;
    if not ErweiternderWeggefunden
    then
    begin
      Y:=TMatchknoten(Ka.Zielknoten(X));
      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
        begin
        if (Y.Matchkante<> nil) then
          begin
          Ka.Wert:='A    ';
          Ka.Farbe:=clblue;
          Ka.Stil:=psDashDot;
          if  Demo then
            ZeichneGraph(Flaeche);
          Demopause;
          Z:=TMatchknoten(Y.MatchKante.Zielknoten(Y));
          Y.VorigeKante:=Ka;
          Y.MatchKante.Wert:='L    ';
```

```
    Y.MatchKante.Farbe:=clgreen;
    Y.MatchKante.Stil:=psdashdotdot;
    if  Demo then
      ZeichneGraph(Flaeche);
    Z.VorigeKante:=Y.MatchKante;
      Demopause;
    if not Z.AusgehendeKantenListe.Leer then
      for index3:=0 to Z.AusgehendeKantenliste.Anzahl-1 do
      begin
         Sucheerweiternderweg(Z,TInhaltskante(Z.AusgehendeKantenliste.Kante(Index3)));
      if ErweiternderWeggefunden then goto Fertig;
      end;
      if not Z.EingehendeKantenliste.Leer then
     for Index4:=0 to Z.EingehendeKantenliste.Anzahl-1 do
        begin
         Sucheerweiternderweg(Z,TInhaltskante(Z.EingehendeKantenliste.Kante(Index4)));
       if ErweiternderWeggefunden then goto Fertig;
        end;
        Fertig:
  end
  else
  begin
    Ka.Wert:='A    ';
   Ka.Farbe:=clblue;
   Ka.Stil:=psDashDot;
    if  Demo then
    ZeichneGraph(Flaeche);
  if Demo then Ausgabe.Caption:='Erweitender Weg gefun
     den!';
   if Demo then Ausgabe.Refresh;
    Demopause;
  ErweiternderWeggefunden:=true;
  Application.ProcessMessages;
   goto NaechsteKante;
  end;
end;
if not ErweiternderWeggefunden then
begin
  Y.MatchKante.Wert:='M';
  Y.MatchKante.Farbe:=clred;
  Y.MatchKante.Stil:=psdot;
    if  Demo then
    ZeichneGraph(Flaeche);
    Demopause;
Ka.Wert:=g.Kantenliste.Kante(Kantenliste.Position(Ka)).Wert;
   Ka.Farbe:=clblack;
   Ka.Stil:=pssolid;
    if Demo then
    ZeichneGraph(Flaeche);
  Demopause;
```

```
      Z.VorigeKante:=nil;
      Y.VorigeKante:=nil;
    end;
  end;
  NaechsteKante:
end;



procedure SucheerweiterndenWegvonKnoten(Knot:TMatchknoten);
label Ende;
var AnzKanten:Integer;
    Index1,Index2:Integer;
begin
  Application.ProcessMessages;
  if Demo then Ausgabe.Caption:=
  'Exponierten Knoten suchen... ';
  if Demo then Ausgabe.Refresh;
  ErweiternderWeggefunden:=false;
  if Knot.Knotenistexponiert then
  begin
   if Abbruch then exit;
    Startknoten:=Knot;
    AlleKnotensetzevorigeKanteaufNil;
    if Demo then Ausgabe.Caption:='Knoten: '+Knot.Wert+
    ' Erweiternder Weg wird gesucht... ';
    if Demo then Ausgabe.Refresh;
     Demopause;
    if not Knot.AusgehendeKantenliste.Leer then
     for Index1:=0 to Knot.AusgehendeKantenliste.Anzahl-1 do
        begin
        Application.Processmessages;
         if Abbruch then exit;
       SucheerweiterndenWeg(Knot,TInhaltskante
       (Knot.AusgehendeKantenliste.Kante(index1)));
       if ErweiternderWeggefunden then goto Ende;
      end;
    if not Knot.EingehendeKantenliste.Leer then
      for Index2:=0 to Knot.EingehendeKantenliste.Anzahl-1 do
      begin
        Application.Processmessages;
        if Abbruch then exit;
        SucheerweiterndenWeg(Knot,TInhaltskante
        (Knot.EingehendeKantenliste.Kante(Index2)));
        if ErweiternderWeggefunden then goto Ende;
      end;
    Ende:
  end;
  if Abbruch then exit;
```

```
    if ErweiternderWeggefunden then
     begin
      if Demo then Ausgabe.Caption:=’Vergrossere Matching’;
      if Demo then Ausgabe.Refresh;
      VergroessereMatching(G);
      AnzKanten:=(AnzahlKnoten-AnzahlexponierteKnoten)div 2;
      if Demo then Ausgabe.Caption:=
      ‘Neues Matching gefunden! Kantenzahl:
      ‘+Inttostr(AnzKanten);
      if Demo then Ausgabe.Refresh;
       if  Demo then
       ZeichneGraph(Flaeche);
      Demopause;
    end;
    if ErweiternderWeggefunden then
    ErweiternderWegfuerAlleKnotengefunden:=false;;
  end;


begin
  repeat
    if Abbruch then exit;
    ErweiternderWegfuerAlleKnotengefunden:=true;
    Application.ProcessMessages;
    if AnzahlexponierteKnoten>1 then
    begin
      if not Leer then
      for Index:=0 to Knotenliste.Anzahl-1 do
         SucheerweiterndenWegvonKnoten(TMatchknoten(Knotenliste.Knoten(Index)));
      if  Demo then ZeichneGraph(Flaeche);
       Demopause;
      if Demo then Ausgabe.Caption:=’Alle Knoten durchlaufen!‘;
      if Demo then Ausgabe.Refresh;
       Demopause;
      if  Demo and (not ErweiternderWegfueralleKnotengefunden)
        then
        begin
        Ausgabe.Caption:=’Nochmaliger Test...’;
         Ausgabe.Refresh;
         Demopause;
        end;
       if Abbruch then exit;
     end
  until ( ErweiternderWegfueralleKnotengefunden) or
  (AnzahlexponierteKnoten<=1);
end;



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;


end.
```

Unit UMath2:(nur für DWK)

```
unit UMath2;
{$F+}

interface

uses

  UList,UInhGrph,UGraph,UKante,UMath1,
  StdCtrls, Controls, ExtCtrls, Dialogs,
  Menus, Classes,
  SysUtils, WinTypes, WinProcs, Messages,Graphics,
  Forms,ClipBrd,Printers;

type

 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;override;
    property Nummer:Integer read WelcheKnotennummer write
    SetzeKnotennummer;
    property Ergebnis:string read WelchesErgebnis write
    SetzeErgebnis;
    function Wertlisteschreiben:TStringlist;override;
```

```
  procedure Wertlistelesen;override;
end;

TGleichungssystemgraph = class(TInhaltsgraph)
public
  constructor Create;override;
  procedure NumeriereKnoten;
  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;



TMarkovreduziereKnoten = class(TGleichungssystemknoten)
public
  function Randknoten:Boolean;
  function Fehler:Boolean;
  function Auswahlknoten:Boolean;
end;
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;varSliste: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;


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;override;
  property Wahrscheinlichkeit:Extended read
  WelcheWahrscheinlichkeit
  write SetzeWahrscheinlichkeit;
  property MittlereSchrittzahl:Extended read
  WelcheMittlereSchrittzahl
  write SetzeMittlereSchrittzahl;
  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;
TMarkovkante = class(TInhaltskante)
```

```
public
  function KanteistSchlingemitWerteins:Boolean;
  function Fehler:Boolean;
end;

TMarkovgraph = class(TInhaltsgraph)
public
  constructor Create;override;
  function Fehler:Boolean;
  procedure AnzahlgleichNull;
  procedure
   LadeKnotenkritischohneStartundRandknoten(Startknoten:TMarkovknoten);
  function AlleKnotenohneRandsindkritisch:Boolean;
  procedure AddiereAnzahlaufAuswahlknoten(Var
  Gesamtzahl:Extended);
  function AlleKnoten
  ausserRandsindkritischoderunterkritisch: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;Ob:TForm);
  procedure Markovabs
  (Kno:TMarkovknoten;Ausgabe1,Ausgabe2:Tlabel;
  var Gesamtzahl:Extended;var
  Sliste:TStringlist;Flaeche:TCanvas;Genauigkeit:Integer;Ob:TForm);
  procedure Markovkette(Flaeche:TCanvas;Ausgabe1,
  Ausgabe2:TLabel;var Sliste:TStringlist;Ob:TForm);
end;

TMarkovstatgraph = class(TInhaltsgraph)
public
  constructor Create;override;
  function Fehler:Boolean;
  function AnzahlRandknoten:Integer;
  procedure ZieheSteinestat(Ausgabe:TLabel;
  Gesamtzahl:Extended;Flaeche:TCanvas;Ob:TForm);
  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;Ob:TForm);
  procedure Markovkettestat
  (Flaeche:TCanvas;Ausgabe1,Ausgabe2:TLabel;
  var SListe:TStringlist;Ob:TForm);
end;


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;override;
  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;


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;override;
  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 EliminiereRueckkanten;
  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;
implementation

constructor TGleichungssystemknoten.Create;
begin
  inherited Create;
  Nummer_:=0;
  Ergebnis_:='';
  Wertlisteschreiben;
end;

procedure TGleichungssystemknoten.SetzeKnotennummer(Nu:Integer);
begin
  Nummer_:=Nu;
end;

function TGleichungssystemknoten.WelcheKnotennummer:Integer;
begin
  WelcheKnotennummer:=Nummer_;
end;

function TGleichungssystemknoten.WelchesErgebnis:string;
begin
  WelchesErgebnis:=Ergebnis_;
end;

procedure TGleichungssystemknoten.SetzeErgebnis(S:string);
begin
  Ergebnis_:=S;
end;
function TGleichungssystemknoten.Wertlisteschreiben:TStringlist;
```

```
begin
  inherited Wertlisteschreiben;
  Wertliste.Add(IntegertoString(Nummer));
  Wertliste.Add(Ergebnis);
  Wertlisteschreiben:=Wertliste;
end;


procedure TGleichungssystemknoten.Wertlistelesen;
begin
  inherited Wertlistelesen;
  Nummer:=StringtoInteger(Wertliste.Strings[Wertliste.Count-2]);
  Ergebnis:=Wertliste.Strings[Wertliste.Count-1];
end;

constructor TGleichungssystemgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TGLeichungssystemknoten;
  InhaltsKanteclass:=TInhaltskante;
end;

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;

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;

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;


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;
    Akk,Bkk,Aii,Akk,Aki,Bk,Aik,Ail,Bi,Akl:Extended;
    Index1,Index2:Integer;


  procedure Veraendereakl(Kant:TInhaltskante);
  begin
    if not Kant.KanteistSchlinge then
      if Kant.Endknoten<>Kno1  then
      begin
        Akl:=StringtoReal(Kant.Wert);
        Kno3:=TGleichungssystemknoten(Kant.Endknoten);
        if ErsteKantevonKnotenzuKnoten(Kno1,Kno3)<>nil
        then
          Ail:=StringtoReal(TInhaltskante(ErsteKantevonKnotenzuKnoten(
         Kno1,Kno3)).Wert)
        else
          Ail:=0;
        if Aii<>0 then Akl:=Akl-Ail*Aki/Aii;
        Kant.Wert:=RealtoString(Akl);
        if not (self is TMarkovreduzieregraph) then
```

```
          if Demo then
          begin
            Ausgabe.Caption:='Verändere Akl';
            LoescheBild(G,Oberflaeche);
            ZeichneGraph(Flaeche);
            Demopause;
          end;
      end;
  end;


begin
  Ende:=false;
  if Abbruch then exit;
  Kno1:=TGleichungssystemknoten(Kno);
  if AnzahlKnoten>1 then
  begin
    if not Kno1.EingehendeKantenliste.Leer then
      for Index1:=0 to Kno1.EingehendeKantenliste.Anzahl-1 do
      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);
       if  Kno2<>Kno1 then
        begin
         Aki:=StringtoReal(Ka.Wert);
         Bi:=StringtoReal(Kno1.Wert);
         Bk:=StringtoReal(Kno2.Wert);
         if ErsteKantevonKnotenzuKnoten(Kno1,Kno2)<>nil then
          Aik:=StringtoReal(
         TInhaltskante(ErsteKantevonKnotenzuKnoten
          (Kno1,Kno2)).Wert);
         if ErsteSchlingezuKnoten(Kno1)<>nil then
            Aii:=StringtoReal(TInhaltskante(ErsteSchlingezuKnoten(Kno1)).Wert);
           if Aii<>0 then
          begin
         if ErsteSchlingezuKnoten(Kno2)<>nil then
            begin
              Akk:=StringtoReal(TInhaltskante(ErsteSchlingezuKnoten(Kno2)).Wert);
            Akk:=Akk-Aik*Aki/Aii;
              TInhaltskante(ErsteSchlingezuKnoten(Kno2)).Wert:=RealtoString(Akk);
               if not (self is TMarkovreduzieregraph) then
```

```
            if Demo then
            begin
            LoescheBild(G,Oberflaeche);
             ZeichneGraph(Flaeche);
            Ausgabe.Caption:=’Bestimme Akk’;
              Demopause;
            end;
          end;
          Bk:=Bk-Aki*Bi/Aii;
          Kno2.Wert:=RealtoString(Bk);
          if not (self is TMarkovreduzieregraph) then
            if Demo then
            begin
              LoescheBild(G,Oberflaeche);
              ZeichneGraph(Flaeche);
              Ausgabe.Caption:=’Bestimme Bk’;
              Demopause;
            end;
          if not Kno2.AusgehendeKantenliste.Leer then
            for Index2:=0 to
          Kno2.AusgehendeKantenliste.Anzahl-1 do
           Veraendereakl(TInhaltskante(
          Kno2.AusgehendeKantenliste.Kante(Index2)));
        end
        else
        begin
          Ende:=true;
          exit;
        end;
      end
    end;
    Knotenloeschen(Kno1);
    ZeichneGraph(Flaeche);
  end
  else
  begin
    Akkk:=0;
    Bkk:=0;
   Kno1:=TGleichungssystemknoten(Anfangsknoten);
     Akkk:=StringtoReal(TInhaltskante(ErsteSchlingezuKnoten(Kno1)).Wert);
   Bkk:=StringtoReal(Kno1.Wert);
    if Akkk <>0 then
    begin
      Loesung:=Bkk/Akkk;
      Knotenloeschen(Kno1);
      Ausgabe.Caption:=’’;
      Ausgabe.Refresh;
      ZeichneGraph(Flaeche);
    end
   else
```

```
     begin
       Ende:=true;
       exit;
     end;
  end;
end;

function TMarkovreduziereKnoten.Randknoten:Boolean;
begin
  if  (Self.Kantenzahlausgehend=0) or
  ((self.Kantenzahlausgehend=1) and
  (TMarkovkante(self.AusgehendeKantenliste.Kante(0)).
  KanteistSchlingemitWertEins))
   or
  (StringtoReal(Self.Wert)<>0)
   then
    Randknoten:=true
   else
    Randknoten:=false;
end;


function TMarkovreduziereknoten.Fehler:Boolean;
var Index:Integer;
    Wsumme:Extended;
    Ka:TInhaltskante;
begin
  Fehler:=false;
  if not Randknoten then
  begin
    Wsumme:=0;
    if not self.AusgehendeKantenliste.Leer then
    for Index:=0 to self.AusgehendeKantenliste.Anzahl-1  do
    begin
      Application.Processmessages;
      Ka:=TInhaltskante(self.AusgehendeKantenliste.Kante(Index));
      Wsumme:=Wsumme+StringtoReal(Ka.Wert);
    end;
    if (Wsumme<(1-exp(-ln(10)*(TInhaltsgraph(Graph).Kantengenauigkeit+1))
    or (Wsumme>1+exp(-ln(10)*(TInhaltsgraph(Graph).Kantengenauigkeit+1)))
      then begin Fehler:=true;
      ShowMessage('Fehler! Wahrscheinlichkeitssumme ungleich 1
       bei Knoten: '+Self.Wert);
      exit;
    end;
  end;
end;


function TMarkovreduziereknoten.Auswahlknoten:Boolean;
```

```
var Index:Integer;
     Gefunden:Boolean;
     Ka:TInhaltskante;
begin
  Gefunden:=false;
  if not ausgehendeKantenliste.Leer then
  for Index:=0 to ausgehendeKantenliste.Anzahl-1 do
  begin
    Ka:=TInhaltskante(self.AusgehendeKantenliste.Kante(Index));
    if Ka.KanteistSchlinge and (Ka.Wert='q')
    and (self.EingehendeKantenliste.Anzahl<=2) then
    Gefunden:=true;
  end;
  if (((self.Kantenzahlausgehend=1) and
     (TMarkovkante(self.AusgehendeKantenliste.Kante(0)).
      KanteistSchlingemitWertEins)))
       or Gefunden
   then
    Auswahlknoten:=true
  else
    Auswahlknoten:=false;
end;


function TMarkovreduzieregraph.Fehler:Boolean;
var Index:Integer;
    Falsch:Boolean;
begin
  Falsch:=false;
  Fehler:=false;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Application.ProcessMessages;
      if TMarkovknoten(Knotenliste.Knoten(Index)).Fehler  then
      begin
        Fehler:=true;
        Falsch:=true;
        exit;
      end;
    end;
    if not Falsch then
      if not Kantenliste.Leer then
        for Index:=0 to kantenliste.Anzahl-1 do
        begin
          Application.ProcessMessages;
          if TMarkovkante(Kantenliste.Kante(Index)).Fehler then
          begin
            Fehler:=true;
            exit;
```

```
          end;
        end;
end;


procedure TMarkovreduzieregraph.
SetzeKnotenNullundAuswahlknotenWert;
label Marke;
var Index:Integer;
    Kno:TMarkovreduziereknoten;
    S:string;
    Gefunden:Boolean;
begin
  Gefunden:=false;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TMarkovreduziereknoten(Knotenliste.Knoten(Index));
      if not Kno.Auswahlknoten
       then
        Kno.Wert:='0'
      else
      begin
        Marke:
        Gefunden:=true;
        S:=Inputbox('Knoten: '+Kno.Wert , '(Signal-)Fluss einge
          ben:' ,'1');
         if (not StringistRealZahl(S)) or
        (StringistRealZahl(S) and (Abs(StringtoReal(S))<1.0E30))
          then
            Kno.Wert:=S
          else
        begin
        ShowMessage('Fehler! Eingabe nicht im zulässigen numeri
         schen Bereich!');
          goto Marke;
        end;
      end;
    end;
  if not Gefunden then Showmessage('Kein Auswahlknoten
  vorhanden!'+chr(13)+
  '(Auswahlknoten haben keine ausgehenden Kanten
  und nur eine Schlingenkante mit dem Wert q oder 1 '+
  ' und darüberhinaus '+'maximal nur eine eingehende Kante!)');
end;




procedure TMarkovreduzieregraph.
```

```
SetzeKnotenNullundAuswahlknoteneins;
var Index:Integer;
    Kno:TMarkovreduziereknoten;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TMarkovreduziereknoten(Knotenliste.Knoten(Index));
      if not Kno.Auswahlknoten
      then
        Kno.Wert:=’0'
      else
        Kno.Wert:=’1';
    end;
end;


procedure TMarkovreduzieregraph.FindeundreduziereSchlinge;
var Schlingensumme:Extended;
    Index1,Index2,Index3:Integer;
    Kno:TKnoten;


  procedure NeuerWertKante(Ka:TKante);
  var Re:Extended;
  begin
    if not Ka.KanteistSchlinge then
    begin
      Re:=StringtoReal(TInhaltskante(Ka).Wert);
      if abs(1-Schlingensumme)>0.00001
      then
       TInhaltskante(Ka).Wert:=RealtoString(Re/(1-Schlingen
         summe))
      else
       ShowMessage(‘Fehler:Schlingenreduzieren nicht möglich!’);
    end;
  end;


  procedure SummiereSchlinge(Ka:TKante);
  begin
    if StringtoReal(TInhaltskante(Ka).Wert) <>1 then
    if Ka.KanteistSchlinge then
    Schlingensumme:=Schlingensumme+StringtoReal(TInhaltskante(Ka).Wert);
  end;

 procedure SetzeSchlingeNull(Ka:TKante);
  begin
    if Ka.KanteistSchlinge then
      TInhaltskante(Ka).Wert:=’0';
```

```
  end;


begin
  if not Leer then
  begin
    for Index1:=0 to Knotenliste.Anzahl-1 do
    begin
      Schlingensumme:=0;
      Kno:=Knotenliste.Knoten(Index1);
      if (Kno.Kantenzahlausgehend>Kno.AnzahlSchlingen) then
        begin
          if not Kno.AusgehendeKantenliste.Leer then
            for Index2:=0 to Kno.AusgehendeKantenliste.Anzahl-1
             do
          SummiereSchlinge(Kno.AusgehendeKantenliste.Kante(Index2));
          if not Kno.AusgehendeKantenliste.Leer then
            if abs(1-Schlingensumme)>0.00001 then
            for Index3:=0 to Kno.AusgehendeKantenliste.Anzahl-1
            do
            begin
              SetzeSchlingeNull(Kno.AusgehendeKantenliste.Kante(index3));
         NeuerWertKante(Kno.AusgehendeKantenliste.Kante(index3));
            end;
        end;
    end;
  end;
end;

procedure TMarkovreduzieregraph.SetzeSchlingenaufNull;
var Re:Extended;
    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.KanteistSchlinge then
        if StringtoReal(Ka.Wert)=1 then
        begin
          Re:=StringtoReal(Ka.Wert);
          Ka.Wert:=RealtoString(1-Re);
        end;
    end;
end;



procedure TMarkovreduziereGraph.ErgaenzeSchlingenWerteins;
```

```
var Index:Integer;
    Kno:TInhaltsknoten;
    Ka:TInhaltskante;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TInhaltsknoten(Knotenliste.Knoten(Index));
      if Kno.AnzahlSchlingen=0 then
      begin
        Ka:=TInhaltskante.Create;
        Ka.Wert:=’1';
        Ka.Weite:=0;
        Ka.Gerichtet:=true;
        FuegeKanteein(Kno,Kno,true,Ka);
      end;
    end;
end;


procedure TMarkovreduzieregraph.SpeichereSchlingenundKantenum;
var Re:Extended;
    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.KanteistSchlinge
      then
      begin
       Re:=StringtoReal(Ka.Wert);
       if self.AnzahlKnoten>1 then  Ka.Wert:=RealtoString(1-
         Re);
      end
      else
      begin
        Re:=StringtoReal(Ka.Wert);
        Ka.Wert:=RealtoString((-Re));
      end;
    end;
end;



procedure TMarkovreduzieregraph.SucheundreduziereParallelkanten;
var Index1,Index2,Index3,Zaehler:Integer;
    Kno:TMarkovreduziereKnoten;
    Ka,Kant:TInhaltskante;
```

```
  procedure SucheParallelkante(Kant:TInhaltsKante);
  begin
    if ((Kant.Endknoten=Ka.Endknoten) and
    (StringtoReal(Kant.Wert)>0)) then
     begin
      Zaehler:=Zaehler+1;
      if Zaehler>1 then
      begin
        Ka.Wert:=RealtoString(StringtoReal(Ka.Wert)+StringtoReal(Kant.Wert));
       Kant.Wert:='0';
      end;
    end;
end;

begin
  if not Leer then
    for Index1:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TMarkovreduziereKnoten(Knotenliste.Knoten(Index1));
      if not Kno.AusgehendeKantenliste.Leer then
      for Index2:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
        begin
          Zaehler:=0;
       Ka:=TInhaltskante(Kno.AusgehendeKantenliste.Kante(Index2));
          if (not Ka.KanteistSchlinge) then
          for Index3:=0 to Kno.AusgehendeKantenliste.Anzahl-1
             do
             begin
               Kant:=TInhaltskante(Kno.AusgehendeKantenliste.Kante(Index3));
             SucheParallelkante(Kant);
            end;
        end;
    end;
end;

procedure TMarkovreduziereGraph.LoescheKantenmitWertNull;
var N,Index:Integer;
    Gefunden:Boolean;
    Ka:TInhaltskante;

begin
  repeat
    Gefunden:=false;
    if not Kantenliste.Leer then
    begin
      N:=Kantenliste.Anzahl-1;
      Index:=0;
     while (Gefunden=false) and (Index<=N) do
```

```
       begin
       if StringtoReal(Kantenliste.Kante(Index).Wert)=0 then
         begin
         Ka:=TInhaltskante(Kantenliste.Kante(Index));
           Gefunden:=true;
         end;
        Index:=Index+1;
       end;
    end;
    if Gefunden then Kanteloeschen(Ka)
  until not Gefunden;
end;

function TMarkovreduzieregraph.AnzahlRandknoten:Integer;
var Index,Anzahl:Integer;
begin
  Anzahl:=0;
  if not Knotenliste.Leer then
  for Index:=0 to Knotenliste.Anzahl-1 do
   if TMarkovreduziereKnoten
   (Knotenliste.Knoten(Index)).Randknoten
    then
      Anzahl:=Anzahl+1;
  AnzahlRandknoten:=Anzahl;
end;



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
   ShowMessage('Randknoten können erst gelöscht werden,wenn ma
    ximal'
   +chr(13)+'noch ein innerer Knoten zusätzlich vorhanden
   ist!');
   goto Endproc;
  end;
  ErgaenzeKanten;
  if Demo then LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  if Demo then Ausgabe2.Caption:='Ergänze Kanten';
  Demopause;
```

```
  if Abbruch then goto Endproc;
  SpeichereSchlingenundKantenum;
  if Demo then LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  if Demo then Ausgabe2.Caption:=’Speichere Schlingen und Kanten
  um’;
  Demopause;
  if Abbruch then goto Endproc;
  ErgaenzeSchlingenWerteins;
  if Demo then LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  if Demo then Ausgabe2.Caption:=’Ergänze Schlingen Wert eins’;
  Demopause;
  if Abbruch then goto Endproc;
 EliminiereKnoten(Kno,Flaeche,Ausgabe1,Loesung,Ende,G,Oberflaeche);
  if Demo then LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  if Demo then Ausgabe2.Caption:=’Eliminiere Knoten’;
  Demopause;
  if Abbruch then goto Endproc;
  SpeichereSchlingenundKantenum;
  if Demo then LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  if Demo then Ausgabe2.Caption:=’Speichere Schlingen
  und Kanten um’;
  Demopause;
  if Abbruch then goto Endproc;
  LoescheKantenmitWertNull;
  if Demo then LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  if Demo then Ausgabe2.Caption:=’Lösche Kanten Wert Null’;
  Demopause;
  FindeundreduziereSchlinge;
  LoescheKantenmitWertNull;
  if Demo then LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  if Demo then Ausgabe2.Caption:=’Finde und reduziere Schlin
  gen’;
  Demopause;
  ZeichneGraph(Flaeche);
  Endproc:
end;



procedure TMarkovreduziereGraph.
Graphinit(Markov:Boolean;Flaeche:TCanvas;
  Var Oberflaeche:TForm;
  var G:TInhaltsgraph;Ausgabe2:Tlabel);
label Endproc;
```

```
begin
  if Markov
  then
    SetzeKnotenNullundAuswahlknoteneins
  else
    SetzeKnotenNullundAuswahlKnotenWert;
  ZeichneGraph(Flaeche);
  Demopause;
  if Abbruch then goto Endproc;
  SucheundreduziereParallelkanten;
  if Demo then LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  if Demo then Ausgabe2.Caption:=’Suche und reduziere Parallel
  kanten’;
  Demopause;
  if Abbruch then goto Endproc;
  FindeundReduziereSchlinge;
  if Demo then LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  if self.Demo then Ausgabe2.Caption:=’Finde und reduziere
  Schlingen’;
  Demopause;
  if Abbruch then goto Endproc;
  SetzeSchlingenaufNull;
  LoescheKantenmitwertNull;
  if Demo then LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  if Demo then Ausgabe2.Caption:=’Lösche Kanten mit Wert Null’;
  Demopause;
  Ausgabe2.Caption:=’’;
  LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  Endproc:
end;

constructor TMarkovknoten.Create;
begin
  inherited Create;
  Wahrscheinlichkeit_:=0;
  MittlereSchrittzahl_:=0;
  Anzahl_:=0;
  Minimum_:=0;
  VorigeAnzahl_:=0;
  Delta_:=0;
  Ergebnis_:=’’;
  Anzeige_:=’’;
  Wertlisteschreiben;
end;

procedure TMarkovknoten.SetzeWahrscheinlichkeit(Wa:Extended);
```

```
begin
   Wahrscheinlichkeit_:=Wa;
end;

function TMarkovknoten.WelcheWahrscheinlichkeit:Extended;
begin
  WelcheWahrscheinlichkeit:=Wahrscheinlichkeit_;
end;

procedure TMarkovknoten.SetzeMittlereSchrittzahl(Schr:Extended);
begin
  MittlereSchrittzahl_:=Schr;
end;


function TMarkovknoten.WelcheMittlereSchrittzahl:Extended;
begin
  WelcheMittlereSchrittzahl:=MittlereSchrittzahl_;
end;

procedure TMarkovknoten.SetzeAnzahl(Anz:Extended);
begin
  Anzahl_:=Anz;
end;

function TMarkovknoten.WelcheAnzahl:Extended;
begin
  WelcheAnzahl:=Anzahl_;
end;

procedure TMarkovknoten.SetzeMinimum(Mi:Extended);
begin
  Minimum_:=Mi;
end;

function TMarkovknoten.WelchesMinimum:Extended;
begin
  WelchesMinimum:=Minimum_;
end;

procedure TMarkovknoten.SetzevorigeAnzahl(Vaz:Extended);
begin
  VorigeAnzahl_:=Vaz;
end;

function TMarkovknoten.WelchevorigeAnzahl:Extended;
begin
  WelchevorigeAnzahl:=VorigeAnzahl_;
end;
procedure TMarkovknoten.SetzeDelta(De:Extended);
```

```
begin
  Delta_:=De;
end;

function TMarkovknoten.WelchesDelta:Extended;
begin
  WelchesDelta:=Delta_;
end;
procedure TMarkovknoten.SetzeErgebnis(S:string);
begin
  Ergebnis_:=S;
end;


function TMarkovknoten.WelchesErgebnis:string;
begin
  WelchesErgebnis:=Ergebnis_;
end;

function TMarkovknoten.WelcheAnzeige:string;
begin
  WelcheAnzeige:=Anzeige_;
end;


procedure TMarkovknoten.SetzeAnzeige(S:string);
begin
  Anzeige_:=S;
end;


function TMarkovknoten.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;
  Wertliste.Add(Realtostring(Wahrscheinlichkeit));
  Wertliste.Add(Realtostring(MittlereSchrittzahl));
  Wertliste.Add(Realtostring(Anzahl));
  Wertliste.Add(Realtostring(Minimum));
  Wertliste.Add(Realtostring(VorigeAnzahl));
  Wertliste.Add(Realtostring(Delta));
  Wertliste.Add(Ergebnis);
  Wertliste.Add(Anzeige);
  Wertlisteschreiben:=Wertliste;
end;


procedure TMarkovknoten.Wertlistelesen;
begin
  inherited Wertlistelesen;
 Wahrscheinlichkeit:=StringtoReal(Wertliste.Strings[Wertliste.Count-
```

```
  8]);
  MittlereSchrittzahl:=StringtoReal(Wertliste.Strings[Wertliste.Count-
  7]);
  Anzahl:=StringtoReal(Wertliste.Strings[Wertliste.Count-6]);
  Minimum:=StringtoReal(Wertliste.Strings[Wertliste.Count-5]);
  VorigeAnzahl:=StringtoReal(Wertliste.Strings[Wertliste.Count-
  4]);
  Delta:=StringtoReal(Wertliste.Strings[Wertliste.Count-3]);
  Ergebnis:=Wertliste.Strings[Wertliste.Count-2];
  Anzeige:=Wertliste.Strings[Wertliste.Count-1];
end;



procedure TMarkovknoten.ErhoeheAnzahlumEins(var
Steine:Extended);
begin
  Anzahl:=Anzahl+1;
  Steine:=Steine+1;
end;

function TMarkovKnoten.Randknoten:Boolean;
begin
  if (Self.Kantenzahlausgehend=0) or
    ((Self.Kantenzahlausgehend=1) and
    (TMarkovkante(Self.AusgehendeKantenliste.Kante(0)).
    KanteistSchlingemitWertEins))
  then
   Randknoten:=true
  else
   Randknoten:=false;
end;

function TMarkovKnoten.KnotenungleichRandistkritisch:Boolean;
begin
  if not Randknoten
  then
    KnotenungleichRandistkritisch:=(Anzahl=Minimum-1)
  else
    KnotenungleichRandistkritisch:=true;
end;

function TMarkovknoten.Fehler:Boolean;
Var Index:Integer;
    Wsumme:Extended;
    Ka:TMarkovkante;
begin
  Fehler:=false;
  if not Randknoten then
  begin
```

```
    Wsumme:=0;
    if not self.AusgehendeKantenliste.Leer then
     for Index:=0 to self.AusgehendeKantenliste.Anzahl-1  do
       begin
        Application.Processmessages;
       Ka:=TMarkovkante(self.AusgehendeKantenliste.Kante(Index));
        Wsumme:=Wsumme+StringtoReal(Ka.Wert);
       end;
     if (Wsumme<(1-exp(-ln(10)*(TInhaltsgraph(Graph).Kantengenauigkeit+1)))
      or (Wsumme>1+exp(-ln(10)*(TInhaltsgraph(Graph).Kantengenauigkeit+1)))
        then begin Fehler:=true;
        ShowMessage('Fehler! Wahrscheinlichkeitssumme ungleich
         Eins bei Knoten: '+Self.Wert);
        exit;
      end;
  end;
end;



function TMarkovknoten.Auswahlknoten:Boolean;
var Index:Integer;
     Gefunden:Boolean;
begin
  if self.Randknoten then
  begin
    if ((Self.Kantenzahlausgehend=1)
    and (TMarkovkante(Self.AusgehendeKantenliste.Kante(0)).
    KanteistSchlingemitWertEins))
    then
      Auswahlknoten:=true
    else
      Auswahlknoten:=false;
  end
  else
    Auswahlknoten:=false;
end;


function TMarkovknoten.
KnotenungleichRandistueberkritisch:Boolean;
begin
  if not Randknoten
  then
    KnotenungleichRandistueberkritisch:=(Anzahl>Minimum-1)
  else
    KnotenungleichRandistueberkritisch:=false;
end;

procedure TMarkovknoten.LadeKnotenkritisch;
```

```
begin
  Anzahl:=Minimum-1;
end;


function TMarkovkante.KanteistSchlingemitWerteins:Boolean;
begin
  if KanteistSchlinge and(StringtoReal(Wert)=1)
  then
    KanteistSchlingemitWerteins:=true
  else
    KanteistSchlingemitWerteins:=false;
end;

function TMarkovkante.Fehler:Boolean;
begin
  Fehler:=false;
  if (StringtoReal(Self.Wert)>1) or (StringtoReal(Self.Wert)<0)
then
  begin
    Fehler:=true;
    ShowMessage('Wahrscheinlichkeitwert unzulässig bei Kante
    'self.Wert);
    exit;
  end;
end;

constructor TMarkovgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TMarkovknoten;
  InhaltsKanteclass:=TMarkovkante;
end;

function TMarkovgraph.Fehler:Boolean;
var Index:Integer;
begin
  Fehler:=false;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Application.ProcessMessages;
      if TMarkovknoten(Knotenliste.Knoten(Index)).Fehler  then
      begin
        Fehler:=true;
        exit;
      end;
    end;
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
```

```
    begin
      Application.ProcessMessages;
      if TMarkovkante(Kantenliste.Kante(Index)).Fehler then
      begin
        Fehler:=true;
        exit;
      end;
    end;
end;

procedure TMarkovgraph.AnzahlgleichNull;
var Index:Integer;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      TMarkovknoten(Knotenliste.Knoten(Index)).Anzahl:=0;
end;

procedure TMarkovgraph.
LadeKnotenkritischohneStartundRandknoten(Startknoten:TMarkovknoten);
var Index:Integer;
    Kno:TMarkovknoten;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TMarkovknoten(Knotenliste.Knoten(Index));
      if (Kno<> Startknoten) and (not Kno.Randknoten) then
        Kno.Anzahl:=Kno.Minimum-1;
    end;
end;

function TMarkovgraph.AlleKnotenohneRandsindkritisch:Boolean;
var Index:Integer;
    Gefunden:Boolean;
begin
  Gefunden:=false;
  if not self.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      if not TMarkovknoten(Knotenliste.Knoten(Index)).
      KnotenungleichRandistkritisch
        then
        Gefunden:=true;
        if not Gefunden
         then
        AlleKnotenohneRandsindkritisch:=true
         else
        AlleKnotenohneRandsindkritisch:=false;
end;
procedure TMarkovgraph.AddiereAnzahlaufAuswahlknoten(var
```

```
Gesamtzahl:Extended);
var Index:Integer;

  procedure AddiereAnzahlAuswahlknoten(Kno:TKnoten);
  begin
    if TMarkovKnoten(Kno).Auswahlknoten then
      Gesamtzahl:=Gesamtzahl+TMarkovknoten(Kno).Anzahl;
  end;

begin
  Gesamtzahl:=0;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      AddiereAnzahlAuswahlknoten(Knotenliste.Knoten(Index));
end;

function TMarkovgraph.AlleKnotenausserRand
sindkritischoderunterkritisch:Boolean;
var Index:Integer;
    Gefunden:Boolean;
begin
  Gefunden:=false;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      if TMarkovknoten(Knotenliste.Knoten(Index)).
      KnotenungleichRandistueberkritisch then Gefunden:=true;
  if not Gefunden
  then
    AlleKnotenausserRandsindkritischoderunterkritisch:=true
  else
    AlleKnotenausserRandsindkritischoderunterkritisch:=false;
end;

procedure TMarkovgraph.
LadeStartknotenkritischnach(Startknoten:TMarkovknoten;
  var Steine:Extended);
begin
  Steine:=Steine+Startknoten.Minimum-1-Startknoten.Anzahl;
  Startknoten.Anzahl:=Startknoten.Minimum-1;
  if Steine>2E23 then
  begin
    ShowMessage('Lösung nicht gefunden!');
    Stop:=true;
    exit;
  end;
end;

function TMarkovgraph.EnthaeltAuswahlknoten:Boolean;
var Index:Integer;
begin
```

```
  EnthaeltAuswahlknoten:=false;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      if TMarkovknoten(Knotenliste.Knoten(Index)).Auswahlknoten
        then
        EnthaeltAuswahlknoten:=true;
end;


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;
    Divisor:=Anfangswert;
    if not Kno.AusgehendeKantenliste.Leer then
      for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
        FindeDivisor(Kno.AusgehendeKantenliste.Kante(Index));
    Kno.Minimum:=Round(Anfangswert) div Round(Divisor);
  end;



begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      BestimmeMinimum(TMarkovKnoten(Knotenliste.Knoten(Index)));
end;


procedure TMarkovgraph.SetzeKnotenWahrscheinlichkeitgleichNull;
var Index:Integer;


  procedure WahrscheinlichkeitgleichNull(Kno:TMarkovknoten);
```

```
  begin
    Kno.Wahrscheinlichkeit:=0;
    Kno.MittlereSchrittzahl:=0;
  end;

begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    WahrscheinlichkeitgleichNull(TMarkovknoten(Knotenliste.Knoten(Index)));
end;


procedure TMarkovgraph.ZieheSteineabs(Ausgabe:Tlabel;
var Schrittzahl:Extended;Flaeche:TCanvas;Ob:TForm);
var Index,H:Integer;
  procedure ZieheAnzahl(Kno:TMarkovknoten);
  var Rest,Diff:Extended;
      Index:Integer;
      ZahlderzuziehendenSteine:Extended;
      SumSteine:Extended;


     procedure VerteileZahlderzuziehendenSteine(Ka:Tkante);
     var Y:TMarkovKnoten;
         Kant:TMarkovkante;
     begin
       Kant:=TMarkovkante(Ka);
       Y:=TMarkovknoten(Ka.Endknoten);
       Y.Anzahl:=Y.Anzahl+Round(StringtoReal(Kant.Wert)*
       ZahlderzuziehendenSteine);
         SumSteine:=SumSteine+Round(Round(100000*StringtoReal(Kant.Wert))/
       100000*ZahlderzuziehendenSteine);
       if TInhaltsgraph(self).Demo then
         begin
          Ausgabe.Refresh;
         Ausgabe.Caption:='Knoten: '+y.Wert+': Anzahl: '
          +RealtoString(Y.Anzahl)+'                    ';
         Knotenwertposition:=3;H:=Knotengenauigkeit;Knotengenauigkeit:=0;
         LoescheBild(TInhaltsgraph(self),Ob);ZeichneGraph(Flaeche);
          Demopause;
         Knotengenauigeit:=H;Knotenwertposition:=0;
         end;
      end;

  begin
    if not Kno.Randknoten then
    begin
      SumSteine:=0;
      Rest:=Round(Kno.Anzahl) mod Round(Kno.Minimum);
      ZahlderzuziehendenSteine:=Kno.Anzahl-Rest;
```

```
      if ZahlderzuziehendenSteine>0 then
      begin
        Schrittzahl:=Schrittzahl+ZahlderzuziehendenSteine;
        if Schrittzahl>1E25 then
        begin
          ShowMessage('Lösung nicht gefunden!');
          TInhaltsgraph(self).Stop:=true;
          exit;
        end;
        Kno.Anzahl:=Rest;
        if not Kno.ausgehendeKantenliste.Leer then
          for Index:=0 to Kno.ausgehendeKantenliste.Anzahl-1 do
            VerteileZahlderzuziehendenSteine
           (Kno.AusgehendeKantenliste.Kante(Index));
        Diff:=SumSteine-ZahlderzuziehendenSteine;
        Schrittzahl:=Schrittzahl+Diff;
        Kno.Anzahl:=Kno.Anzahl+Diff;
      end;
    end;
  end;

begin
  if not Leer then
  for Index:=0 to Knotenliste.Anzahl-1 do
    ZieheAnzahl(TMarkovknoten(Knotenliste.Knoten(Index)));
end;


procedure TMarkovgraph.Markovabs
(Kno:TMarkovknoten;Ausgabe1,Ausgabe2:TLabel;
var Gesamtzahl:Extended;Var Sliste:TStringlist;
Flaeche:TCanvas;Genauigkeit:Integer;Ob:TForm);
label Endproc;
var Index,H:Integer;
    Startknoten:TMarkovknoten;
    Schrittzahl,Steine:Extended;
begin
  if  Abbruch then  goto Endproc;
  AnzahlgleichNull;
  Startknoten:=Kno;
  Schrittzahl:=0;
  Steine:=0;
  Gesamtzahl:=0;
  BestimmeMinimum(Genauigkeit);
  if not Startknoten.Randknoten then
  begin
    LadeKnotenkritischohneStartundRandknoten(Startknoten);
    Startknoten.LadeKnotenkritisch;
    if Demo then
    begin
```

```
      Ausgabe1.Caption:='Startknoten: '+Kno.Wert+':'+' Steine:
      '+Realtostring(Steine)+'
       Schritte: ';
      if Schrittzahl< 2000000000 then Ausgabe1.Caption:=
      Ausgabe1.Caption+RealtoString(Schrittzahl);
      Ausgabe1.Refresh;H:=Knotengenauigkeit;Knotengenauigkeit:=0;
      Knotenwertposition:=3;
      ZeichneGraph(Flaeche);
       Demopause;
      Knotengenauigkeit:=H;Knotenwertposition:=0;
    end;
    repeat
      if Abbruch then goto Endproc;
      if Demo then
      begin
       Ausgabe1.Caption:='Startknoten: '+Kno.Wert+':'+' Steine:
'      +Realtostring(Steine)+' Schritte: ';
       if Schrittzahl< 2000000000 then Ausgabe1.Caption:=
       Ausgabe1.Caption+RealtoString(Schrittzahl);
        Ausgabe1.Refresh;
       Knotenwertposition:=3;H:=Knoengenauigkeit;Knotengenauigkeit:=0;
       ZeichneGraph(Flaeche);
        Demopause;
       Knotengenauigkeit:=H;Knotenwertposition:=0;
      end;
      Application.ProcessMessages;
      Startknoten.ErhoeheAnzahlumeins(Steine);
      if Demo then
      begin
       Knotenwertposition:=3;H:=Knotengenauigkeit;Knotengenauigkeit:=0;
       ZeichneGraph(Flaeche);
        Demopause;
       Knotengenauigkeit:=H;Knotenwertposition:=0;
      end;
      repeat
       Application.ProcessMessages;
        if Abbruch then goto endproc;
       ZieheSteineabs(Ausgabe2,Schrittzahl,Flaeche,Ob)
     until AlleKnotenausserRandsindkritischoderunterkritisch
      or Abbruch;
      LadeStartKnotenkritischnach(Startknoten,Steine);
       if Abbruch then goto Endproc;
      if Demo then
      begin
      Knotenwertposition:=3;H:=Knotengenauigeit;Knotengenauigkeit:=0;
      ZeichneGraph(Flaeche);
       Demopause;
      Knotengenauigkeit:=H;Knotenwertposition:=0;
      end;
  until  AlleKnotenohneRandsindKritisch or Abbruch or (Steine>1E12);
```

```
if Abbruch then goto Endproc;if Steine>1E12 then ShowMessage('Zuviele
Steine!Abbruch!Näherungslösung für '+Startknoten.Wert);
AddiereAnzahlaufAuswahlknoten(Gesamtzahl);Startknoten.
Wahrscheinlichkeit:=Gesamtzahl/Steine;Startknoten.
MittlereSchrittzahl:=Schrittzahl/Steine;end
else
 if Startknoten.Auswahlknoten then
 begin
   if Abbruch then goto Endproc;
   Startknoten.Wahrscheinlichkeit:=1;
   Startknoten.MittlereSchrittzahl:=0;
 end
 else
   if Startknoten.Randknoten then
   begin
    Startknoten.Wahrscheinlichkeit:=0;
    Startknoten.MittlereSchrittzahl:=0;
   end;
 if Demo then
 begin
   Ausgabe1.Caption:='Startknoten: '+Kno.Wert+':'+' Steine:
   '+Realtostring(Steine)
    +' Schritte: ';
   if Schrittzahl< 2000000000 then Ausgabe1.Caption:=
   Ausgabe1.Caption+RealtoString(Schrittzahl);
   Ausgabe1.Refresh;
   Knotenwertposition:=3;H:=Knotengenauigkeit;Knotengenauigkeit:=0;
   ZeichneGraph(Flaeche);
    Demopause;
   Knotengenauigkeit:=H;Knotenwertposition:=0;
 end;
  with Startknoten do
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;
 if Demo then
 with Startknoten do
 begin
  Ausgabe1.Caption:='Steine im Endzustand:
  '+Realtostring(Gesamtzahl)+' Gesamtsteine: '
 +Realtostring(Steine);;
```

```
    Ausgabe1.Refresh;
    Demopause;
   Ausgabe1.Caption:=’w:
   ‘+RundeZahltostring(Wahrscheinlichkeit,Knotengenauigkeit)
    +’ S:
   ‘+RundeZahltostring(MittlereSchrittzahl,Knotengenauigkeit);
   Ausgabe1.Refresh;
   Demopause;
   Ausgabe1.Caption:=’’;
   Ausgabe2.Caption:=’’;
   Ausgabe1.Refresh;
   Ausgabe2.Refresh;
  end;
  Endproc:
end;

procedure TMarkovgraph.
Markovkette(Flaeche:TCanvas;Ausgabe1,Ausgabe2:TLabel;
var Sliste:TStringlist;Ob:TForm);
label Endproc;
var Kno:TMarkovknoten;
    Strz,Fehlerstri:string;
    NurEinknoten,eingabe:Boolean;
    Index:Integer;
    Gesamtzahl:Extended;
    Str:string;
    Genauigkeit:Integer;
begin
  Str:=’3';repeat Eingabe:=Inputquery
    (‘Eingabe Genauigkeit:’,’Genauigkeit (1 bis 5 Stellen):’,str);
    5 Stellen):’,str);if StringtoInteger(Str)<=0 then Genauigkeit:=3 else
    Genauigkeit:=abs(StringtoInteger(Str));if not Eingabe then exit;
    Knotengenauigkeit:=Genauigkeit;Kantengenauigkeit:=Genauigkeit
  until Eingabe and (Genauigkeit<6);
  if not EnthaeltAuswahlknoten then begin
   ShowMessage(‘Keine ausgewählten Zustände (Knoten) des
   ‘+chr(13)+’Randes
   (mit 1) markiert!’);
   exit;
 end;
 if MessageDlg(‘Nur ein
  Knoten?’,mtConfirmation,[mbYes,mbNo],0)=mryes
 then
   NureinKnoten:=true
 else
   NurEinknoten:=false;
  if Fehler
  then
   ShowMessage(‘Zuerst Fehler beheben!');
   exit;
```

```
   end;
   BestimmeMinimum(Genauigkeit);Knotengenauigkeit:=Genauigkeit;
   Kantengenauigkeit:=Genauigkeit;if not Demo  then
   begin
    Ausgabe1.Caption:='Berechnung läuft...';
    Ausgabe1.Refresh;
   end;
   SetzeKnotenWahrscheinlichkeitgleichNull;
   Kno:=TMarkovknoten(LetzterMausklickknoten);
   if Kno=nil then Kno:=TMarkovknoten(Self.Anfangsknoten);
   Gesamtzahl:=0;
   if  NurEinKnoten
   then
     Markovabs(Kno,Ausgabe1,Ausgabe2,Gesamtzahl,SListe,Flaeche,Genauigkeit,O)
    else
    if not Leer then
      for Index:=0 to Knotenliste.Anzahl-1 do
        Markovabs(Tmarkovknoten(Knotenliste.knoten(Index)),
        Ausgabe1,Ausgabe2,Gesamtzahl,SListe,
        Flaeche,Genauigkeit,Ob);
   if Abbruch then  goto endproc;
   Ausgabe2.Caption:='';
   Ausgabe2.Refresh;
   if  Ausgabe1.Caption='Berechnung läuft...' then
   begin
    Ausgabe1.Caption:='';
    Ausgabe1.Refresh;
  end;
  Strz:='Startknoten '+Kno.Wert+':'+chr(13)+
  'Wahrscheinlichkeit:
  '+RundeZahltoString(Kno.Wahrscheinlichkeit,
  Knotengenauigkeit)
   +chr(13)+'Mittlere Schrittzahl: '+
   RundeZahltoString(Kno.MittlereSchrittzahl,Knotengenauigkeit);
  ShowMessage(Strz);
  Endproc:
  Ausgabe1.Caption:='';
  Ausgabe1.Refresh;
end;


constructor TMarkovstatgraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TMarkovknoten;
  InhaltsKanteclass:=TMarkovkante;
end;
function TMarkovstatgraph.Fehler:Boolean;
```

```
var Index:Integer;
begin
  Fehler:=false;
  if not Leer then
  for Index:=0 to Knotenliste.Anzahl-1 do
  begin
    Application.ProcessMessages;
    if TMarkovknoten(Knotenliste.Knoten(Index)).Fehler  then
    begin
      Fehler:=true;
      exit;
    end;
  end;
  if not Kantenliste.Leer then
    for Index:=0 to Kantenliste.Anzahl-1 do
    begin
      Application.ProcessMessages;
      if TMarkovkante(Kantenliste.Kante(Index)).Fehler then
      begin
        Fehler:=true;
        exit;
      end;
    end;
end;

function TMarkovstatgraph.AnzahlRandknoten:Integer;
var Index,Anzahl:Integer;
begin
  Anzahl:=0;
  if not Knotenliste.Leer then
  for Index:=0 to Knotenliste.Anzahl-1 do
    if (TMarkovKnoten(Knotenliste.Knoten(Index)).
    AusgehendeKantenliste.Leer)or(TMarkovKante(Knotenliste.Knoten(Index).
     Kante(0)).KanteistSchlingemitWertEins) then
      Anzahl:=Anzahl+1;
  AnzahlRandknoten:=Anzahl;
end;


procedure TMarkovstatgraph.
ZieheSteinestat(Ausgabe:Tlabel;Gesamtzahl:Extended;Flaeche:TCanvas;Ob:TForm)
Var Index:Integer;

  procedure ZieheAnzahl(Kno:TMarkovknoten);
  var Rest,Diff,ZahlderzuziehendenSteine:Extended;
      Index,H:Integer;

    procedure VerteileAnzahlstat(Kant:TMarkovKante);
    var Y:TMarkovknoten;
  begin
```

```
    Y:=TMarkovknoten(Kant.Endknoten);
     Y.Delta:=Y.Delta+Round(StringtoReal(Kant.Wert)*ZahlderzuziehendenSteine);


    Application.ProcessMessages;
  end;

begin

  Application.ProcessMessages;
  if Demo then
  begin
    Ausgabe.Refresh;
    Ausgabe.Caption:='';
    Ausgabe.Caption:=Ausgabe.Caption+' Knoten: '+Kno.Wert+': An
    zahl: ';
    Ausgabe.Caption:=Ausgabe.Caption+Realtostring(Kno.Anzahl);
    Ausgabe.Caption:=Ausgabe.Caption+' Steine:
   '+RealtoString(Gesamtzahl);
    Knotenwertposition:=3;H:=Knotengenauigkeit;Knotengenauigkeit:=0;
    LoescheBild(TInhaltsgraph(self),Ob);ZeichneGraph(Flaeche);
    Demopause;
    Knotengenauigkeit:=H;Knotenwertposition:=0;
  end;
  Rest:=Round(Kno.Anzahl) mod Round(Kno.Minimum);
  ZahlderzuziehendenSteine:=Kno.Anzahl-Rest;
  if ZahlderzuziehendenSteine>0 then
  begin
    Kno.Anzahl:=Rest;
    if not Kno.ausgehendeKantenliste.Leer then
      for Index:=0 to Kno.ausgehendeKantenliste.Anzahl-1 do
     VerteileAnzahlstat(TMarkovkante(Kno.ausgehendeKantenliste.Kante(Index)));


  end;
end;

begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      ZieheAnzahl(TMarkovknoten(Knotenliste.Knoten(Index)));
end;



procedure TMarkovstatgraph.LadeKnotenAnzahlmitMinimum;
var Index:Integer;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
```

```
      with TMarkovknoten(Knotenliste.Knoten(Index)) do
        Anzahl:=Minimum;
end;

procedure TMarkovstatgraph.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;
  Divisor:=Anfangswert;
  if not Kno.ausgehendeKantenliste.Leer then
    for Index:=0 to Kno.AusgehendeKantenliste.Anzahl-1 do
      FindeDivisor(Kno.AusgehendeKantenliste.Kante(Index));
  Kno.Minimum:=Round(Anfangswert) div Round(Divisor);
end;



begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      BestimmeMinimum(TMarkovKnoten(Knotenliste.Knoten(Index)));
end;




procedure TMarkovstatgraph.ErhoeheAnzahlumDelta;
var Index:Integer;
begin
  if not Leer then
    for Index:=0 to knotenliste.Anzahl-1 do
      with TMarkovknoten(Knotenliste.Knoten(Index)) do
      begin
        Anzahl:=Anzahl+Delta;
```

```
        Delta:=0;
      end
end;

procedure TMarkovstatgraph.SpeichereVerteilung;
var Index:Integer;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      with TMarkovknoten(Knotenliste.Knoten(Index)) do
        VorigeAnzahl:=Anzahl;
end;

procedure TMarkovstatgraph.SummiereAnzahlstat(var
Gesamtzahl:Extended);
var Index:Integer;
begin
  Gesamtzahl:=0;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      with TMarkovknoten(Knotenliste.Knoten(Index)) do
        Gesamtzahl:=Gesamtzahl+Anzahl;
end;

procedure TMarkovstatgraph.
BestimmeWahrscheinlichkeit(Gesamtzahl:Extended);
var Index:Integer;
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      with TMarkovknoten(Knotenliste.Knoten(Index)) do
      begin
        Wahrscheinlichkeit:=Anzahl/Gesamtzahl;

        if Wahrscheinlichkeit<> 0
        then
          MittlereSchrittzahl:=1/Wahrscheinlichkeit

        else
          MittlereSchrittzahl:=0;
      end;

end;


procedure TMarkovstatgraph.ErzeugeAusgabe(var S:string);
Var Index:Integer;
begin
  S:='';
  if not Leer then
```

```
    for Index:=0 to Knotenliste.Anzahl-1 do
      with TMarkovknoten(Knotenliste.Knoten(Index)) do
        begin
         S:=S+Wert+':'+
       RundeZahltostring(Wahrscheinlichkeit,Knotengenauigkeit)+
         '  ';
         Anzeige:=RundeZahltostring(Wahrscheinlichkeit,Knotengenauigkeit);
        end;
end;



function TMarkovstatgraph.
VorigeAnzahlgleichAnzahl(Gesamtzahl:Extended):Boolean;
var Index:Integer;
    Gefunden:boolean;

  function KnotenVorigeAnzahlgleichAnzahl
  (Kno:TMarkovknoten):Boolean;
  begin
    if  Gesamtzahl< trunc(exp(ln(10)*(Knotengenauigkeit)))
    then
      KnotenVorigeAnzahlgleichAnzahl:=(Abs(Kno.Anzahl-
      Kno.VorigeAnzahl)<1)
    else
      KnotenVorigeAnzahlgleichAnzahl:=(Abs(Kno.Anzahl-
      Kno.VorigeAnzal)<Trunc(Gesamtzahl/exp(ln(10)*Knotengenauigkeit)));
  end;

begin
  Gefunden:=false;
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      if not KnotenVorigeAnzahlgleichAnzahl
     (TMarkovknoten(Knotenliste.Knoten(Index)))
     then
       Gefunden:=true;
  if Gefunden
  then
    result:=false
  else
    result:=true;
end;


procedure TMarkovstatgraph.LadeKnotenAnzahlmitkleinsterZahl;
var Faktor:Extended;
    Index:Integer;
    Kno:TMarkovknoten;
```

```
begin
  if not Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
    begin
      Kno:=TMarkovknoten(Knotenliste.Knoten(Index));
      Faktor:=0;
      Faktor:=Round(Kno.Anzahl) div Round(Kno.Minimum);
      if Round(Kno.Anzahl) mod Round(Kno.Minimum)<>0 then begin
      Faktor:=Faktor+1;Kno.Anzahl:=Faktor*Kno.Minimum;end;
    end;
end;



procedure TMarkovstatgraph.Markovstat
(Ausgabe1,Ausgabe2:Tlabel;var Gesamtzahl:Extended;
var Sliste:TStringlist;
Flaeche:TCanvas;Genauigkeit:Integer;Ob:TForm);
var Wiederholung:Longint;
   Steine,Schrittzahl:Extended;
    Ende:Boolean;
    St:string;
begin
  Ende:=false;
  Wiederholung:=0;
  Steine:=0;
  Schrittzahl:=0;
  BestimmeMinimum(Genauigkeit);
  LadeKnotenAnzahlmitMinimum;
  SummiereAnzahlstat(Gesamtzahl);
  BestimmeWahrscheinlichkeit(Gesamtzahl);
  ErzeugeAusgabe(St);
  if Demo then Ausgabe2.Caption:=St;
  Ausgabe2.Refresh;



  SummiereAnzahlstat(Gesamtzahl);
  BestimmeWahrscheinlichkeit(Gesamtzahl);
  ErzeugeAusgabe(St);
  Application.Processmessages;
  if Demo then Ausgabe2.Caption:=St;
  Ausgabe2.refresh;
  repeat
    if Abbruch then
    begin
      ShowMessage('Abbruch');
      exit;
    end;
    Wiederholung:=Wiederholung+1;
```

```
    if Wiederholung>1000000 then
     begin
      ShowMessage('Mehr als 1000000 Iterationen:Abbruch!Keine Exakte Lösu
      ShowMessage('Möglicherweise Wahrscheinlichkeiten der Kanten zu unge
      exit;
    end;
    if not Ende then
    begin
      LadeKnotenAnzahlmitkleinsterZahl;SpeichereVerteilung;
      SummiereAnzahlstat(Gesamtzahl);
      BestimmeWahrscheinlichkeit(Gesamtzahl);
      ErzeugeAusgabe(St);
      Application.ProcessMessages;
      ZieheSteinestat(Ausgabe1,Gesamtzahl,Flaeche);
      ErhoeheAnzahlumDelta;
      if Demo then
      begin
        Knotenwertposition:=3;H:=Knotengenauigkeit;Knotengenauigkeit:=0;
        ZeichneGraph(Flaeche);
        if Demo then Ausgabe2.Caption:=St;
        Ausgabe2.Refresh;
        Demopause;
        Knotengenauigkeit:=H;Knotenwertposition:=0;
      end;

    end
  until VorigeAnzahlgleichAnzahl(Gesamtzahl) or Ende;
  if Abbruch then exit;
  SummiereAnzahlstat(Gesamtzahl);
  BestimmeWahrscheinlichkeit(Gesamtzahl);
  ErzeugeAusgabe(St);
  if Demo then Ausgabe2.Caption:=St;
  Ausgabe2.Refresh;
  if Demo then Ausgabe1.Caption:='GesamtSteine:
  '+Realtostring(Gesamtzahl);
  Ausgabe1.Refresh;
  if Demo then Ausgabe2.Caption:=St;
  if not Demo then
  begin
    Ausgabe1.Caption:='';
    Ausgabe2.Caption:='';
  end;
  ShowMessage('Wahrscheinlichkeitsverteilung:'
  +St);
end;


procedure TMarkovstatgraph.
Markovkettestat(Flaeche:TCanvas;Ausgabe1,Ausgabe2:TLabel;
  var Sliste:TStringlist;Ob:TForm);
```

```
var Fehlerstri,Str:string;
    Index,Genauigkeit:Integer;
    Gesamtzahl:Extended;
    Eingabe:Boolean;
begin
    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));if not Eingabe then exit;
    Knotengenauigkeit:=Genauigkeit;Kantengenauigkeit:=Genauigkeit;
    until Eingabe and (Genauigkeit<6);if Fehler then begin
    if Fehler then
    begin
      ShowMessage('Zuerst Fehler beheben!');
      exit;
    end;
    if not Demo  then
    begin
      Ausgabe1.Caption:=’Berechnung läuft...’;
      Ausgabe1.Refresh;
    end;
    Gesamtzahl:=0;
    Markovstat(Ausgabe1,
    Ausgabe2,Gesamtzahl,Sliste,Flaeche,Genauigkeit);
    if Abbruch then exit;
    if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      with TMarkovknoten(Knotenliste.Knoten(Index)) do
      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;
  Ausgabe2.Caption:=’ ’;
  Ausgabe2.Refresh;
  Ausgabe1.Caption:=’ ’;
  Ausgabe1.Refresh;
end;


constructor TMinimaleKostenKante.create;
begin
  inherited create;
  Fluss_:=0;
```

```
  Kosten_:=0;
  ModKosten_:=’0';
  KostenWert_:=0;
  Schranke_:=0;
  Richtung_:=false;
  Partnerkante_:=-1;
  Ergebnis_:=’’;
  Wertlisteschreiben;
end;

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;


procedure TMinimaleKostenKante.SetzeFluss(Fl:Extended);
begin
  Fluss_:=Fl;
end;
function TMinimaleKostenKante.WelcherFluss:Extended;
```

```
begin
  WelcherFluss:=Fluss_;
end;

procedure TMinimaleKostenKante.SetzeKosten(Ko:Extended);
begin
  Kosten_:=Ko;
end;


function TMinimaleKostenKante.WelcheKosten:Extended;
begin
  WelcheKosten:=Kosten_;
end;

procedure TMinimaleKostenKante.SetzeModKosten(Mko:string);
begin
  ModKosten_:=Mko;
end;

function TMinimaleKostenKante.WelcheModKosten:string;
begin
  WelcheModKosten:=ModKosten_;
end;

procedure TMinimaleKostenKante.SetzeKostenWert(Ako:Extended);
begin
  KostenWert_:=Ako;
end;

function TMinimaleKostenKante.WelcherKostenWert:Extended;
begin
  WelcherKostenWert:=KostenWert_;
end;

procedure TMinimaleKostenKante.SetzeSchranke(Schr:Extended);
begin
  Schranke_:=Schr;
end;

function TMinimaleKostenKante.WelcheSchranke:Extended;
begin
  WelcheSchranke:=Schranke_;
end;

procedure TMinimaleKostenKante.SetzeRichtung(Ri:Boolean);
begin
  Richtung_:=Ri;
end;
function TMinimaleKostenKante.WelcheRichtung:Boolean;
```

```
begin
  WelcheRichtung:=Richtung_;
end;

procedure
TMinimaleKostenKante.SetzePartnerKante(Ka:TMinimaleKostenkante);
begin
  if Ka=nil
  then
    Partnerkante_:=-1
  else
    begin
      if (Ka.Anfangsknoten<>nil) and
       (Ka.Anfangsknoten.Graph<>nil) and
       (not Ka.Anfangsknoten.Graph.Kantenliste.Leer)
        then
      Partnerkante_:=Ka.Anfangsknoten.Graph.Kantenliste.Position(Ka)
        else
        Partnerkante_:=-1;
    end;
end;

function  TMinimaleKostenkante.
WelchePartnerkante:TMinimalekostenkante;
begin
  if (self.Anfangsknoten<>nil) and
     (self.Anfangsknoten.Graph<>nil) and (not
     self.Anfangsknoten.Graph.Kantenliste.Leer)
      and (Partnerkante_>=0)
      and   (Partnerkante_<=Self.Anfangsknoten.
      Graph.Kantenliste.Anzahl-1)
    then
      WelchePartnerkante:=TMinimaleKostenKante(self.Anfangsknoten.Graph.
     Kantenliste.Kante(Partnerkante_))
    else
     WelchePartnerkante:=nil;
end;


procedure TMinimaleKostenkante.SetzeKantenindex(I:Integer);
begin
  Partnerkante_:=I;
end;


function TMinimaleKostenkante.WelcherKantenindex:Integer;
begin
  WelcherKantenindex:=Partnerkante_;
end;
procedure TMinimaleKostenkante.SetzeErgebnis(Erg:string);
```

```
begin
  Ergebnis_:=Erg;
end;


function TMinimaleKostenKante.WelchesErgebnis:string;
begin
  WelchesErgebnis:=Ergebnis_;
end;


constructor TMinimaleKostengraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TInhaltsknoten;
  InhaltsKanteclass:=TMinimaleKostenKante;
  Gesamtfluss_:=0;
  VorgegebenerFluss_:=0;
  Ganzzahlig_:=false;
end;


function TMinimaleKostenGraph.Wertlisteschreiben:TStringlist;
begin
  inherited Wertlisteschreiben;
  Wertliste.Add(Realtostring(Gesamtfluss));
  Wertliste.Add(Realtostring(VorgegebenerFluss));
  if Ganzzahlig
  then
    Wertliste.Add('true')
  else
    Wertliste.Add('false');
  Wertlisteschreiben:=Wertliste;
end;

procedure TMinimaleKostenGraph.Wertlistelesen;
begin
  inherited Wertlistelesen;
  Gesamtfluss:=StringtoReal(Wertliste.Strings[Wertliste.Count-
  3]);
 VorgegebenerFluss:=StringtoReal(Wertliste.Strings[Wertliste.Count-
  2]);
  if Wertliste.Strings[Wertliste.Count-1]='true'
  then
    Ganzzahlig:=true
  else
    Ganzzahlig:=false;
end;

procedure TMinimaleKostengraph.SetzeGesamtfluss(Gfl:Extended);
```

```
begin
  Gesamtfluss_:=Gfl;
end;

function TMinimaleKostenGraph.WelcherGesamtfluss:Extended;
begin
  WelcherGesamtfluss:=Gesamtfluss_;
end;

procedure TMinimaleKostengraph.
SetzeVorgegebenenFluss(Vfl:Extended);
begin
  VorgegebenerFluss_:=Vfl;
end;

function TMinimaleKostenGraph.WelcherVorgegebeneFluss:Extended;
begin
  WelcherVorgegebeneFluss:=VorgegebenerFluss_;
end;


procedure TMinimaleKostenGraph.SetzeGanzzahlig(Gz:Boolean);
begin
  Ganzzahlig_:=Gz;
end;

function TMinimaleKostenGraph.WelcherWertganzzahlig:Boolean;
begin
  WelcherWertganzzahlig:=Ganzzahlig_;
end;


procedure TMinimalekostengraph.SpeichereSchranken;
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.Schranke:=StringtoReal(Ka.Wert);
    end;
end;

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;

procedure TMinimaleKostenGraph.SpeichereKosten;
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.KostenWert:=Ka.Kosten;
    end;
end;


procedure TMinimaleKostenGraph.LadeKosten;
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:=Ka.KostenWert;
    end;
end;

procedure TMinimaleKostengraph.InitialisiereKanten;
var Index:Integer;
begin
  if not self.Kantenliste.Leer then
    for Index:=0 to self.Kantenliste.Anzahl-1 do
    begin
     TMinimaleKostenKante(self.Kantenliste.Kante(Index)).Fluss:=0;
        TMinimaleKostenKante(self.Kantenliste.Kante(Index)).Partnerkante:=nil;
        TMinimaleKostenKante(self.Kantenliste.Kante(Index)).ModKosten:='0';
        TMinimaleKostenKante(self.Kantenliste.Kante(Index)).KostenWert:=0;
    end;
end;

procedure TMinimaleKostenGraph.
SetzeVorwaertsKantenRichtungaufvor;
var Index:Integer;
begin
  if not self.Kantenliste.Leer then
   for Index:=0 to self.Kantenliste.Anzahl-1 do
```

```
      TMinimaleKostenKante(self.Kantenliste.Kante(Index)).Richtung:=false;
end;

procedure TMinimaleKostengraph.ErzeugeRueckkanten;
var Index,Weite:Integer;
    HilfsKantenliste:Tkantenliste;
    Ka,Kr:TMinimaleKostenKante;
begin
  Hilfskantenliste:=TKantenliste.Create;
  if not self.Kantenliste.Leer then
    for Index:=0 to self.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;
       self.FuegeKanteein(TInhaltsknoten(Ka.Endknoten),
       TInhaltsknoten(Ka.Anfangsknoten),true,Kr);
      end;
    end;
  Hilfskantenliste.Free;
  Hilfskantenliste:=nil;
end;

procedure TMinimaleKostengraph.EliminiereRueckkanten;
var Index:Integer;
begin
  Index:=0;
  if not self.Kantenliste.Leer then
    while Index<=self.Kantenliste.Anzahl-1 do
    begin
    if TMinimaleKostenkante(self.Kantenliste.Kante(Index)).
    Richtung=true
     then
     begin
     self.Kanteloeschen(self.Kantenliste.Kante(Index));
      Index:=Index-1;
```

```
     end;
     Index:=Index+1;
    end;
end;

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
      ShowMessage('Fehler:Partnerkante existiert nicht!');
    if Ka.Partnerkante.Fluss=0
    then
      MinimaleKostenKantenWert:=1E32
    else
      MinimalekostenKantenwert:=-(Ka.Partnerkante).Kosten;
  end;
end;

procedure TMinimaleKostenGraph.ErzeugeModKosten;
var Index:Integer;
Ka:TMinimaleKostenkante;
```

```
begin
  if not self.Kantenliste.Leer then
    for Index:=0 to self.Kantenliste.Anzahl-1 do
    begin
      Ka:=TMinimaleKostenkante(self.Kantenliste.Kante(Index));
      if MinimaleKostenKantenwert(Ka)<1E8
      then
        Ka.ModKosten:=
       RundeZahltostring(MinimaleKostenKantenWert(Ka),Knotengenauigkeit)
      else
        Ka.ModKosten:='i';
    end;
end;


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
  Application.ProcessMessages;
  if Abbruch then exit;
  MinimalerPfadgraph:=
  TMinimaleKostengraph(BestimmeMinimalenPfad(Quelle,
  Senke,MinimaleKostenKantenWert));
```

```
  if Demo then
  begin
    MinimalerPfadgraph.FaerbeGraph(clred,psdot);
    MinimalerPfadgraph.ZeichneGraph(Flaeche);
    Demopause;
    MinimalerPfadGraph.FaerbeGraph(clblack,pssolid);
    MinimalerPfadGraph.zeichneGraph(Flaeche);
  end;
  if (MinimalerPfadgraph.Kantenliste.Leer)
  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);
    end;
  if Gesamtfluss+MinflussSchranke>VorgegebenerFluss then
    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);
    end;
  GesamtFluss:=GesamtFluss+Differenz;
  Endproc:
  SucheMinimalenWegundbestimmeneuenFluss:=Weggefunden;
end;

procedure TMinimaleKostengraph.EingabeVorgegebenerFluss;
var Str:string;
begin
  repeat
    Str:=Inputbox('Eingabe vorgebener Fluss: ','Eingabe Fluss
    durch den Quellknoten','0');
    if (not StringistRealZahl(Str)) or
    ((StringistRealZahl(Str) and
   (Abs(StringtoReal(Str))<1.0E30)))
    then
```

```
      self.VorgegebenerFluss:=StringtoReal(Str)
    else
    begin
     ShowMessage('Fehler! Eingabe nicht im zulässigen numeri
      schen Bereich!');
      Str:='';
    end;
  until Str<>'';
end;


procedure TMinimaleKostenGraph.BestimmeFlussKostenfuerKanten(Var
Sliste:Tstringlist);
var Index:Integer;
    Ka:TMinimaleKostenKante;
begin
  if not self.Kantenliste.Leer then
    for Index:=0 to self.Kantenliste.Anzahl-1 do
    begin
      Ka:=TMinimaleKostenkante(self.Kantenliste.Kante(Index));
      if Ka.Richtung=false then
      Ka.KostenWert:=Ka.Fluss*Ka.Kosten;
      Sliste.Add('Kante:
      '+TInhaltsknoten(Ka.Anfangsknoten).Wert+' '
      +TInhaltsknoten(Ka.Endknoten).Wert+' Fluss: '+
      RundeZahltoString(Ka.Fluss,Kantengenauigkeit)
       +' Flusskosten:
      '+RundeZahltostring(Ka.Kostenwert,Kantengenauigkeit)
        +' Schranke: '+
      RundeZahltoString(Ka.Schranke,Kantengenauigkeit)+
        ' Kosten:
      '+RundeZahltoString(Ka.Kosten,Kantengenauigkeit));
    end;
end;

function TMinimaleKostenGraph.GesamtKosten:Extended;
var GKosten:Extended;
    Index:Integer;
    Ka:TMinimaleKostenkante;
begin
  GKosten:=0;
  if not self.Kantenliste.Leer then
    for Index:=0 to self.Kantenliste.Anzahl-1 do
    begin
     Ka:=TMinimaleKostenkante(self.Kantenliste.Kante(Index));
     if Ka.Richtung=false then
     Gkosten:=Gkosten+Ka.Fluss*Ka.Kosten;
    end;
  GesamtKosten:=GKosten;
end;
```

```
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;
  if not self.Kantenliste.Leer then
    for Zaehl:=0 to self.Kantenliste.Anzahl-1 do
      InverseKosten(TMinimaleKostenkante(Self.Kantenliste.Kante(Zaehl)));
end;



function TMinimalekostengraph.BestimmeQuelleundSenke(var
Quelle,Senke:
  TInhaltsknoten;Flaeche:TCanvas;
  Anzeigen:Boolean):Boolean;
label Endproc;
var Zaehle,Index:Integer;
    Gefunden:Boolean;
begin
  Gefunden:=true;
  Zaehle:=0;
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      if Knotenliste.Knoten(Index).EingehendeKantenliste.Leer
```

```
then
  begin
    Quelle:=TInhaltsknoten(Knotenliste.Knoten(Index));
     Zaehle:=Zaehle+1;
   end;
  if Zaehle<>1 then
  begin
    if Anzeigen then ShowMessage('mehrere Anfangsknoten');
    Gefunden:=false;
    Quelle:=nil;
    Senke:=nil;
    goto Endproc;
  end;
  if Gefunden then
  begin
    TInhaltsknoten(Quelle).Farbe:=clblue;
    TInhaltsknoten(Quelle).Stil:=psdashdot;
    ZeichneGraph(Flaeche);
    if Anzeigen then ShowMessage('Quelle: '+Quelle.Wert);
  end;
  Zaehle:=0;
  if not Knotenliste.Leer then
    for Index:=0 to Knotenliste.Anzahl-1 do
      if Knotenliste.Knoten(Index).AusgehendeKantenliste.Leer
        then
        begin
        Senke:=TInhaltsknoten(Knotenliste.Knoten(Index));
         Zaehle:=Zaehle+1;
      end;
  if Zaehle<>1 then
  begin
    Gefunden:=false;
    if Anzeigen then ShowMessage('Mehrere Endknoten');
    Quelle:=nil;
    Senke:=nil;
    goto Endproc;
  end;
  TInhaltsknoten(Senke).Farbe:=clgreen;
  TInhaltsknoten(Senke).Stil:=psdash;
  ZeichneGraph(Flaeche);
  if Anzeigen then ShowMessage('Senke: '+Senke.Wert);
  Endproc:
  BestimmeQuelleundSenke:=Gefunden;
end;



function TMinimaleKostengraph.
ErzeugeQuellenundSenkenknotensowieKanten(VarQuelle,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;
  Quelle.X:=10;
  Quelle.Y:=10;
  Quelle.Wert:='+';
  Senke:=TInhaltsknoten.Create;
  Senke.X:=Screen.Width-10;
  Senke.Y:=10;
  Senke.Wert:='-';
  FuegeKnotenein(Quelle);
  FuegeKnotenein(Senke);
  FlussQuellenzaehler:=0;
  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)
     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
                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 zulässi
             gen numerischen Bereich!');
              Eingabe:=false;
            end;
        FlussQuellenzaehler:=FlussQuellenzaehler+Ka.Schranke;
           if not Eingabe then
            begin
           ShowMessage('Wiederholung der Eingabe der Quel
             len-Schranken!');
            FlussQuellenzaehler:=0;
             goto Marke1;
            end;
          end;
      'o':begin
            Ka.Schranke:=1;
         FlussQuellenzaehler:=Flussquellenzaehler+1;
          end;
      'b':begin
          Ka.Schranke:=abs(Kno.Grad(true));
             Flussquellenzaehler:=FlussQuellenzaehler+abs(Kno.Grad(true));
          end;
    end;
    FuegeKanteein(Quelle,Kno,true,Ka);
  end;
if (((Art='t') or(Art='o')) and
Kno.AusgehendeKantenliste.Leer)
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 nume
            rischen 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
  begin
    Flussok:=false;
    ShowMessage('Fluss durch Quelle ist ungleich Fluss durch
     Senke');
  end;
  ErzeugeQuellenundSenkenknotensowieKanten:=FlussOk;
end;


procedure TMinimalekostengraph.
BestimmeQuelleSenkesowievorgegebenenFluss(var Quelle,Senke:
TInhaltsknoten;Ausgabe1:TLabel;Flaeche:TCanvas;Art:char);
var FlussOk:Boolean;
    Fluss:Extended;
begin
  Flussok:=true;
   case Art of
    'm':begin
```

```
          Ausgabe1.Caption:=’Quelle und Senke bestimmen!’;
         BestimmeQuelleundSenke(Quelle,Senke,Flaeche,false);
          Ausgabe1.Caption:=’Vorgabe Fluss’;
           EingabeVorgegebenerFluss;
        end;
       ‘t’:begin
          Ausgabe1.Caption:=’Quelle und Senke erzeugen’;
          Flussok:= ErzeugeQuellenundSenkenknotensowieKanten
           (Quelle,Senke,Fluss,’t’);
            if not FlussOK then
             begin
             ShowMessage(‘Der Fluss durch die Quellenknoten
               ist’+chr(13)
             +’ungleich dem Fluss durch die Senkenknoten!’);
             end;
           VorgegebenerFluss:=Fluss;
           end;
       ‘o’:begin
          Ausgabe1.Caption:=’Quelle und Senke erzeugen’;
            Flussok:= ErzeugeQuellenundSenkenknotensowieKanten
             (Quelle,Senke,Fluss,’o’);
              if not FlussOK then
            ShowMessage(‘Es entstehen isolierte Knoten.’);
             VorgegebenerFluss:=Fluss;
           end;
        ‘b’:begin
          Ausgabe1.Caption:=’Quelle und Senke erzeugen’;
          Flussok:= ErzeugeQuellenundSenkenknotensowieKanten
           (Quelle,Senke,Fluss,’b’);
           if not FlussOK then ShowMessage(‘Fehler:Fluss durch
             Quellen ist
           ungleich Fluss durch Senken!’);
           VorgegebenerFluss:=Fluss;
        end;
    end;
end;


procedure TMinimaleKostengraph.SetzeSchrankenMax;
var Index:Integer;
    Ka:TMinimaleKostenkante;
begin
  if not Kantenliste.Leer then
    for Index:=0 to self.Kantenliste.Anzahl-1 do
    begin
      Ka:=TMinimalekostenkante(self.Kantenliste.Kante(Index));
      Ka.Wert:=’1E32';
    end;
end;
```

```
procedure TMinimaleKostengraph.
LoescheQuellenundSenkenknoten(Quelle,Senke:TInhaltsknoten);
begin
  Knotenloeschen(Quelle);
  Knotenloeschen(Senke);
end;

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;


procedure TMinimaleKostengraph.LoescheNichtMatchKanten;
var Index:Integer;
begin
  Index:=0;
  if not self.Kantenliste.Leer then
    while Index<=self.Kantenliste.Anzahl-1 do
    begin
      if TMinimaleKostenkante(self.Kantenliste.Kante(Index)).
        Fluss<>1
        then
        begin
```

```
        self.Kanteloeschen(self.Kantenliste.Kante(Index));
        Index:=Index-1;
      end;
      Index:=Index+1;
    end;
end;

procedure TMinimalekostengraph.
MinimaleKosten(Flaeche:TCanvas;var G:TInhaltsgraph;
var Oberflaeche:TForm;Ausgabe1:TLabel;Maximal:Boolean;var
Quelle,Senke:TInhaltsknoten;
Art:char;var Sliste:TStringlist);
label Endproc,Wiederholung;
var Gesamtkosten:Extended;
begin
  InitialisiereKanten;
  Kantenwertposition:=1;
  if Demo then LoescheBild(G,Oberflaeche);
  if Demo then self.ZeichneGraph(Flaeche);
  if Demo then Ausgabe1.Caption:='Initialisiere Fluss';
  Demopause;
  SetzeVorwaertsKantenrichtungaufVor;
  Kantenwertposition:=6;
  if Demo then LoescheBild(G,Oberflaeche);
  if Demo then self.ZeichneGraph(Flaeche);
  if Demo then Ausgabe1.Caption:='Kantenrichtung setzen';
  Demopause;
  Kantenwertposition:=0;
  SpeichereSchranken;
  Kantenwertposition:=5;
  LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  if Demo then Ausgabe1.Caption:='Speichere Schranken';
  Demopause;
  Wiederholung:
  BestimmeQuelleSenkesowievorgegebenenFluss(Quelle,Senke,Ausgabe1,Flaeche,Art);
  Kantenwertposition:=2;
  LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  if Demo then Ausgabe1.Caption:='Kanten-Kosten';
  Demopause;
  Ausgabe1.Caption:='Berechnung läuft...';
  LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  Demopause;
  Kantenwertposition:=5;
  LoescheBild(G,Oberflaeche);
  ZeichneGraph(Flaeche);
  if Demo then Ausgabe1.Caption:='Schranken';
  Demopause;
```

```
if Maximal then
begin
  SpeichereKosten;
  BildeinverseKosten;
  Kantenwertposition:=2;
  if Demo then LoescheBild(G,Oberflaeche);
  if Demo then self.ZeichneGraph(Flaeche);
  if Demo then Ausgabe1.Caption:=’Invertierte Kanten-Kosten’;
  Demopause;
end;
Kantenwertposition:=0;
ErzeugeRueckkanten;
Kantenwertposition:=6;
if Demo then LoescheBild(G,Oberflaeche);
if Demo then Ausgabe1.Caption:=’Erzeuge Rueckkanten’;
if Demo then self.ZeichneGraph(Flaeche);
Demopause;
Kantenwertposition:=0;
if Demo then LoescheBild(G,Oberflaeche);
if Demo then self.ZeichneGraph(Flaeche);
Demopause;
repeat
  Application.Processmessages;
  if Abbruch then goto Endproc;
  ErzeugeModKosten;
  Kantenwertposition:=3;
  if Demo then LoescheBild(G,Oberflaeche);
  if Demo then self.ZeichneGraph(Flaeche);
  if self.Demo then Ausgabe1.Caption:=’Erzeuge ModKosten’;
  Demopause;
  if not
 SucheminimalenWegundbestimmeneuenFluss(Quelle,Senke,Flaeche)
  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;
  self.Kantenwertposition:=1;
  if self.Demo then LoescheBild(G,Oberflaeche);
  if self.Demo then self.ZeichneGraph(Flaeche);
  if self.Demo then Ausgabe1.Caption:=’Erzeuge neuen Fluss’;
  Demopause;
until self.VorgegebenerFluss<=self.Gesamtfluss;
Endproc:
EliminiereRueckkanten;
Kantenwertposition:=1;
```

```
if Demo then LoescheBild(G,Oberflaeche);
if Demo then self.ZeichneGraph(Flaeche);
if Demo then Ausgabe1.Caption:=’Eliminiere Rueckkanten ‘;
Demopause;
if Maximal then LadeKosten;
If Art in [‘t’,’b’,’o’] then
  LoescheQuellenundSenkenknoten(Quelle,Senke);
BestimmeFlussKostenfuerKanten(Sliste);
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));
Kantenwertposition:=4;
LoescheBild(G,Oberflaeche);
ZeichneGraph(Flaeche);
Ausgabe1.Caption:=’Kosten pro Kante’;
Demopause;
Gesamtkosten:=self.Gesamtkosten;
if not Abbruch then
if Art<>’b’
then
  ShowMessage(‘Gesamtkosten:
  ‘+RundeZahltoString(Gesamtkosten,Kantengenauigkeit))
else
  ShowMessage(‘Unproduktive Weglängen:
  ‘+RundeZahltoString(Gesamtkosten,Kantengenauigkeit));
Kantenwertposition:=1;
LoescheBild(G,Oberflaeche);
ZeichneGraph(Flaeche);
Ausgabe1.Caption:=’Fluss’;
Demopause;
if not Abbruch then
  ShowMessage(‘Vorgegebener Fluss:’+
  RundeZahltostring(self.VorgegebenerFluss,Kantengenauigkeit));
if not Abbruch then
  ShowMessage(‘Erreichter Fluss durch Quelle oder Senke: ‘+
  RundeZahltostring(self.Gesamtfluss,Kantengenauigkeit));
if  Art in [‘m’,’t’] then
 if MessageDlg(‘Wiederholung mit denselben
 Kosten?’,mtConfirmation,[mbYes,mbNo],0)=mryes
  then
  begin
   InitialisiereKanten;
   SetzeVorwaertsKantenrichtungaufVor;
   Kantenwertposition:=5;
   Gesamtfluss:=0;
   goto Wiederholung;
```

```
    end;
end;

end.

Unit UKnoten:

unit UKnoten;
{$F+}

interface

uses

  UList,UInhGrph,UGraph,UKante,UAusgabe,
  StdCtrls, Controls, ExtCtrls, Dialogs,
  Menus, Classes,
  SysUtils, WinTypes, WinProcs, Messages,Graphics,
  Forms,ClipBrd,Printers, Buttons;

type
  TKnotenform = class(TForm)
    PaintBox: TPaintBox;
    Panel: TPanel;
    Button: TButton;
    ListBox: TListBox;
    Image: TImage;
    Ausgabe1: TLabel;
    Ausgabe2: TLabel;
    Eingabe: TEdit;
    OpenDialog: TOpenDialog;
    SaveDialog: TSaveDialog;
    PrintDialog: TPrintDialog;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    BitBtn7: TBitBtn;
    BitBtn8: TBitBtn;
    BitBtn9: TBitBtn;
    BitBtn10: TBitBtn;
    BitBtn11: TBitBtn;
    MainMenu: TMainMenu;
    Datei: TMenuItem;
    NeueKnoten: TMenuItem;
    Graphladen: TMenuItem;
    Graphhinzufgen: TMenuItem;
    Graphspeichern: TMenuItem;
```

```
Graphspeichernunter: TMenuItem;
Graphdrucken: TMenuItem;
Quit: TMenuItem;
Bild: TMenuItem;
Ergebniszeichnen: TMenuItem;
Bildzeichnen: TMenuItem;
Bildkopieren: TMenuItem;
Bildwiederherstellen: TMenuItem;
UngerichteteKanten: TMenuItem;
Knotenradius: TMenuItem;
GenauigkeitKnoten: TMenuItem;
GenauigkeitKanten: TMenuItem;
Knoten: TMenuItem;
Knotenerzeugen: TMenuItem;
Knotenloeschen: TMenuItem;
Knoteneditieren: TMenuItem;
Knotenverschieben: TMenuItem;
Kanten: TMenuItem;
Kantenerzeugen: TMenuItem;
Kantenloeschen: TMenuItem;
Kanteeditieren: TMenuItem;
Kanteverschieben: TMenuItem;
Eigenschaften: TMenuItem;
AnzahlKantenundEcken: TMenuItem;
ParallelkantenundSchlingen: TMenuItem;
Eulerlinie: TMenuItem;
Kreise: TMenuItem;
AnzahlBruecken: TMenuItem;
Knotenanzeigen: TMenuItem;
Kantenanzeigen: TMenuItem;
Ausgabe: TMenuItem;
Ausgabefenster: TMenuItem;
Abbruch: TMenuItem;
Demo: TMenuItem;
Pausenzeit: TMenuItem;
Hilfe: TMenuItem;
Inhalt: TMenuItem;
Info: TMenuItem;
{allgemeine Methoden}
procedure Bildloeschen;
procedure StringlistnachListbox(S:TStringList;var
L:TListbox);
procedure Menuenabled(Enabled:Boolean);
procedure Fehler;
{Ereignismethoden der Komponenten}
procedure FormActivate(Sender: TObject);
procedure PaintBoxPaint(Sender: TObject);
procedure PaintBoxMouseDown(Sender: TObject; Button:
TMouseButton;Shift: TShiftState; X, Y: Integer);
procedure PaintBoxDblClick(Sender: TObject);
```

```
procedure PaintBoxMouseMove(Sender: TObject; Shift:
TShiftState; X,Y: Integer);
procedure PanelClick(Sender: TObject);
procedure PanelDblClick(Sender: TObject);
procedure PanelMouseDown(Sender: TObject; Button:
TMouseButton;Shift: TShiftState; X, Y: Integer);
procedure Ausgabe1Click(Sender: TObject);
procedure Ausgabe1DblClick(Sender: TObject);
procedure Ausgabe1MouseDown(Sender: TObject; Button:
TMouseButton;Shift: TShiftState; X, Y: Integer);
procedure Ausgabe2Click(Sender: TObject);
procedure Ausgabe2DblClick(Sender: TObject);
procedure Ausgabe2MouseDown(Sender: TObject; Button:
TMouseButton;Shift: TShiftState; X, Y: Integer);
procedure FormCloseQuery(Sender: TObject; var CanClose:
 Boolean);
{Ersatz-Ereignismethoden}
procedure KnotenerzeugenMouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure KnotenloeschenMousedown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure KnoteneditierenMouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure KnotenverschiebenaufMouseUp(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure KnotenverschiebenabMousedown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure KanteerzeugenMouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure KanteloeschenMouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure KanteeditierenMouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure KanteverschiebenaufMouseup(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure KanteverschiebenabMouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure PanelDownMouse(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
{Menu-Ereignismethoden}
procedure NeueKnotenClick(Sender: TObject);
procedure GraphladenClick(Sender: TObject);
procedure GraphhinzufgenClick(Sender: TObject);
procedure GraphspeichernClick(Sender: TObject);
procedure GraphspeichernunterClick(Sender: TObject);
procedure GraphdruckenClick(Sender: TObject);
procedure QuitClick(Sender: TObject);
procedure ErgebniszeichnenClick(Sender: TObject);
procedure BildzeichnenClick(Sender: TObject);
procedure BildkopierenClick(Sender: TObject);
```

```
  procedure BildwiederherstellenClick(Sender: TObject);
  procedure UngerichteteKantenClick(Sender: TObject);
  procedure KnotenradiusClick(Sender: TObject);
  procedure GenauigkeitKnotenClick(Sender: TObject);
  procedure GenauigkeitKantenClick(Sender: TObject);
  procedure KnotenerzeugenClick(Sender: TObject);
  procedure KnotenloeschenClick(Sender: TObject);
  procedure KnoteneditierenClick(Sender: TObject);
  procedure KnotenverschiebenClick(Sender: TObject);
  procedure KantenerzeugenClick(Sender: TObject);
  procedure KantenloeschenClick(Sender: TObject);
  procedure KanteeditierenClick(Sender: TObject);
  procedure KanteverschiebenClick(Sender: TObject);
  procedure AnzahlKantenundEckenClick(Sender: TObject);
  procedure ParallelkantenundSchlingenClick(Sender: TObject);
  procedure EulerlinieClick(Sender: TObject);
  procedure KreiseClick(Sender: TObject);
  procedure AnzahlBrueckenClick(Sender: TObject);
  procedure KnotenanzeigenClick(Sender: TObject);
  procedure KantenanzeigenClick(Sender: TObject);
  procedure AusgabefensterClick(Sender: TObject);
  procedure AbbruchClick(Sender: TObject);
  procedure DemoClick(Sender: TObject);
  procedure PausenzeitClick(Sender: TObject);
  procedure InhaltClick(Sender: TObject);
  procedure InfoClick(Sender: TObject);
  procedure InitBitBtnMenu;
  procedure BitBtn1Click(Sender: TObject);
  procedure BitBtn2Click(Sender: TObject);
  procedure BitBtn3Click(Sender: TObject);
  procedure BitBtn4Click(Sender: TObject);
  procedure BitBtn5Click(Sender: TObject);
  procedure BitBtn6Click(Sender: TObject);
  procedure BitBtn7Click(Sender: TObject);
  procedure BitBtn8Click(Sender: TObject);
  procedure BitBtn9Click(Sender: TObject);
  procedure BitBtn10Click(Sender: TObject);
  procedure BitBtn11Click(Sender: TObject);
private
  { Private-Deklarationen }
  Aktiv_:Boolean;
  Graph_:TInhaltsgraph;
  GraphH_:TInhaltsgraph;
  GraphK_:TInhaltsgraph;
  GraphZ_:TInhaltsgraph;
  function Istaktiv:Boolean;
  procedure Setzeaktiv(A:Boolean);
  function WelcherGraph:TInhaltsgraph;
  procedure SetzeGraph(Gr:TInhaltsgraph);
  function WelcherGraphH:TInhaltsgraph;
```

```
    procedure SetzeGraphH(Gr:TInhaltsgraph);
    function WelcherGraphK:TInhaltsgraph;
    procedure SetzeGraphK(Gr:TInhaltsgraph);
    function WelcherGraphZ:TInhaltsgraph;
    procedure SetzeGraphZ(Gr:TInhaltsgraph);
    procedure WMMenuselect(var Message:TWMMenuselect);message
    WM_Menuselect;
  public
    { Public-Deklarationen }
    property Aktiv:boolean read Istaktiv write setzeaktiv;
    property Graph:TInhaltsgraph read WelcherGraph write
    SetzeGraph;
    property GraphH:TInhaltsgraph read WelcherGraphH write
    SetzeGraphH;
    property GraphK:TInhaltsgraph read WelcherGraphK write
    SetzeGraphK;
    property GraphZ:TInhaltsgraph read WelcherGraphZ write
    SetzeGraphZ;


  end;

var
  Knotenform: TKnotenform;


implementation

{$R *.DFM}


procedure TKnotenform.Bildloeschen;
var Oberflaeche:TKnotenform;
    Graph:TInhaltsgraph;
begin
  Graph:=self.Graph;
  Oberflaeche:=self;
  LoescheBild(Graph,TForm(Oberflaeche))
end;


procedure TKnotenform.StringlistnachListbox(S:TStringList;var
L:TListbox);
var Index:Integer;
begin
   L.Clear;
   for Index:=0 to S.Count-1 do
     L.Items.Add(S.Strings[Index]);
end;
```

```
procedure TKnotenform.Menuenabled(Enabled:Boolean);
var Index,Zaehl:Integer;
begin
  if Enabled=false
  then
  begin
    for Zaehl:=0 to Mainmenu.Items.Count-1 do
      for Index:=0 to Mainmenu.Items[Zaehl].Count-1 do
      begin
      Mainmenu.Items[Zaehl].Items[Index].Enabled:=false;
      if Mainmenu.Items[Zaehl].Items[Index].Caption='Bild kopie
        ren' then
           Mainmenu.Items[Zaehl].Items[Index].Enabled:=true;
           if (Mainmenu.Items[Zaehl].Caption='Ausgabe')or
           (Mainmenu.Items[Zaehl].Caption='Hilfe')
             then
           Mainmenu.Items[Zaehl].Items[Index].Enabled:=true;
      end;
     BitBtn1.Enabled:=false;
     BitBtn2.Enabled:=false;
     BitBtn3.Enabled:=false;
     BitBtn4.Enabled:=false;
     BitBtn5.Enabled:=false;
     BitBtn6.Enabled:=false;
     BitBtn7.Enabled:=false;
     BitBtn8.Enabled:=false;
     BitBtn9.Enabled:=false;
     BitBtn10.Enabled:=false;
     BitBtn11.Enabled:=false;
   end
   else
   begin
      for Zaehl:=0 to 6 do
        for Index:=0 to Mainmenu.Items[Zaehl].Count-1 do
          Mainmenu.Items[Zaehl].Items[Index].Enabled:=true;
      BitBtn1.Enabled:=true;
      BitBtn2.Enabled:=true;
      BitBtn3.Enabled:=true;
      BitBtn4.Enabled:=true;
      BitBtn5.Enabled:=true;
      BitBtn6.Enabled:=true;
      BitBtn7.Enabled:=true;
      BitBtn8.Enabled:=true;
      BitBtn9.Enabled:=true;
      BitBtn10.Enabled:=true;
      BitBtn11.Enabled:=true;
    end;
    if (Caption='Einzelschrittmodus')and Enabled then
    begin
      Caption:='Knotenform';
```

```
      Graph.Demo:=false;
      Demo.Caption:='Demo an';
    end;
end;

procedure TKnotenform.Fehler;
begin
  Menuenabled(true);
  ShowMessage(‘Fehler’);
end;

function TKnotenform.Istaktiv:Boolean;
begin
  Istaktiv:=Aktiv_;
end;

procedure TKnotenform.Setzeaktiv(A:Boolean);
begin
  Aktiv_:=A;
end;

function TKnotenform.WelcherGraph:TInhaltsgraph;
begin
  WelcherGraph:=Graph_
end;

procedure TKnotenform.SetzeGraph(Gr:TInhaltsgraph);
begin
  Graph_:=Gr;
end;

function TKnotenform.WelcherGraphH:TInhaltsgraph;
begin
  WelcherGraphH:=GraphH_
end;

procedure TKnotenform.SetzeGraphH(Gr:TInhaltsgraph);
begin
  GraphH_:=Gr;
end;

function TKnotenform.WelcherGraphK:TInhaltsgraph;
begin
  WelcherGraphK:=GraphK_
end;

procedure TKnotenform.SetzeGraphK(Gr:TInhaltsgraph);
begin
  GraphK_:=Gr;
end;
```

```
function TKnotenform.WelcherGraphZ:TInhaltsgraph;
begin
  WelcherGraphZ:=GraphZ_
end;

procedure TKnotenform.SetzeGraphZ(Gr:TInhaltsgraph);
begin
  GraphZ_:=Gr;
end;

procedure TKnotenform.WMMenuselect(var Message:TWMMenuselect);
begin
  if
  (Message.IdItem=Knotenerzeugen.Command)or(Message.Iditem=Kantenloeschen.Command)
  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);
  if Message.IDitem <> 0 then
    if (not (Message.IDitem in [1..9])) 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.Visible:=false;
          Listbox.Top:=0;
          Listbox.Height:=0;
          Aktiv:=true;
          GraphZ:=Graph;
         if (GraphH<>nil) and (GraphH<>Graph) then
          if (Message.IDitem>=Knotenerzeugen.Command)
             and(Message.IDitem<>Inhalt.Command)and
             (Message.IDitem<>Bildkopieren.Command)and
             (Message.IDitem<>Info.Command)
```

```
          then
          begin
            GraphH.Freeall;
            GraphH:=nil;Aktiv:=true;
            Listbox.Clear;
          end;
         if GraphH<>nil then
          begin
           GraphH.Zustand:=false;
          GraphH.letzterMausklickknoten:=nil;
           end;
         Ausgabeloeschen(false);
      end;
      if (Message.IDitem<> Ausgabefenster.Command) and
         (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;

procedure TKnotenform.FormActivate(Sender: TObject);
begin
  Kantenform:=TKantenform.Create(self);
  with Kantenform do
  begin
    Width:=Round(Maximum(Weite.Left+Weite.Width,
    Maximum(Inhalt.Left+Inhalt.Width,
    Maximum(Abbruch.Left+Abbruch.Width,
    Maximum(CheckAusgehend.Left+CheckAusgehend.Width,
    Maximum(CheckEingehend.Width+CheckEingehend.Left,
     Maximum(CheckReal.Width+CheckReal.Left,CheckInteger.Width+CheckInteger.left))))))*1.2);
    Height:=Round((Abbruch.Top+Abbruch.Height)*1.2);
  end;
  Graph:=TInhaltsgraph.Create;
  GraphZ:=Graph;
  GraphK:=TInhaltsgraph.Create;
  Aktiv:=true;
  Application.HelpFile:=’Project.hlp’;
  Application.Icon.LoadFromFile(‘GRAPH.ICO’);
  if Paramcount=1 then
  begin
    Graph.Dateiname:=ParamStr(1);
    Bildloeschen;
```

```
    Graph.LadeGraph(Graph.Dateiname);
    Graph.ZeichneGraph(Paintbox.Canvas);
  end;
  Listbox.Visible:=false;
  Listbox.Top:=0;
  Listbox.Height:=0;
  Listbox.Clear;
  Listbox.Width:=Screen.Width-20;
  Kantenform.Kanteaus:=true;
  Kantenform.Kanteein:=false;
end;

procedure TKnotenform.PaintBoxPaint(Sender: TObject);
begin
  if Graph.Liniendicke =1 then Paintbox.Canvas.Font.Height:=15;
  if Graph.Liniendicke=2 then Paintbox.Canvas.Font.Height:=20;
  if Graph.Liniendicke=3 then Paintbox.Canvas.Font.Height:=24;
  if Graph.Liniendicke=1 then Paintbox.Canvas.Font.Style:=[];
  if Graph.Liniendicke=2 then
    Paintbox.Canvas.Font.Style:=[fsbold];
  if Graph.Liniendicke=3 then
   Paintbox.Canvas.Font.Style:=[fsbold];
  if Graph.Radius>29
  then
   Paintbox.Canvas.Font.Height:=Paintbox.Canvas.Font.Height+4;
  if Graph.Radius>59
  then
   Paintbox.Canvas.Font.Height:=Paintbox.Canvas.Font.Height+4;
  if Graph.Radius>78
  then
    Paintbox.Canvas.Font.Height:=Paintbox.Canvas.Font.Height+4;
  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;


procedure TKnotenform.PaintBoxMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var Kn:TInhaltsknoten;
begin
  try
    if Button=mbleft then
```

```
    begin
     Graph.Knoteninhaltzeigen(X,Y);
     if Graph.FindezuKoordinatendenKnoten(X,Y,Kn)=false then
       Graph.Kanteninhaltzeigen(X,Y);
      if Graph.FindezuKoordinatendenKnoten(X,Y,Kn)=true then
        Graph.LetzterMausklickknoten:=Kn;
    end;
    if Button=mbright then
    begin
      GraphK.Freeall;
       GraphK:=Graph.InhaltskopiedesGraphen(TInhaltsgraph,TInhaltsknoten,TInhaltskante,false);
      Ausgabe1.Caption:='1. Knoten wählen!';
       Ausgabe1.Refresh;
      Paintbox.OnMouseDown:=KanteerzeugenMouseDown;
      Paintbox.OnDblClick:=nil;
      Paintbox.OnMousemove:=nil;
      Graph.Graphistgespeichert:=false;
    end;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
  except
    Fehler;
  end;
end;

procedure TKnotenform.PaintBoxDblClick(Sender: TObject);
begin
  try
    Listbox.Visible:=false;
    Listbox.Top:=0;
    Listbox.Height:=0;
    Listbox.Clear;
    GraphK.Freeall;
     GraphK:=Graph.InhaltskopiedesGraphen(TInhaltsgraph,TInhaltsknoten,TInhaltskante,false);
    Paintbox.OnMouseDown:=KnotenerzeugenMouseDown;
    Graph.Graphistgespeichert:=false;
except
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenform.PaintBoxMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
  try
    if (ssleft in shift) then
    begin
      if Graph.Knotenverschieben(X,Y) or
Graph.Kanteverschieben(X,Y)
      then
```

```
        Bildloeschen;
      Graph.ZeichneGraph(Paintbox.Canvas);
    end;
  except
    Fehler;
  end;
end;

procedure TKnotenform.PanelClick(Sender: TObject);
begin
  try
    if Listbox.Items.Count>0 then
    begin
      Listbox.Top:=Panel.Top-20;
      Listbox.Width:=Knotenform.Width-5;
      Listbox.Height:=20;
      Listbox.Visible:=true;
    end;
  except
    Fehler;
  end;
end;

procedure TKnotenform.PanelDblClick(Sender: TObject);
begin
  try
    if Listbox.Items.Count>0
    then
    begin
      Listbox.Width:=Screen.Width;
      Listbox.Visible:=true;
      Listbox.Top:=Knotenform.Height DIV 3;
      Listbox.Height:=Panel.Top-Listbox.Top+5;
    end;
  except
    Fehler;
  end;
end;

procedure TKnotenform.PanelMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  try
    if Button =mbright then
    begin
      Listbox.Visible:=false;
    end;
    if (ssctrl in Shift ) and (Button=mbleft)
    then
      if Graph.LetzterMausklickknoten <>nil
```

```
      then
        ShowMessage('Letzter mit der Maus angeklickter Knoten:
         '+chr(13)+
          'Inhalt:
        '+Graph.LetzterMausklickknoten.Wert+chr(13)+'x= '+
        Integertostring(Graph.letzterMausklickknoten.X)+' y= '+
        Integertostring(Graph.letzterMausklickknoten.Y));
      if (ssshift in Shift ) and (Button=mbleft)
       then
       begin
        Graph.Pausenzeit:=-1;
        if GraphH<>nil then GraphH.Pausenzeit:=Graph.Pausenzeit;
        Caption:='Einzelschrittmodus';
         Graph.Demo:=true;
        if GraphH<>nil then GraphH.Demo:=true;
       end;
      if (ssalt in Shift ) and (Button=mbleft)
       then
       begin
        Graph.Pausenzeit:=1;
        if GraphH<>nil then GraphH.Pausenzeit:=Graph.Pausenzeit;
       end;
       if ([ssalt,ssshift] <=Shift ) and (Button=mbleft)
       then
       begin
        Graph.Pausenzeit:=0;
        if GraphH<>nil then GraphH.Pausenzeit:=Graph.Pausenzeit;
         Graph.Demo:=false;
        if GraphH<>nil then GraphH.Demo:=false;
        Caption:='Knotenform';
       end;
  except
    Fehler;
  end;
end;


procedure TKnotenform.Ausgabe1Click(Sender: TObject);
begin
  try
    if Listbox.Items.Count>0
    then
    begin
      Listbox.Top:=Panel.Top-15;
      Listbox.Width:=Screen.Width-20;
      Listbox.Height:=25;
      Listbox.Visible:=true;
    end;
  except
    Fehler;
```

```
  end;
end;

procedure TKnotenform.Ausgabe1DblClick(Sender: TObject);
begin
  try
    if Listbox.Items.Count>0
    then
    begin
      Listbox.Width:=Screen.Width-20;
      Listbox.Visible:=true;
      Listbox.Top:=25;
      Listbox.Height:=Panel.Top-20;
    end;
  except
    Fehler;
  end;
end;

procedure TKnotenform.Ausgabe1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  try
    if Button =mbright then
    begin
      Listbox.Visible:=false;
    end;
   if (ssctrl in Shift ) and (Button=mbleft)
    then
      if Graph.LetzterMausklickknoten <>nil
      then
        ShowMessage('Letzter mit der Maus angeklickter Knoten:
          '+chr(13)+'Inhalt:
        '+Graph.LetzterMausklickknoten.Wert+chr(13)+'x= '+
        Integertostring(Graph.letzterMausklickknoten.X)+' y= '+
        Integertostring(Graph.letzterMausklickknoten.Y));
  except
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenform.Ausgabe2Click(Sender: TObject);
begin
  try
    if Listbox.Items.Count>0
    then
    begin
      Listbox.Top:=Panel.Top-18;
      Listbox.Width:=Knotenform.Width-20*Knotenform.Width DIV
        640;
```

```
      Listbox.Height:=25;
      Listbox.Visible:=true;
   end;
  except
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenform.Ausgabe2DblClick(Sender: TObject);
begin
  try
    if Listbox.Items.Count>0
    then
    begin
     Listbox.Width:=Screen.Width-20;
     Listbox.Visible:=true;
     Listbox.Top:=Knotenform.Height div 2;
     Listbox.Height:=Knotenform.Height DIV 2+5;
     ShowMessage(Inttostr(Knotenform.Height));
    end;
  except
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenform.Ausgabe2MouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  try
    if Button =mbright then
    begin
      Listbox.Visible:=false;
    end;
    if (ssctrl in Shift ) and (Button=mbleft)
    then
      if Graph.LetzterMausklickknoten <>nil
      then
        ShowMessage('Letzter mit der Maus angeklickter Knoten:
          '+chr(13)+'Inhalt:
         '+Graph.LetzterMausklickknoten.Wert+chr(13)+'x= '+
         Integertostring(Graph.letzterMausklickknoten.X)+' y= '+
         Integertostring(Graph.letzterMausklickknoten.Y));
  except
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
```

```
begin
try
    if if (GraphK<>nil) and (not Graph.Graphistgespeichert)
     then
      if MessageDlg('Momentaner Graph ist nicht gespeichet!
       Speichern?',mtConfirmation, [mbYes, mbNo], 0) = mrYes
        then GraphspeichernunterClick(Sender);
if (GraphK<>nil) and (not Graph.Demo) then if Mainmenu.Items[0].
Items[0].Enabled=true then if MessageDlg('Knotengraph: Jetzt beenden?',
         mtConfirmation, [mbYes, mbNo], 0) = mrYes
      then
      begin
        MessageDlg('Beenden von Knotengraph!', mtInformation,
          [mbOk], 0);
        Application.HelpCommand(Help_Quit,0);
         Canclose:=true;
         Graph.Pausenzeit:=0;
        if GraphH<> nil then GraphH.Pausenzeit:=0;
        if GraphH<> nil then GraphH.Abbruch:=true;
        if (GraphH<>nil) and (GraphH<>Graph) then
         GraphH.Freeall;

         GraphK.Freeall;
         GraphK:=nil;
         Kantenform.Free;
         Kantenform:=nil;
       end
       else
        CanClose:=false;
  except
    ShowMessage('Fehler');
    CanClose:=true;
  end;
end;

procedure TKnotenform.KnotenerzeugenMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  try
    Listbox.Clear;
    Graph.Knotenzeichnen(X,Y);
    Paintbox.OnMouseDown:=PaintboxMouseDown;
    Graph.ZeichneGraph(Paintbox.Canvas);
  except
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenform.KnotenloeschenMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
```

```
begin
  try
    if Button =mbleft then
    begin
      Graph.Inhaltsknotenloeschen(X,Y);
      Bildzeichnenclick(Sender);
      Graph.ZeichneGraph(Paintbox.Canvas);
    end
    else
      Paintbox.OnMouseDown:=PaintboxMouseDown;
  except
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.KnoteneditierenMouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  try
    if Button=mbleft then
    begin
      Graph.Knoteneditieren(X,Y);
      Graph.ZeichneGraph(Paintbox.Canvas);
    end
    else
      Paintbox.OnMouseDown:=PaintboxMouseDown;
  except
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.KnotenverschiebenaufMouseup(Sender:
TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  try
    Graph.LoescheKnotenbesucht;
    Ausgabe1.Caption:='';
    Paintbox.Onmouseup:=nil;
  except
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenform.KnotenverschiebenabMouseDown(Sender:
TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
label Marke;
```

```
var Index:Integer;
    Kno:TInhaltsknoten;
begin
  try
    if Button=mbleft then
    begin
      Graph.LoescheKnotenbesucht;
      if not Graph.Knotenliste.Leer then
        for Index:=0 to Graph.Knotenliste.Anzahl-1 do
        begin
          Kno:=TInhaltsknoten(Graph.Knotenliste.Knoten(Index));
          if  Graph.FindezuKoordinatendenKnoten(X,Y,Kno) then
          begin
            Kno.Besucht:=true;
            Ausgabe1.Caption:='Knoten: '+Kno.Wert;
            goto Marke;
          end
        end;
        ShowMessage('Kein Knoten!');
      marke:
    end
    else
      Paintbox.Onmousedown:=Paintboxmousedown;
  except
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenform.KanteerzeugenMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  try
    Listbox.Clear;
    if Graph.Kantezeichnen(X,Y) then
    begin
      Paintbox.OnMouseDown:=PaintboxMousedown;
      Paintbox.OnDblClick:=PaintboxDblclick;
      Paintbox.OnMousemove:=PaintboxMousemove;
      Ausgabe1.Caption:='';
      Ausgabe1.Refresh;
    end
    else
      Ausgabe1.Caption:='2. Knoten wählen!';;
    Aktiv:=true;
    Graph.ZeichneGraph(Paintbox.Canvas);
  except
    ShowMessage('Fehler');
  end;
end;
```

```
procedure TKnotenform.KanteloeschenMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  try
    if Button=mbleft then
    begin
      if Graph.Inhaltskanteloeschen(X,Y) then
        Bildloeschen;
      Graph.ZeichneGraph(Paintbox.Canvas);
    end
    else
      Paintbox.OnMouseDown:=PaintboxMouseDown;
    except
     ShowMessage('Fehler');
    end;
end;

procedure TKnotenform.KanteeditierenMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  try
    if Button=mbleft then
    begin
      if Graph.Kanteeditieren(X,Y) then
      begin
        Bildloeschen;
        Graph.ZeichneGraph(Paintbox.Canvas);
      end;
    end
    else
      Paintbox.OnMouseDown:=PaintboxMouseDown;
  except
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.KanteverschiebenaufMouseUp(Sender:
TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  try
    Graph.LoescheKantenbesucht;
    Ausgabe1.Caption:='';
    Paintbox.Onmouseup:=nil;
  except
    ShowMessage('Fehler');
  end;
end;
```

```
procedure TKnotenform.KanteverschiebenabMouseDown(Sender:
TObject;Button: TMouseButton; Shift: TShiftState; X, Y: Inte-
ger);
label Marke;
var Index:Integer;
   Ka:TInhaltskante;
begin
  try
   if Button=mbleft then
    begin
     Graph.LoescheKantenbesucht;
     if not Graph.Kantenliste.leer then
     for Index:=0 to Graph.Kantenliste.Anzahl-1 do
       begin
       Ka:=TInhaltskante(Graph.Kantenliste.Kante(Index));
       if Ka.MausklickaufKante(X,Y) then
         begin
         Ka.Besucht:=true;
        Ausgabe1.Caption:='Kante: '+Ka.Wert;
          goto Marke;
         end
      end;
      ShowMessage('Keine Kante!');
      Marke:
    end
    else
    Paintbox.OnMouseDown:=PaintboxMousedown;
  except
     ShowMessage('Fehler');
  end;
 end;

 procedure TKnotenform.PanelDownMouse(Sender: TObject;
 Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var GraphK:TInhaltsknoten;Kantenwert,Knotenwert:Integer;
begin
  try
   if Graph.FindezuKoordinatendenKnoten(X,Y,GraphK)=true
    then
      Graph.LetzterMausklickknoten:=GraphK;
    if (Button=mbleft)and (GraphH<>nil) then begin
    Knotenwertposition:=GraphH:Knotenwertposition;
    GraphH.Knoteninhaltzeigen(X,Y);
    GraphH.Knotenwertposition:=Knotenwert;end;
    if GraphH.FindezuKoordinatendenKnoten(X,Y,GraphK)=false then begin
    Kantenwert:=GraphH.Kantenwertposition;
    GraphH.Kantenwertposition:=Kantenwert;
   end;
  except
    ShowMessage('Fehler');
```

```
  end;
end;


procedure TKnotenform.NeueKnotenClick(Sender: TObject);
begin
  try
    Listbox.Clear;
    if not Graph.Graphistgespeichert
     then
      if MessageDlg('Momentaner Graph ist nicht gespeichet!
      Speichern?',mtConfirmation, [mbYes, mbNo], 0) = mrYes
        then
        GraphspeichernunterClick(Sender);
    Graph.Graphistgespeichert:=false;
    Graph.Dateiname:='';
    Graph.ImGraphKnotenundKantenloeschen;
    Graph.LetzterMausklickknoten:=nil;BitBtn11Click(Sender);
   if Aktiv then if (GraphH<>nil)and (GraphH<>Graph) then GraphH.Freeall;
    GraphH:=nil;
    Aktiv:=true;
    Graph.Knotengenauigkeit:=3;
    Graph.Kantengenauigkeit:=3;
    Bildloeschen;
  except
    Graph.LetzterMausklickknoten:=nil;
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.GraphladenClick(Sender: TObject);
begin
  try
    Listbox.Clear;
    if not Graph.Graphistgespeichert
    then
     if MessageDlg('Momentaner Graph ist nicht gespeichet! Spei
     chern?',mtConfirmation, [mbYes, mbNo], 0) = mrYes
      then
      GraphspeichernunterClick(Sender);
    Graph.Graphistgespeichert:=true;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Opendialog.Execute then
    begin
      Graph.ImGraphKnotenundKantenloeschen;
      Graph.LetzterMausklickknoten:=nil;
      Graph.Demo:=false;
```

```
      Graph.Pausenzeit:=0;
      if (GraphH<>nil) and (GraphH<>Graph) then GraphH.Freeall;
       GraphH:=nil;
      Graph.Dateiname:=Opendialog.Filename;
       Aktiv:=true;
       Bildloeschen;
      Graph.LadeGraph(Graph.Dateiname);
      Paintbox.Canvas.Pen.Width:=Graph.Liniendicke;
      if Graph.Liniendicke=0 then Graph.Liniendicke:=1;
      if Graph.Liniendicke =1 then
      Paintbox.Canvas.Font.Height:=15;
      if Graph.Liniendicke=2 then
      Paintbox.Canvas.Font.Height:=20;
      if Graph.Liniendicke=3 then
      Paintbox.Canvas.Font.Height:=24;
      if Graph.Liniendicke=1 then
      Paintbox.Canvas.Font.Style:=[];
      if Graph.Liniendicke=2 then
      Paintbox.Canvas.Font.Style:=[fsbold];
      if Graph.Liniendicke=3 then
      Paintbox.Canvas.Font.Style:=[fsbold];
      if Graph.Radius>29
      then
      Paintbox.Canvas.Font.Height:=Paintbox.Canvas.Font.Height+4;
      if Graph.Radius>59
      then
      Paintbox.Canvas.Font.Height:=Paintbox.Canvas.Font.Height+4;
      if Graph.Radius>79
      then
      Paintbox.Canvas.Font.Height:=Paintbox.Canvas.Font.Height+4;
      Graph.ZeichneGraph(Paintbox.Canvas);
    end;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.GraphhinzufgenClick(Sender: TObject);
var Dicke,Radius,Index:Integer;
begin
  try
    Radius:=Graph.Radius;
    Dicke:=Graph.Liniendicke;
    Listbox.Clear;
    if not Graph.Graphistgespeichert
    then
     if MessageDlg('Momentaner Graph ist nicht gespeichet!
     Speichern?',mtConfirmation, [mbYes, mbNo], 0) = mrYes
```

```
      then
       GraphspeichernClick(Sender);
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Opendialog.execute then
     begin
      Graph.Dateiname:=Opendialog.Filename;
       Bildloeschen;
      BitBtn11Click(Sender);
      if (GraphH<>nil) and (GraphH<>Graph) then GraphH.Freeall;
       GraphH:=nil;
      Graph.LadeGraph(Graph.Dateiname);
      Paintbox.Canvas.Pen.Width:=Dicke;
      Graph.Liniendicke:=Dicke;
      Graph.Radius:=Radius;
      if not Graph.Knotenliste.leer
        then
        for index:=0 to Graph.Knotenliste.Anzahl-1 do
       TInhaltsknoten(Graph.Knotenliste.Knoten(Index)).Radius:=Radius;
      If Graph.Liniendicke=0 then Graph.Liniendicke:=1;
      if Graph.Liniendicke =1 then
      Paintbox.Canvas.Font.Height:=12;
      if Graph.Liniendicke=2 then
      Paintbox.Canvas.Font.Height:=14;
      if Graph.Liniendicke=3 then
      Paintbox.Canvas.Font.Height:=16;
      if Graph.Liniendicke=1 then
      Paintbox.Canvas.Font.Style:=[];
      if Graph.Liniendicke=2 then
      Paintbox.Canvas.Font.Style:=[fsbold];
      if Graph.Liniendicke=3 then
      Paintbox.Canvas.Font.Style:=[fsbold];
      if Graph.Radius>29
       then
      Paintbox.Canvas.Font.Height:=Paintbox.Canvas.Font.Height+4;
      if Graph.Radius>59
       then
      Paintbox.Canvas.Font.Height:=Paintbox.Canvas.Font.Height+4;
      if Graph.Radius>79
      then
      Paintbox.Canvas.Font.Height:=Paintbox.Canvas.Font.Height+4;
      Bildloeschen;
     Graph.ZeichneGraph(Paintbox.Canvas);
    end;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;
```

```
procedure TKnotenform.GraphspeichernClick(Sender: TObject);
begin
  try
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.Dateiname<>'' then
    begin
     Graph.SpeichereGraph(Graph.Dateiname);
     ShowMessage('Graph gespeichert')
    end
    else
      Graphspeichernunterclick(Sender);
    Graph.Graphistgespeichert:=true;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;
```

```
procedure TKnotenform.GraphspeichernunterClick(Sender:
TObject);
begin
  try
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Savedialog.Filename:=Graph.Dateiname;
    if Savedialog.execute then
    begin
      Graph.Dateiname:=Savedialog.Filename;
      Graph.SpeichereGraph(Graph.Dateiname);
      ShowMessage('Graph gespeichert');
      Graph.Graphistgespeichert:=true;
    end;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;
```

```
procedure TKnotenform.GraphdruckenClick(Sender: TObject);
var L,D:TInhaltsgraph;
    Rect:TRect;
    Bitmap:TBitmap;
begin
  try
    if (MessageDlg('Ist der Drucker
    graphikfähig?',mtConfirmation, [mbYes,
    mbNo], 0) = mrYes)
```

```
then

begin
  if (GraphH<>nil) and (MessageDlg('Ergebnis
  drucken?',mtConfirmation,
  [mbYes, mbNo], 0) = mrYes)
  then
  L:=GraphH
  else
  L:=Graph;
  if PrintDialog.execute then
  begin
    Bildloeschen;
    L.ZeichneGraph(Paintbox.Canvas);
    Screen.Cursor:=Crhourglass;
    Rect.Left:=0;
    Rect.Top:=0;
    Rect.Right:=Printer.PageWidth;
    Rect.Bottom:=Trunc(Minimum((Printer.Pagewidth div
    Paintbox.Width)
    *Paintbox.Height,Printer.PageHeight));
    Bitmap:=TBitmap.Create;
    Bitmap.Width:=Paintbox.Width;
    Bitmap.Height:=Paintbox.Height;
    Image.Left:=0;
    Image.Top:=0;
    Image.Height:=Paintbox.Width;
    Image.Width:=Paintbox.Width;
    Image.Picture.Graphic:=Bitmap;
    Image.Visible:=false;
    L.ZeichneGraph(Image.Canvas);
    Printer.Begindoc;
    Printer.Canvas.StretchDraw(Rect,Image.Picture.Graphic);
    Printer.Enddoc;
    Image.Visible:=false;
    Bitmap.Free;
     Screen.Cursor:=CrDefault;
  end
end
else
begin
  ShowMessage('Empfohlene Druckerauflösung: 600 dpi');
  if PrintDialog.execute then
  begin
    L:=Graph;
    Bildloeschen;
    BitBtn11Click(Sender);
    L.ZeichneGraph(Paintbox.Canvas);
    Screen.Cursor:=Crhourglass;
    D:=L.InhaltskopiedesGraphen
    (TInhaltsgraph,TInhaltsknoten,
    TInhaltskante,false);
    Printer.Canvas.Pen.Width:=5*Graph.Liniendicke;
    Printer.Canvas.Font.Name:='Courier New';
    Paintbox.Canvas.Pen.Width:=Graph.Liniendicke;
```

```
   drucken?',mtConfirmation, [mbYes, mbNo], 0) = mrYes)
    then
      L:=GraphH
    else
      L:=Graph;
    if PrintDialog.execute then
    begin
      D:=L.InhaltskopiedesGraphen(TInhaltsgraph,TInhaltsknoten,TInhaltskante,false);
      Bildloeschen;
      L.ZeichneGraph(Paintbox.Canvas);
      Screen.Cursor:=Crhourglass;
      Printer.Canvas.Pen.Width:=4*Graph.Liniendicke;
      Printer.Canvas.Font.Name:='Courier New';
      Paintbox.Canvas.Pen.Width:=Graph.Liniendicke;
      if Graph.Liniendicke =1 then Printer.Canvas.Font.Size:=12;
      if Graph.Liniendicke=2 then Printer.Canvas.Font.Size:=16;
      if Graph.Liniendicke=3 then Printer.Canvas.Font.Size:=20;
      if Graph.Liniendicke=1 then Printer.Canvas.Font.Style:=[];
      if Graph.Liniendicke=2 then
      Printer.Canvas.Font.Style:=[fsbold];
      if Graph.Liniendicke=3 then
      Printer.Canvas.Font.Style:=[fsbold];
      if Graph.Radius>29
       then
       Printer.Canvas.Font.Height:=Printer.Canvas.Font.Size+4;
      if Graph.Radius>59
       then
        Printer.Canvas.Font.Height:=Printer.Canvas.Font.Size+4;
      if Graph.Radius>79
       then
        Printer.Canvas.Font.Height:=Printer.Canvas.Font.Size+4;
      Printer.Begindoc;
      D.ZeichneDruckGraph(Printer.Canvas,
      Printer.PageWidth div Paintbox.Width);
      Printer.Enddoc;
     Screen.Cursor:=CrDefault;
    end;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.QuitClick(Sender: TObject);
begin
  try
    Graph.Freeall;Close;
  except
    Menuenabled(true);
```

```
    ShowMessage('Fehler');
    Close;
  end;
end;

procedure TKnotenform.ErgebniszeichnenClick(Sender: TObject);
begin
  try Aktiv:=false;
    if GraphH<>nil
    then
    begin
      Aktiv:=false;
      Bildloeschen;
      GraphH.ZeichneGraph(Paintbox.Canvas);
      Paintbox.OnMouseDown:=PanelDownMouse;
      Paintbox.OnDblclick:=nil;
      Paintbox.OnMouseMove:=nil;
    end
    else
      ShowMessage('Kein letztes Bild gespeichert!');
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenform.BildzeichnenClick(Sender: TObject);
begin
  try
    Aktiv:=true;
    Paintbox.Refresh;
    Graph.ZeichneGraph(Paintbox.Canvas);
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.BildkopierenClick(Sender: TObject);
var Bitmap:TBitmap;
    L:TInhaltsgraph;

begin
  try
    if  Mainmenu.Items[0].Items[0].Enabled=true then
    begin Paintbox.OnMouseDown:=PanelDownMouse; Paintbox.OnDblClick:=nil;
       Paintbox.OnMousemove:=nil;Aktiv:=false;if (GraphH<>nil) and
        (MessageDlg('Ergebnisbild in die Zwischenablage kopie
         ren?',mtConfirmation, [mbYes, mbNo], 0) = mrYes)
```

```
      then
      begin
        L:=GraphH;
       Showmessage('Ergebnisbild in Zwischenablage kopiert!');
      Aktiv:=false;end
      else
      begin
        L:=Graph;
        showmessage('Bild in Zwischenablage kopiert!');
      Aktiv:=true;end
    end
    else
    begin
      L:=GraphH;
      Aktiv:=false;showmessage('Ergebnisbild in Zwischenablage kopiert!');
    end;
    Bildloeschen;
    L.ZeichneGraph(Paintbox.Canvas);
    Bitmap:=TBitmap.Create;
    Bitmap.Width:=700;
    Bitmap.Height:=700;
    Image.Left:=0;
    Image.Top:=0;
    Image.Height:=700;
    Image.Width:=700;
    Image.Picture.Graphic:=Bitmap;
    Image.Visible:=false;
    L.ZeichneGraph(Image.Canvas);
    Clipboard.Assign(Image.Picture);
    Image.Visible:=false;
    Bitmap.Free;
    Bitmap:=nil;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenform.BildwiederherstellenClick(Sender:
TObject);
begin
  try Aktiv:=false;
    Listbox.Clear;
    if (GraphK<>nil) and (not GraphK.leer)
    then
    begin Aktiv:=true;
      Graph.Freeall;
     Graph:=GraphK.InhaltskopiedesGraphen
     (TInhaltsgraph,TInhaltsknoten,TInhaltskante,false);
    Bildloeschen;Graph.zeichneGraph(Paintbox.Canvas); end
```

```
    else
      ShowMessage('Letzter Graph kann nicht wiederhergestellt
       werden!');


  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenform.UngerichteteKantenClick(Sender: TObject);
begin
  try Bildloeschen;Graph.ZeichneGraph(Paintbox.Canvas);
    Listbox.Clear;
    Graph:=Graph.InhaltskopiedesGraphen
    (TInhaltsgraph,TInhaltsknoten,TInhaltskante,true);
    Bildloeschen;
    Graph.zeichneGraph(Paintbox.Canvas);
    Graph.Graphistgespeichert:=false;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.KnotenradiusClick(Sender: TObject);
label Endproc;
var St:string;
    Eingabe:Boolean;
    Index:Integer;
    R,Dicke:Integer;
begin
  try Bildloeschen;Graph.ZeichneGraph(Paintbox.Canvas);
    Graph.Graphistgespeichert:=false;
    St:='15';
    repeat
      Eingabe:=InputQuery('Eingabe Knotenradius', 'Eingabe des
       Radius: ', St);
      if not Eingabe then goto Endproc;
    until St<>'';
    R:=StringtoInteger(St);
    if R<10 then goto Endproc;
    if R>100 then goto Endproc;
    if not Graph.leer then
      for Index:=0 to Graph.Knotenliste.Anzahl-1 do
       TInhaltsknoten(Graph.Knotenliste.Knoten(Index)).Radius:=R;
    if GraphH<>nil then
      if not GraphH.leer then
```

```
       for Index:=0 to GraphH.Knotenliste.Anzahl-1 do
      TInhaltsknoten(GraphH.Knotenliste.Knoten(Index)).Radius:=R;
    Graph.Radius:=R;
    if GraphH<>nil then GraphH.Radius:=R;
    St:='1';
    repeat
      Eingabe:=InputQuery('Eingabe Liniendicke', 'Eingabe der
       Liniendicke (1-3): ', St);
      if not Eingabe then goto Endproc;
    until St<>'';
    Dicke:=StringtoInteger(St);
    if Dicke<1 then goto Endproc;
    if Dicke>3 then goto Endproc;
    Graph.Liniendicke:=Dicke;
    if GraphH<>nil then GraphH.Liniendicke:=Dicke;
    Paintbox.Canvas.Pen.Width:=Dicke;
    If Dicke =1 then Paintbox.Canvas.Font.Height:=15;
    if Dicke=2 then Paintbox.Canvas.Font.Height:=20;
    if Dicke=3 then Paintbox.Canvas.Font.Height:=24;
    if Dicke=1 then Paintbox.Canvas.Font.Style:=[];
    if Dicke=2 then Paintbox.Canvas.Font.Style:=[fsbold];
    if Dicke=3 then Paintbox.Canvas.Font.Style:=[fsbold];
    if Graph.Radius>29
     then
      Paintbox.Canvas.Font.Height:=Paintbox.Canvas.Font.Height+4;
    if Graph.Radius>59
    then
      Paintbox.Canvas.Font.Height:=Paintbox.Canvas.Font.Height+4;
    if Graph.Radius>79
    then
      Paintbox.Canvas.Font.Height:=Paintbox.Canvas.Font.Height+4;
    Endproc:
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.GenauigkeitKnotenClick(Sender: TObject);
label Endproc;
var St:string;
    Eingabe:Boolean;
    Index:Integer;
    Ge:Integer;
begin
  try Bildloeschen;Graph.ZeichneGraph(Paintbox.Canvas);
    Graph.Graphistgespeichert:=false;
```

```
    St:=IntegertoString(Graph.Knotengenauigkeit);
    if St='100' then St:='X';
    repeat
      Eingabe:=InputQuery('Eingabe Genauigkeit Knoten', 'Eingabe
       der Stellenzahl: ', St);
       if not Eingabe then goto Endproc;
    until St<>'';
    if St='X' then St:='100';
    Ge:=StringtoInteger(St);
    if (Ge>7) and (Ge<>100) then goto Endproc;
    Graph.Knotengenauigkeit:=ge;
    if GraphH<>nil then GraphH.Knotengenauigkeit:=Ge;
    Endproc:
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenform.GenauigkeitKantenClick(Sender: TObject);
label Endproc;
var St:string;
    Eingabe:Boolean;
    Index:Integer;
    Ge:Integer;
begin
  try Bildloeschen;Graph.ZeichneGraph(Paintbox.Canvas);
    Graph.Graphistgespeichert:=false;
    St:=Integertostring(Graph.Kantengenauigkeit);
    if St='100' then St:='x';
    repeat
      Eingabe:=InputQuery('Eingabe Genauigkeit Kanten', 'Eingabe
       der Stellenzahl: ', St);
       if not Eingabe then goto Endproc;
    until St<>'';
    if St='x' then St:='100';
    Ge:=StringtoInteger(St);
    if (Ge>7) and (Ge<>100) then goto Endproc;
    Graph.Kantengenauigkeit:=Ge;
    if GraphH<>nil then GraphH.Kantengenauigkeit:=Ge;
    Endproc:
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;
```

```
procedure TKnotenform.KnotenerzeugenClick(Sender: TObject);
begin
  try
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Paintbox.OnMouseDown:=KnotenerzeugenMouseDown;
    Graph.Graphistgespeichert:=false;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.KnotenloeschenClick(Sender: TObject);
begin
  try
    Listbox.Clear;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    ShowMessage('Knoten wählen');
    Paintbox.OnMouseDown:=KnotenloeschenMouseDown;
    Graph.Graphistgespeichert:=false;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.KnoteneditierenClick(Sender: TObject);
begin
  try
    Listbox.Clear;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Paintbox.Onmousedown:=KnoteneditierenMouseDown;
    ShowMessage('Knoten wählen');
    Graph.Graphistgespeichert:=false;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.KnotenverschiebenClick(Sender: TObject);
begin
  try
    Listbox.Clear;
```

```
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Paintbox.Onmousedown:=KnotenverschiebenabMouseDown;
    Paintbox.Onmouseup:=KnotenverschiebenaufMouseUp;
    ShowMessage('Knoten mit Maus ziehen');
    Graph.Graphistgespeichert:=false;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.KantenerzeugenClick(Sender: TObject);
begin
  try
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Kantenform.Kanteninhalt:=' ';
    Kantenform.Kantenweite:=0;
    Kantenform.KantenInteger:=false;
    Kantenform.KantenReal:=false;
    Kantenform.Kanteein:=false;
    Kantenform.Kanteaus:=true;
    Paintbox.OnMouseDown:=KanteerzeugenMouseDown;
    ShowMessage('1. und 2. Knoten nacheinander  auswählen!');
    Ausgabe1.Caption:='1.Knoten wählen!';
    Ausgabe1.Refresh;
    Graph.Graphistgespeichert:=false;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.KantenloeschenClick(Sender: TObject);
begin
  try
    Listbox.Clear;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Paintbox.OnMouseDown:=KanteloeschenMouseDown;
    ShowMessage('Kante wählen!');
    Graph.Graphistgespeichert:=false;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;
```

```
procedure TKnotenform.KanteeditierenClick(Sender: TObject);
begin
  try
    Listbox.Clear;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Paintbox.OnMouseDown:=KanteeditierenMouseDown;
    ShowMessage('Kante wählen');
    Graph.Graphistgespeichert:=false;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.KanteverschiebenClick(Sender: TObject);
var Index:Integer;
    Ka:TInhaltskante;
begin
  try
    Listbox.Clear;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Paintbox.OnMousedown:=KanteverschiebenabMouseDown;
    Paintbox.OnMouseUp:=KanteverschiebenaufMouseUp;
    ShowMessage('Kante mit Maus ziehen');
    Graph.Graphistgespeichert:=false;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.AnzahlKantenundEckenClick(Sender:
TObject);
var Strin:string;
    Gebiete:Integer;
    Komponenten:Integer;
begin
  try
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Strin:='Der Graph enthält: ' + Inttostr(Graph.AnzahlKnoten)
    +' Knoten und '+ Inttostr(Graph.AnzahlKantenmitSchlingen)+'
    Kante(n) ';Gebiete:=2+Graph.AnzahlKanten-
    Graph.AnzahlKnoten;
    if (Graph.AnzahlKnoten>0) and (Graph.AnzahlKomponenten=1)
     then
```

```
     Strin:=Strin+Chr(13)+'Wenn der Graph planar ist,enthält er
     '+ Inttostr(Gebiete)
      +' Gebiet(e).';
     Komponenten:=Graph.AnzahlKomponenten;
     Strin:=Strin+chr(13)+'Zahl der Komponenten:
     '+Inttostr(Graph.AnzahlKomponenten);
     Strin:=Strin+chr(13)+'Zahl der Kanten vom Typ:';
     Strin:=Strin+chr(13)+'String:
     '+Inttostr(Graph.AnzahlTypKanten('s'));
     Strin:=Strin+' Integer:
     '+Inttostr(Graph.AnzahlTypKanten('i'));
     Strin:=Strin+' Real:
     '+Inttostr(Graph.AnzahlTypKanten('r'));
     ShowMessage(Strin);
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.ParallelkantenundSchlingenClick(Sender:
TObject);
var Strin2:string;
    Schlingen:Integer;
    Komponenten:INteger;
begin
  try
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Schlingen:=Graph.AnzahlSchlingen;
    Strin2:='Zahl der Kanten zwischen ungleichen Knoten:
    '+Inttostr(Graph.AnzahlKanten);
    Strin2:=Strin2+chr(13)+'Zahl der Kanten zwischen gleichen
    Knoten : '+Inttostr(Schlingen);
    Strin2:=Strin2+chr(13)+'Zahl paralleler Kanten im
    gerichteten Graph: '+Inttostr(Graph.AnzahlparallelerKanten);
    Strin2:=Strin2+chr(13)+'Zahl antiparalleler Kanten im
    gerichteten Graph:
    '+Inttostr(Graph.AnzahlantiparallelerKanten);
    Strin2:=Strin2+chr(13)+'Zahl paralleler Kanten im
    ungerichteten Graph:
    '+Inttostr(Graph.AnzahlparallelerKantenungerichtet);
    ShowMessage(Strin2);
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;
```

```
procedure TKnotenform.EulerlinieClick(Sender: TObject);
var Strin,St:string;
    Kno1,Kno2:TInhaltsknoten;
    Gebiete,Schlingen:Integer;
    Komponenten,zahl:Integer;
    Kreiszahl:string;
begin
  try
    Bil*dloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Strin:='';
    Ausgabe1.Caption:='Berechnung läuft...';
    Ausgabe1.Refresh;
    if not Graph.Leer then
    begin
      if Graph.GraphhatgeschlosseneEulerlinie(false) then
      Strin:='Wären alle Kanten  ungerichtet, hat der Graph ei
       ne geschlossene Eulerlinie.';
      if Graph.GraphhatoffenenEulelinie
      (TKnoten(Kno1),TKnoten(Kno2),false) then
        Strin:='Wären alle Kanten ungerichtet,hat der Graph  ei
         ne offene Eulerlinie';
      if (not Graph.GraphhatgeschlosseneEulerlinie(false))and
      (not Graph.GraphhatoffeneEulerlinie
      (TKnoten(Kno1),TKnoten(Kno2),false))
        then
        Strin:='Wären alle Kanten ungerichtet, hat der Graph
         keine geschlossene'
         +' und keine offene Eulerlinie.';
        if Graph.GraphhatoffeneEulelinie
        (TKnoten(Kno1),TKnoten(Kno2),false) then
        Strin:=Strin+' zwischen '+Kno1.Wert+' und
         '+Kno2.Wert+'.';
         Strin:=Strin+chr(13);
        if Graph.GraphhatgeschlosseneEulerlinie(true) then
          Strin:=Strin+'Falls alle Kanten gerichtet sind,hat der
           Graph'+' eine geschlossene Eulerlinie.';
           if Graph.GraphhatoffeneEulerlinie
          (TKnoten(Kno1),TKnoten(Kno2),true) then
          Strin:=Strin+'Falls alle Kanten gerichtet sind,hat der
             Graph'+' eine offene Eulerlinie.';
         if (not Graph.GraphhatgeschlosseneEulerlinie(true))and
         (not Graph.GraphhatoffeneEulerlinie
         (TKnoten(Kno1),TKnoten(Kno2),true))
            then
          Strin:=Strin+'Falls alle Kanten gerichtet sind,hat
           der Graph'+' keine geschlossene'+' und keine
            offene Eulerlinie.';
         if Graph.GraphhatoffeneEulerlinie
         (TKnoten(Kno1),TKnoten(Kno2),true) then
```

```
       Strin:=Strin+' zwischen '+Kno1.Wert+' und
        '+Kno2.Wert+'.';
        Ausgabe1.Caption:='';
        Ausgabe1.Refresh;
       if Strin<>'' then ShowMessage(Strin);
      end
      else
      ShowMessage('Leerer Graph!');Ausgabe1.Caption:=' ';
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenform.KreiseClick(Sender: TObject);
var Strin,St1,St2,SLaenge:string;
    Zahl1,Zahl2,Zahl3,Laenge:Integer;
    Kreiszahl1,Kreiszahl2:string;
    Sliste:TStringList;
    Baum,Weiter:Boolean;
begin
  try Bildloeschen;Graph.ZeichneGraph(Paintbox.Canvas);
   if MessageDlg('Bei umfangreichen Graphen dauert diese Unter
   suchung länger!Trotzdem ausführen?',
   mtConfirmation, [mbYes, mbNo], 0) = mrYes
    then
    begin
      Sliste:=TStringList.Create;
      Bildloeschen;
      if not Graph.Leer then
      begin
        menuenabled(false);
        Paintbox.OnMouseDown:=PanelDownMouse;
        Paintbox.OnDblClick:=nil;
        Paintbox.OnMouseMove:=nil;
        GraphH:=Graph.InhaltskopiedesGraphen
        (TInhaltsgraph,TInhaltsknoten,TInhaltskante,true);
         Aktiv:=false;
        Ausgabe1.Caption:='Berechnung läuft... ';
         Ausgabe1.Refresh;
         Bildloeschen;
        GraphH.ZeichneGraph(Paintbox.Canvas);
        Showmessage('Untersuchung des ungerichteten Graphen');
        if GraphH.GraphhatKreise
          then
          Strin:='Der vorgegebene Graph hat einen Kreis!'
          else
          Strin:='Der vorgegebene Graph hat keinen Kreis.';
         SListe.Add(Strin);
        Ausgabe1.Caption:='Kleinster Kreis wird gesucht...';
```

```
 Zahl1:=GraphH.AnzahlKnotenkleinsterKreis(St1,Paintbox.Canvas);
  Ausgabe1.Caption:=’Größter Kreis wird gesucht...’;
  Kreiszahl1:=Integertostring(Zahl1);
Zahl2:=GraphH.AnzahlKnotengroesterKreis(St2,Paintbox.Canvas);
  Kreiszahl2:=Integertostring(Zahl2);
  Strin:=Strin+chr(13)+’Ungerichteter Graph:’;
  Strin:=Strin+chr(13)+ ‘Kantenzahl des Kreises mit klein
  ster Kantenzahl(>2): ‘+Kreiszahl1;
  Strin:=Strin+chr(13)+ ‘Kantenzahl des Kreises mit größter
  Kantenzahl(>2): ‘+Kreiszahl2;
  Sliste.Add(‘Ungerichteter Graph:’);
  Sliste.Add(‘Kreis mit kleinster Kantenzahl(>2):’);
  Sliste.Add(‘Kantenzahl: ‘+Kreiszahl1);
  Sliste.Add(‘Kantenfolge:’);
   Sliste.Add(St1);
  Sliste.Add(‘Kreis mit größter Kantenzahl(>2):’);
  Sliste.Add(‘Kantenzahl: ‘+Kreiszahl2);
  Sliste.Add(‘Kantenfolge:’);
   Sliste.Add(St2);
  if Graph.AnzahlKomponenten=1 then
   if Graph.AnzahlKanten-Graph.AnzahlSchlingen-Graph.
     AnzahlparallelerKantenungerichtet=Graph.AnzahlKnoten*
      (Graph.AnzahlKnoten-1) Div 2 then
        Strin:=Strin+chr(13)+’Der schlichte Graph ist voll
           ständig.’;
   Ausgabe1.Caption:=’’;
    Baum:=false;
   if (Graph.AnzahlKomponenten=1) and
   (Graph.AnzahlSchlingen=0)and
   (Graph.AnzahlparallelerKantenungerichtet=0)
   and ((Graph.AnzahlKnoten-Graph.AnzahlKanten)=1) then
       Baum:=true;
   if Baum then Strin:=Strin+chr(13)+’Der Graph ist ein
     Baum.’;
    if (Graph.AnzahlKnoten>0) and
   (Graph.AnzahlKomponenten=1) then
    if ((Graph.AnzahlKanten-
   Graph.AnzahlparallelerKantenungerichtet-Graph.
   AnzahlSchlingen>3*Graph.AnzahlKnoten-6)
   and(Graph.AnzahlKomponenten=1)and(Graph.AnzahlKnoten>=3))
   or ((Zahl1>2)and ((Zahl1-2)*(Graph.AnzahlKanten-Graph.
   AnzahlparallelerKantenungerichtet-
   Graph.AnzahlSchlingen)>Zahl1*
   (Graph.AnzahlKnoten-2)))
     then
     Strin:=Strin+chr(13)+’Der schlichte Graph ist nicht
        planar.’
      else
     if (Graph.AnzahlKanten-
    Graph.AnzahlparallelerKantenungerichtet=
```

```
          3*Graph.AnzahlKnoten-6)
         and(Graph.AnzahlKomponenten=1)and(Graph.AnzahlKnoten>=3)
         then
           Strin:=Strin+chr(13)+'Falls der Graph eben ist,ist
           der schlichte Graph Maximal planar.'
         else
        if Baum then Strin:=Strin+chr(13)+'Der Graph ist eben.'
        else
          Strin:=Strin+chr(13)+'Keine Aussage über (Nicht-
           )Planarität des Graphen!';
         SLaenge:='3';
       Weiter:=InputQuery('Suchen aller Kreise fester Länge',
       'Kreislaenge eingeben:',SLaenge);
        if not Weiter
         then
         begin
          menuenabled(true);
           exit;
        end;
       Bildloeschen;
      GraphH.zeichneGraph(Paintbox.Canvas);
      Laenge:=StringtoInteger(SLaenge);
      Ausgabe1.Caption:='Suchen aller Kreise fester Länge...';
      Showmessage('Anzeige im Ausgabefenster und als Pfade im
       Demomodus');
      Sliste.Add('Alle Kreise der Länge:
      '+Integertostring(Laenge));
      SListe.Add('(Ausgabe als Knotenfolge)');
      Zahl3:=GraphH.KreisefesterLaenge
      (Laenge,SListe,Paintbox.Canvas,Ausgabe1);
      SListe.Add('Zahl der Kreise: '+Integertostring(Zahl3));
      Strin:=Strin+chr(13)+'Die Zahl der Kreise der Länge '+
      Integertostring(Laenge)+' ist :'+Integertostring(Zahl3);
      StringlistnachListbox(Sliste,Listbox);
       Sliste.Free;
       Sliste:=niL;
      Ausgabe1.Caption:='';
      if (Strin<>'') and (not Graph.Abbruch) then
        ShowMessage(Strin);
      if Graph.Abbruch then ShowMessage('Abbruch!');
      end
      else
       ShowMessage('Leerer Graph!');
   Menuenabled(true);
  end;
 except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;
```

```
procedure TKnotenform.AnzahlBrueckenClick(Sender: TObject);
var Sliste:TStringList;
    AnzahlBruecken:Integer;
begin
  try Bildloeschen;Graph.ZeichneGraph(Paintbox.Canvas);
    if MessageDlg('Bei umfangreichen Graphen dauert diese Unter
     suchung länger!Trotzdem ausführen?',
     mtConfirmation, [mbYes, mbNo], 0) = mrYes
     then
     begin
      SListe:=TStringlist.create;
      Sliste.Add('Brücken:');
      GraphH:=Graph.InhaltskopiedesGraphen
      (TInhaltsgraph,TInhaltsknoten,TInhaltskante,true);
      Ausgabe1.Caption:='Berechnung läuft... ';
       Ausgabe1.Refresh;
       Bildloeschen;
       Aktiv:=false;
      Paintbox.OnMouseDown:=PanelDownMouse;
      Paintbox.OnDblClick:=nil;
      Paintbox.OnMouseMove:=nil;
      AnzahlBruecken:=GraphH.AnzahlBruecken
      (Sliste,Ausgabe1,Paintbox.Canvas);
      StringlistnachListbox(Sliste,Listbox);
      Ausgabe1.Caption:=' ';
       Ausgabe1.Refresh;
       Bildloeschen;
      GraphH.ZeichneGraph(Paintbox.Canvas);
       if not Graph.Abbruch then
      ShowMessage('Zahl der Brücken im ungerichteten Graph:
      '+Inttostr(AnzahlBruecken));
       Sliste.Free;
     end;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.KnotenanzeigenClick(Sender: TObject);
var Sliste:TStringList;
begin
  try
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Sliste:=Graph.AlleKnotenbestimmen;
    ShowMessage('Knotenanzahl: '+Inttostr(Sliste.Count));
```

```
    Stringlistnachlistbox(Sliste,Listbox);
    Listbox.Top:=Knotenform.Top+Knotenform.Height-85;
    Listbox.Top:=Panel.Top-15;
    Listbox.Width:=Knotenform.Width-20*Knotenform.Width div 640;
    Listbox.Height:=18;
    VertScrollBar.Position:=VertScrollBar.Range;
    if Listbox.Items.Count>0
    then
      Listbox.Visible:=true;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.KantenanzeigenClick(Sender: TObject);
var Sliste:TStringList;
    Strin:string;
begin
  try
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Sliste:=Graph.AlleKantenbestimmen;
    Strin:= 'Kantenzahl: '+Inttostr(Sliste.Count);
    Strin:=Strin+chr(13)+'Gerichtete Kanten:
    '+Inttostr(Graph.AnzahlgerichteteKanten);
    Strin:=Strin+chr(13)+'Ungerichtete Kanten:
    '+Inttostr(Graph.AnzahlungerichteteKanten);
    Strin:=Strin+chr(13)+'Kantensumme: '
     +RundeZahltostring(Graph.Kantensumme(Bewertung),Graph.Kantengenauigkeit);
    ShowMessage(Strin);
    Stringlistnachlistbox(Sliste,Listbox);
    Listbox.Top:=Panel.Top-15;
    Listbox.Width:=Knotenform.Width-20*Knotenform.Width DIV 640;
    Listbox.Height:=18;
    VertScrollBar.Position:=VertScrollBar.Range;
    if Listbox.Items.Count>0
    then
      Listbox.Visible:=true;
  except
    Menuenabled(true);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenform.AusgabefensterClick(Sender: TObject);
begin
  try
```

```
    Ausgabeform:=TAusgabeform.Create(self);
    Ausgabeform.Listenbox:=Listbox;
    Ausgabeform.Showmodal;
    Ausgabeform.Free;
    Ausgabeform:=nil;
  except
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenform.AbbruchClick(Sender: TObject);
begin
  try
    Paintbox.OnMouseDown:=PaintboxMousedown;
    Paintbox.OnDblClick:=PaintboxDblclick;
    Paintbox.OnMousemove:=PaintboxMousemove;
    Menuenabled(true);
    Graph.FaerbeGraph(clblack,pssolid);
    if Graph.Abbruch=false
    then
      Graph.Abbruch:=true
    else
      Graph.Abbruch:=false;
    if GraphH<>nil then
    begin
      if GraphH.Abbruch=false
      then
        GraphH.Abbruch:=true
      else
        GraphH.Abbruch:=false;
      Graph.Abbruch:=GraphH.Abbruch;
    end;
    if Graph.Abbruch
    then
      begin
        Abbruch.Caption:='Abbruch aus';
        Graph.Demo:=false;
        if GraphH<>nil then GraphH.Demo:=false;
        Demo.Caption:='Demo an';
      end
    else
      Abbruch.Caption:='Abbruch an';
    if Graph.Abbruch
    then
      Caption:='Abbruch an!';
    if not Graph.Abbruch then Caption:='Knotenform';
    if Graph.Abbruch then
    begin
      Graph.Kantenwertposition:=0;
      Graph.Knotenwertposition:=0;
```

```
      if GraphH<>nil then
      begin
        GraphH.Kantenwertposition:=0;
        GraphH.Knotenwertposition:=0;
      end;
      Aktiv:=true;
      Ausgabe1.Caption:='';
      Ausgabe2.Caption:='';
      Eingabe.Visible:=false;
      Button.Visible:=false;
      Graph.FaerbeGraph(clblack,pssolid);
    end;
    if ((Graph<>nil)and (Graph.Stop)) or ((GraphH<>nil)and
    (GraphH.Stop)) then
    begin
      Caption:='Knotenform';
      Abbruch.Caption:='Abbruch an';Aktiv:=false;
      ShowMessage('Die schon ermittelten Ergebnisse wurden
      ausgegeben!');
    end;
  Bildloeschen;
  Paintboxpaint(Sender);
  except
    Fehler;
  end;
end;


procedure TKnotenform.DemoClick(Sender: TObject);
begin
  try
    if Graph.Demo=false
    then
      Graph.Demo:=true
    else
      Graph.Demo:=false;
    if GraphH<>nil then
    begin
      if GraphH.Demo=false
      then
        GraphH.Demo:=true
      else
        GraphH.Demo:=false;
      Graph.Demo:=GraphH.Demo;
      Graph.Pausenzeit:=GraphH.Pausenzeit;
    end;
    if Graph.Demo
    then
      Demo.Caption:='Demo aus'
    else
```

```
      Demo.Caption:=’Demo an’;
    if Graph.Demo
    then
      begin
        Caption:=’Demo an! Pausenzeit: ‘
        +IntegertoString(Graph.Pausenzeit div 1000)+’ s’;
        Graph.Abbruch:=false;
        Abbruch.Caption:=’Abbruch an’;
      end;
    if Graph.Demo then
    begin
      Abbruch.Caption:=’Abbruch an’;
      Graph.Abbruch:=false;
      if GraphH<>nil then GraphH.Abbruch:=false;
    end;
    if not Graph.Demo then Caption:=’Knotenform’;
    if Graph.Demo then  PausenzeitClick(Sender);
  except
    Fehler;
  end;
end;


procedure TKnotenform.PausenzeitClick(Sender: TObject);
label Endproc;
var St:string;
    Eingabe:Boolean;
begin
  try
    St:=’0';
    repeat
      Eingabe:=InputQuery(‘Eingabe Demozeit in s’, ‘Eingabe der
       Demozeit: ‘, St);
       if not Eingabe then goto Endproc;
      if (StringtoInteger(St)<0)or(StringtoInteger(St)>31) then
        begin
         ShowMessage(‘Zeit nicht im Bereich von  0 bis 31 s !’);
           St:=’’;
         end
      until St<>’’;
       if abs(StringtoInteger(St))<32
       then
        Graph.Pausenzeit:=abs(StringtoInteger(St)*1000)
       else
        Graph.Pausenzeit:=31000;
      if GraphH<>nil then GraphH.Pausenzeit:=Graph.Pausenzeit;
      if Graph.Demo then Caption:=’Demo an! Pausenzeit: ‘
        +IntegertoString(Graph.Pausenzeit div 1000)+’ s’;
    Endproc:
  except
```

```
    Fehler;
  end;
end;


procedure TKnotenform.InhaltClick(Sender: TObject);
begin
  try
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Application.Helpjump('Inhalt');
  except
    Fehler;
  end;
end;

procedure TKnotenform.InfoClick(Sender: TObject);
begin
  try
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    ShowMessage('Author Rainer Eidmann');
  except
    Fehler;
  end;
end;

procedure TKnotenform.InitBitBtnmenu;
begin
  try
    GraphK.Freeall;
    GraphK:=Graph.InhaltskopiedesGraphen
    (TInhaltsgraph,TInhaltsknoten,TInhaltskante,false);
    if Graph.Abbruch then Menuenabled(true);
    if (Mainmenu.Items[0].Items[0].Enabled=true) then
     begin
      Paintbox.OnMouseDown:=PaintboxMousedown;
      Paintbox.OnDblClick:=PaintboxDblclick;
      Paintbox.OnMousemove:=PaintboxMousemove;
      Listbox.Visible:=false;
      Listbox.Top:=360;
      Listbox.Height:=16;
       Aktiv:=true;
      if (GraphH<>nil) and (GraphH<>Graph)and(not
       (BitBtn3.Focused or
      BitBtn4.Focused or BItBtn11.Focused)) then
        begin
         GraphH.Freeall;
         GraphH:=nil;
      end;
```

```
      if GraphH<>nil then
      begin
        GraphH.Zustand:=false;
        GraphH.letzterMausklickknoten:=nil;
      end;
      Ausgabe1.Caption:='';
      Ausgabe1.Refresh;
      Ausgabe2.Caption:='';
      Ausgabe2.Refresh;
    end;
    Graph.Abbruch:=false;
    Abbruch.Caption:='Abbruch an';
    if Graph.demo = false then Caption:='Knotenform';
  except
    Fehler;
  end;
end;


procedure TKnotenform.BitBtn1Click(Sender: TObject);
begin
  InitBitBtnmenu;Bildloeschen;Graph.ZeichneGraph(Paintbox.Canvas);
  NeueKnotenClick(Sender);
  BitBtn1.Enabled:=false;
  BitBtn1.Enabled:=true;
end;

procedure TKnotenform.BitBtn2Click(Sender: TObject);
begin
  InitBitBtnmenu;Bildloeschen;Aktiv:=true;
  Graph.ZeichneGraph(Paintbox.Canvas);GraphladenClick(Sender);
  BitBtn2.Enabled:=false;
  BitBtn2.Enabled:=true;
end;

procedure TKnotenform.BitBtn3Click(Sender: TObject);
begin
  InitBitBtnmenu;Bildloeschen;Aktiv:=true;
  Graph.ZeichneGraph(Paintbox.Canvas);GraphspeichernunterClick(Sender);
   BitBtn3.Enabled:=false;
   BitBtn3.Enabled:=true;
end;

procedure TKnotenform.BitBtn4Click(Sender: TObject);
begin
  InitBitBtnmenu;if GraphH<>nil then Aktiv:=false;
  GraphdruckenClick(Sender);Bildloeschen;Aktiv:=true;
  Graph.ZeichneGraph(Paintbox.Canvas);BitBtn4.Enabled:=false;
  BitBtn4.Enabled:=true;
end;
```

```
procedure TKnotenform.BitBtn5Click(Sender: TObject);
begin
  InitBitBtnmenu;
  KnotenloeschenClick(Sender);
  BitBtn5.Enabled:=false;
  BitBtn5.Enabled:=true;
end;

procedure TKnotenform.BitBtn6Click(Sender: TObject);
begin
  InitBitBtnmenu;
  KnoteneditierenClick(Sender);
  BitBtn6.Enabled:=false;
  BitBtn6.Enabled:=true;
end;

procedure TKnotenform.BitBtn7Click(Sender: TObject);
begin
  InitBitBtnmenu;
  KnotenverschiebenClick(Sender);
  BitBtn7.Enabled:=false;
  BitBtn7.Enabled:=true;
end;

procedure TKnotenform.BitBtn8Click(Sender: TObject);
begin
  InitBitBtnmenu;
  KantenloeschenClick(Sender);
  BitBtn8.Enabled:=false;
  BitBtn8.Enabled:=true;
end;

procedure TKnotenform.BitBtn9Click(Sender: TObject);
begin
  InitBitBtnmenu;
  KanteeditierenClick(Sender);
  BitBtn9.Enabled:=false;
  BitBtn9.Enabled:=true;
end;

procedure TKnotenform.BitBtn10Click(Sender: TObject);
begin
  InitBitBtnmenu;
  KanteverschiebenClick(Sender);
  BitBtn10.Enabled:=false;
  BitBtn10.Enabled:=true;
end;

procedure TKnotenform.BitBtn11Click(Sender: TObject);
begin
```

```
  InitBitBtnmenu;
  BildzeichnenClick(Sender);
  BitBtn11.Enabled:=false;
  BitBtn11.Enabled:=true;
end;


end.
```

Unit UForm: (für DWK)

```
unit UForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs,
  UKNOTEN, Menus, Buttons, StdCtrls,
  ExtCtrls,UList,UGraph,UInhgrph,
  UMath1,UMath2,UPfad,UAusgabe,UKante;

type
  TKnotenformular = class(TKnotenform)
    Pfade: TMenuItem;
    AllePfade: TMenuItem;
    AlleKreise: TMenuItem;
    MinimalePfade: TMenuItem;
    AnzahlZielknoten: TMenuItem;
    TieferBaum: TMenuItem;
    WeiterBaum: TMenuItem;
    AbstandvonzweiKnoten1: TMenuItem;
    AllePfadezwischenzweiKnoten: TMenuItem;
    MinimalesGerstdesGraphen: TMenuItem;
    Anwendungen: TMenuItem;
    Netzwerkzeitplan: TMenuItem;
    Hamiltonkreise: TMenuItem;
    Eulerkreis: TMenuItem;
    Frbbarkeit: TMenuItem;
    EulerLinie: TMenuItem;
    EndlicherAutomat: TMenuItem;
    GraphalsRelation: TMenuItem;
    MaximalerNetzfluss: TMenuItem;
    Gleichungssystem: TMenuItem;
    Markovketteabs: TMenuItem;
    Markovkettestat: TMenuItem;
    Graphreduzieren: TMenuItem;
    MinimaleKosten: TMenuItem;
    Transportproblem: TMenuItem;
```

```
OptimalesMatching: TMenuItem;
ChinesischerBrieftrger: TMenuItem;
MaximalesMatching: TMenuItem;
{Ersatz-Ereignismethoden}
procedure AllePfadeMousedown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure AbstandvonzweiKnotenMouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure NetzMouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure Automaten1Mousedown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure Automaten2Mousedown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure GleichungssystemMousedown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure MarkovMouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure KostenMouseDown(Sender: TObject;Button:
TMouseButton;Shift: TShiftState; X, Y: Integer);
{Menü-Ereignismethoden}
procedure AllePfadeClick(Sender: TObject);
procedure AllePfade1Click(Sender: TObject);
procedure AlleKreiseClick(Sender: TObject);
procedure MinimalePfadeClick(Sender: TObject);
procedure AnzahlZielknotenClick(Sender: TObject);
procedure TieferBaumClick(Sender: TObject);
procedure WeiterBaumClick(Sender: TObject);
procedure AbstandvonzweiKnoten1Click(Sender: TObject);
procedure AllePfadezwischenzweiKnotenClick(Sender: TObject);
procedure MinimalesGerstdesGraphenClick(Sender: TObject);
procedure NetzwerkzeitplanClick(Sender: TObject);
procedure HamiltonkreiseClick(Sender: TObject);
procedure EulerkreisgschlossenClick(Sender: TObject);
procedure FrbbarkeitClick(Sender: TObject);
procedure EulerLinieoffenClick(Sender: TObject);
procedure EndlicherAutomatClick(Sender: TObject);
procedure EingabeKeyPress(Sender: TObject; var Key: Char);
procedure ButtonClick(Sender: TObject);
procedure GraphalsRelationClick(Sender: TObject);
procedure MaximalerNetzflussClick(Sender: TObject);
procedure MaximalesMatchingClick(Sender: TObject);
procedure GleichungssystemClick(Sender: TObject);
procedure MarkovketteabsClick(Sender: TObject);
procedure MarkovkettestatClick(Sender: TObject);
procedure GraphreduzierenClick(Sender: TObject);
procedure MinimaleKostenClick(Sender: TObject);
procedure TransportproblemClick(Sender: TObject);
procedure OptimalesMatchingClick(Sender: TObject);
procedure ChinesischerBrieftrgerClick(Sender: TObject);
```

```
    procedure AbbruchClick(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

 TAutomatenneugraph=class(TAutomatengraph)
   constructor Create;override;
 end;

var
  Knotenformular: TKnotenformular;
  Automatenneugraph:TAutomatenneugraph;
  Automatenneugraphaktiv:Boolean;


implementation

{$R *.DFM}

Constructor TAutomatenneugraph.Create;
begin
  inherited Create;
  InhaltsKnotenclass:=TAutomatenknoten;
  Inhaltskanteclass:=TInhaltskante;
  Knotenformular.Graph:=self;
end;



procedure TKnotenformular.AllePfadeMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var Sliste:TStringList;
begin
  try
    Sliste:=TStringList.Create;
    if TPfadgraph(GraphZ).
    AllePfadezwischenzweiKnotenbestimmen(X,Y,Ausgabe1,
    Sliste,Paintbox.Canvas)
     then
      Paintbox.OnMouseDown:=PanelDownMouse;
      Paintbox.OnDblClick:=nil;
      Paintbox.OnMouseMove:=nil;
    if Sliste.count>0 then
    StringlistnachListbox(Sliste,Listbox);SListe.Free;SListe:=nil;
    GraphZ.ZeichneGraph(Paintbox.Canvas);
    if GraphZ.Abbruch then begin Bildloeschen;Graph.ZeichneGraph
    (Paintbox.Canvas);Aktiv:=true;ShowMessage('Abbruch!');end;
  except
```

```
    ShowMessage(‘Fehler’);
   end;
end;


procedure TKnotenformular.AbstandvonzweiKnotenMouseDown(Sender:
TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var Sliste:TStringList;
begin
  try
    Sliste:=TStringList.Create;
    if TPfadgraph(GraphZ).
    MinimalenPfadzwischenzweiKnotenbestimmen(X,Y,Ausgabe1,
       Sliste,Paintbox.Canvas)
     then
      Paintbox.OnMouseDown:=PanelDownMouse;
      Paintbox.OnDblClick:=nil;
      Paintbox.OnMouseMove:=nil;
      if Sliste.Count>0 then
     StringlistnachListbox(Sliste,Listbox);SListe.Free;SListe:=nil;
      GraphZ.ZeichneGraph(Paintbox.Canvas);
      if GraphZ.Abbruch then begin Bildloeschen;Graph.ZeichneGraph
     (Paintbox.Canvas);Aktiv:=true;ShowMessage('Abbruch!');end;
  except
    ShowMessage(‘Fehler’);
  end;
end;


procedure TKnotenformular.NetzMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var Kno:TInhaltsknoten;
begin
  try
    GraphH.Knotenwertposition:=4;
    GraphH.Knoteninhaltzeigen(X,Y);
    GraphH.Knotenwertposition:=0;
    if GraphH.FindezuKoordinatendenKnoten(X,Y,Kno)=false then
    begin
      GraphH.Kantenwertposition:=1;
      GraphH.Kanteninhaltzeigen(X,Y);
      GraphH.Kantenwertposition:=0;
    end;
  except
    ShowMessage(‘Fehler’);
  end;
end;
```

```
procedure TKnotenformular.Automaten1Mousedown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
label Ende;
  var Kno:TAutomatenknoten;
begin
  try
    if GraphH.Abbruch then goto Ende;
    if GraphH.FindezuKoordinatendenKnoten
    (X,Y,TInhaltsknoten(Kno))
     then
     begin
      Kno.Knotenart:=1;
      Kno.Stil:=psdash;
      Kno.Farbe:=clgreen;
      Paintbox.OnMouseDown:=PaintboxMouseDown;
      GraphH.ZeichneGraph(Paintbox.Canvas);
      ShowMessage('Anfangszustand: '+Kno.Wert);
      Ende:
      GraphH.Zustand:=true;
      end
      else
      ShowMessage('Knoten mit  Maus auswählen!');
     GraphH.ZeichneGraph(Paintbox.Canvas);
     Aktiv:=false;
  except
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenformular.Automaten2Mousedown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
label Ende;
  var Kno:TAutomatenknoten;
begin
  try
  if GraphH.Abbruch then goto Ende;;
    if
GraphH.FindezuKoordinatendenKnoten(X,Y,TInhaltsknoten(Kno))
    then
    begin
      if Kno.Knotenart<> 1 then
      begin
     Kno.Knotenart:=2;if not Mainmenu.Items[0].Items[0].Enabled          th
     then begin Kno.stil:=psdashdot;
        Kno.Farbe:=clblue;
        GraphH.ZeichneGraph(Paintbox.Canvas);end;
      end
      else
      begin if not Mainmenu.Items[0].Items[0].Enabled then begin
```

```
        ShowMessage('Knoten ist auch Anfangszustand!');
        Kno.Knotenart:=3;end;
      end;
      Paintbox.OnMouseDown:=PaintboxMouseDown;
      ShowMessage('Endzustand: '+Kno.Wert);
      Ende:
      GraphH.Zustand:=true;
      if GraphH.Abbruch then
      begin
        Aktiv:=true;
        Bildloeschen;
        if (GraphH<>nil) and (GraphH<>Graph) then
           GraphH.Freeall;
          GraphH:=nil;
        Graph.zeichneGraph(Paintbox.Canvas);
      end;
    end
    else
      ShowMessage('Knoten mit Maus auswählen!');
    GraphH.ZeichneGraph(Paintbox.Canvas);
    Aktiv:=false;
  except
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenformular.GleichungssystemMousedown(Sender:
TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
label Endproc;
var Kno:TGleichungssystemknoten;
begin
  try
    if GraphH.Abbruch then
    begin
      GraphH.Zustand:=true;
      Paintbox.OnMouseDown:=PaintboxMousedown;
      Paintbox.OnDblClick:=PaintboxDblclick;
      Paintbox.OnMousemove:=PaintboxMousemove;
      Aktiv:=true;
      Menuenabled(true);
      Bildloeschen;
      Graph.zeichneGraph(Paintbox.Canvas);
      goto Endproc;
    end;
   if GraphH.FindezuKoordinatendenKnoten
   (X,Y,TInhaltsknoten(Kno))
    then
    begin
```

```
            GraphH.LetzterMausklickknoten:=Kno;
            Kno.Stil:=psdot;
            Kno.Farbe:=clred;
if Mainmenu.Items[0].Items[0].Enabled then ShowMessage('Berechnete Wahrscheinlichk
            GraphH.ZeichneGraph(Paintbox.Canvas);
            GraphH.Zustand:=true;
          end
          else
            ShowMessage('Knoten mit Maus auswählen!');
          GraphH.ZeichneGraph(Paintbox.Canvas);
          Aktiv:=false;
          Endproc:
        except
          ShowMessage('Fehler');
        end;
      end;


      procedure TKnotenformular.MarkovMouseDown(Sender: TObject;
        Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
      var GraphK:TInhaltsknoten;
      begin
        try
          if Graph.FindezuKoordinatendenKnoten(X,Y,GraphK)=true
           then
            Graph.LetzterMausklickknoten:=GraphK;
          GraphH.Knotenwertposition:=7;
          GraphH.Knoteninhaltzeigen(X,Y);
          GraphH.Knotenwertposition:=8;
        except
          ShowMessage('Fehler');
        end;
      end;


      procedure TKnotenformular.KostenMouseDown(Sender: TObject;
        Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
      label Ende;
      var Ka:TMinimaleKostenkante;
           Gefunden:Boolean;
           Str1,Str2,Str3:string;
           Index:Integer;
           Eingabe:Boolean;
      begin
        try
          if GraphH.Abbruch then goto Ende;
          Gefunden:=false;
          if not GraphH.Kantenliste.Leer then
            for Index:=0 to GraphH.Kantenliste.Anzahl-1 do
            begin
```

```
        Ka:=TMinimaleKostenKante(GraphH.Kantenliste.Kante(Index));
         if Ka.MausklickaufKante(X,Y) then
         begin
           Gefunden:=true;
           Str2:='Kante '+Ka.Anfangsknoten.Wert
             +Ka.Endknoten.Wert+' :';
           repeat
             Str3:=Ka.Wert;
             Str1:='Eingabe der Kosten ';
             Eingabe:=Inputquery(Str1,Str2,Str3);
             if (Abs(StringtoReal(str3))<1.0E30)
             then
               Ka.Kosten:=StringtoReal(str3)
             else
             begin
             ShowMessage('Eingabe nicht im zulässigen Be
               reich!');
               Str3:='';
               Ka.Kosten:=0;
             end;
             if Eingabe=false then
             begin
               GraphH.Zustand:=true;
               goto Ende;
             end
           until Str3<>'';
           GraphH.Zustand:=false;;
           goto Ende;
         end;
         if Gefunden
         then
           GraphH.Zustand:=false
         else
           GraphH.Zustand:=true;
         Ende:
       end;
     Aktiv:=false;
     GraphH.zeichneGraph(Paintbox.Canvas);
   except
     GraphH.Zustand:=true;
     Paintbox.OnMouseDown:=PaintboxMouseDown;
     ShowMessage('Fehler');
   end;
end;

procedure TKnotenformular.AllePfadeClick(Sender: TObject);
var Sliste:TStringList;
begin
  try
    if Graph.Leer then exit;
```

```
    Ausgabe1.Caption:=’Berechnung läuft...’;
    Ausgabe1.Refresh;
    Paintbox.OnMouseDown:=PanelDownMouse;
    Paintbox.OnDblClick:=nil;
    Paintbox.OnMouseMove:=nil;
    GraphH:=Graph.InhaltskopiedesGraphen
    (TPfadgraph,TPfadknoten,TInhaltskante,false);
    Aktiv:=false;
    Menuenabled(false);
    Bildloeschen;
    Menuenabled(false);
    Bildloeschen;
    GraphH.ZeichneGraph(Paintbox.Canvas);
    SListe:=TStringlist.create;;
    with GraphH.LetzterMausklickknoten do
      TPfadgraph(GraphH).AllePfadevoneinemKnotenbestimmen
      (X,Y,Ausgabe1,Sliste,Paintbox.Canvas);
    StringlistnachListbox(Sliste,Listbox);
    Menuenabled(true);
    Bildloeschen;
    GraphH.zeichneGraph(Paintbox.Canvas);
    if Graph.Abbruch then begin Bildloeschen;Graph.ZeichneGraph
      (Paintbox.Canvas);Aktiv:=true;ShowMessage(‘Abbruch!’);end;
    Ausgabe1.Caption:=’’;
  except
    Menuenabled(true);
    Abbruchclick(Sender);
    ShowMessage(‘Fehler’);
  end;

end;

procedure TKnotenformular.AllePfade1Click(Sender: TObject);
var Sliste:TStringList;
begin
  try
    if Graph.Leer then exit;
    Ausgabe1.Caption:=’Berechnung läuft...’;
    Ausgabe1.Refresh;
    Paintbox.OnMouseDown:=PanelDownMouse;
    Paintbox.OnDblClick:=nil;
    Paintbox.OnMouseMove:=nil;
    GraphH:=Graph.InhaltskopiedesGraphen
    (TPfadgraph,TPfadknoten,TInhaltskante,false);
    Aktiv:=false;
    Menuenabled(false);
    Bildloeschen;
    Menuenabled(false);
    Bildloeschen;
    GraphH.ZeichneGraph(Paintbox.Canvas);
```

```
    SListe:=TStringlist.create;;
    with GraphH.LetzterMausklickknoten do
      TPfadgraph(GraphH).AllePfadevoneinemKnotenbestimmen
      (X,Y,Ausgabe1,Sliste,Paintbox.Canvas);
    StringlistnachListbox(Sliste,Listbox);
    Menuenabled(true);
    Bildloeschen;
    GraphH.zeichneGraph(Paintbox.Canvas);
    if Graph.Abbruch then begin Bildloeschen;Graph.ZeichneGraph
    (Paintbox.Canvas);Aktiv:=true;ShowMessage('Abbruch!');end;
    Ausgabe1.Caption:='';
  except
    Menuenabled(true);
    Abbruchclick(Sender);
    ShowMessage('Fehler');
  end;

end;

procedure TKnotenformular.AlleKreiseClick(Sender: TObject);
var Sliste:TStringList;
begin
  try
    if Graph.Leer then exit;
    Ausgabe1.Caption:='Berechnung läuft...';
    Ausgabe1.Refresh;
    Paintbox.OnMouseDown:=PanelDownMouse;
    Paintbox.OnDblClick:=nil;
    Paintbox.OnMouseMove:=nil;
    GraphH:=Graph.InhaltskopiedesGraphen
    (TPfadgraph,TPfadknoten,TInhaltskante,false);
    Aktiv:=false;
    Menuenabled(false);
    Bildloeschen;
    GraphH.ZeichneGraph(Paintbox.Canvas);
    SListe:=TStringlist.create;
    with GraphH.LetzterMausklickknoten do
     TPfadGraph(GraphH).AlleKreisevoneinemKnotenbestimmen
     (X,Y,Ausgabe1,Sliste,Paintbox.Canvas);
    StringlistnachListbox(Sliste,Listbox);
    Menuenabled(true);
    GraphH.zeichneGraph(Paintbox.Canvas);
    if Graph.Abbruch then begin Bildloeschen;Graph.ZeichneGraph
    (Paintbox.Canvas),Aktiv:=true;ShowMessage('Abbruch!');end;
    Ausgabe1.Caption:='';
    SListe.free;
    Sliste:=nil;
  except
    Menuenabled(true);
    AbbruchClick(Sender);
```

```
    ShowMessage('Fehler');
  end

end;

procedure TKnotenformular.MinimalePfadeClick(Sender: TObject);
var Sliste:TStringList;
begin
  try
    if Graph.Leer then exit;
    Ausgabe1.Caption:='Berechnung läuft...';
    Ausgabe1.Refresh;
    Aktiv:=false;
    Paintbox.OnMouseDown:=PanelDownMouse;
    Paintbox.OnDblClick:=nil;
    Paintbox.OnMouseMove:=nil;
    GraphH:=Graph.InhaltskopiedesGraphen
    (TPfadgraph,TPfadknoten,TInhaltskante,false);
    Menuenabled(false);
    Bildloeschen;
    GraphH.ZeichneGraph(Paintbox.Canvas);
    SListe:=TStringlist.Create;
    with GraphH.LetzterMausklickknoten do
     TPfadgraph(GraphH).AlleminimalenPfadevoneinenKnotenbestimmen
      (X,Y,Ausgabe1,Sliste,Paintbox.Canvas);
    StringlistnachListbox(Sliste,Listbox);
    Menuenabled(true);
    GraphH.zeichneGraph(Paintbox.Canvas);
    if Graph.Abbruch then begin Bildloeschen;Graph.ZeichneGraph
      (Paintbox.Canvas);Aktiv:=true;ShowMessage('Abbruch!');end;
    Ausgabe1.Caption:='';
    SListe.Free;
    Sliste:=nil;
  except
    Menuenabled(true);
    AbbruchClick(Sender);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenformular.AnzahlZielknotenClick(Sender:
TObject);
var Kno:TPfadknoten;
    S:string;
    Xko,Yko:Integer;
begin
  try
    Bi ldloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
```

```
    if not Graph.Leer  then
    begin
      Ausgabe1.Caption:=’Berechnung läuft...’;
      Ausgabe1.Refresh;
      Xko:=Graph.LetzterMausklickknoten.X;
      Yko:=Graph.LetzterMausklickknoten.Y;
      if Graph.FindezuKoordinatendenKnoten
      (Xko,Yko,TInhaltsknoten(Kno))=false then
        Kno:=TPfadknoten(Graph.Anfangsknoten);
      Kno.Graph:=Graph;
      Ausgabe1.Caption:=’’;
      ShowMessage(‘Anzahl Zielknoten:
      ‘+Inttostr(Kno.AnzahlPfadZielKnoten));
       if Graph.Abbruch thenbegin Bildloeschen;Graph.ZeichneGraph
       (Paintbox.Canvas)Aktiv:=true;ShowMessage(‘Abbruch!’);end;
    end;
  except
    Menuenabled(true);
    AbbruchClick(Sender);
    ShowMessage(‘Fehler’);
  end;
end;


procedure TKnotenformular.TieferBaumClick(Sender: TObject);
var Sliste:TStringList;
begin
  try
    if Graph.Leer then exit;
    Menuenabled(false);
    Bildloeschen;
    Aktiv:=false;
    Paintbox.OnMouseDown:=PanelDownMouse;
    Paintbox.OnDblClick:=nil;
    Paintbox.OnMouseMove:=nil;
    GraphH:=Graph.InhaltskopiedesGraphen
    (TPfadgraph,TPfadknoten,TInhaltskante,false);
    Menuenabled(false);
    Bildloeschen;
    GraphH.ZeichneGraph(Paintbox.Canvas);
    SListe:=TStringlist.Create;
    with GraphH.LetzterMausklickknoten do
     TPfadGraph(GraphH).AlletiefenBaumpfadevoneinemKnotenbestimmen
      (X,Y,Ausgabe1,Sliste,Paintbox.Canvas);
    StringlistnachListbox(Sliste,Listbox);
    Bildloeschen;
    GraphH.ZeichneGraph(Paintbox.Canvas);
    Menuenabled(true);
    if Graph.Abbruch then
      ShowMessage(‘Abbruch!’);
```

```
    SListe.Free;
    SListe:=nil;
  except
    Menuenabled(true);
    AbbruchClick(Sender);
    ShowMessage('Fehler');
  end;

end;

procedure TKnotenformular.WeiterBaumClick(Sender: TObject);
var Sliste:TStringList;
begin
  try
    if Graph.Leer then exit;
    Ausgabe1.Caption:='Berechnung läuft...';
    Ausgabe1.Refresh;
    Aktiv:=false;
    Paintbox.OnMouseDown:=PanelDownMouse;
    Paintbox.OnDblClick:=nil;
    Paintbox.OnMouseMove:=nil;
    GraphH:=Graph.InhaltskopiedesGraphen
    (TPfadgraph,TPfadknoten,TInhaltskante,false);
    Menuenabled(false);
    Bildloeschen;
    GraphH.ZeichneGraph(Paintbox.Canvas);
    SListe:=TStringlist.Create;
    with GraphH.LetzterMausklickknoten do
      TPfadgraph(GraphH).
      AlleweitenBaumpfadevoneinemKnotenbestimmen
      (X,Y,Ausgabe1,Sliste,Paintbox.Canvas);
    StringlistnachListbox(Sliste,Listbox);
    GraphH.zeichneGraph(Paintbox.Canvas);
    Menuenabled(true);
    if Graph.Abbruch
    then begin Bildloeschen;Graph.ZeichneGraph(Paintbox.Canvas);
      Aktiv:=true;ShowMessage('Abbruch!');end;
    Ausgabe1.Caption:='';
    SListe.Free;
    Sliste:=nil;
  except
    Menuenabled(true);
    AbbruchClick(Sender);
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenformular.AbstandvonzweiKnoten1Click(Sender:
TObject);
begin
```

```
  try
    if Graph.Leer then exit;
    GraphH:=Graph.InhaltskopiedesGraphen
    (TPfadgraph,TPfadknoten,TInhaltskante,false);
    GraphZ:=GraphH;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.Abbruch then
     begin begin Bildloeschen;Graph.ZeichneGraph(Paintbox.Canvas);
      Aktiv:=true;ShowMessage('Abbruch');end;
      exit;
    end;
    Paintbox.OnMouseDown:=AbstandvonzweiKnotenMouseDown;
    ShowMessage('1. Knoten wählen!');
  except
    Menuenabled(true);
    AbbruchClick(Sender);
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenformular.AllePfadezwischenzweiKnotenClick(
  Sender: TObject);
begin
  try
    if Graph.Leer then exit;
    GraphH:=Graph.InhaltskopiedesGraphen
    (TPfadgraph,TPfadknoten,TInhaltskante,false);
    GraphZ:=GraphH;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.Abbruch then
    begin
       ShowMessage('Abbruch gewählt!');
       exit;
      end;
      Paintbox.OnMouseDown:=AllePfadeMouseDown;
      ShowMessage('1. Knoten wählen!');
      if Graph.Abbruch then begin Bildloeschen;Graph.ZeichneGraph
      (Paintbox.Canvas);Aktiv:=true;ShowMessage('Abbruch!');end;
  except
    Menuenabled(true);
    AbbruchClick(Sender);
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenformular.MinimalesGerstdesGraphenClick(Sender:
TObject);
Var Sliste:TStringList;
```

```
    Graph:TInhaltsgraph;
begin
  try
    Graph:=self.Graph;
    if Graph.Leer then exit;
    Ausgabe1.Caption:='Berechnung läuft...';
    Ausgabe1.Refresh;
    Aktiv:=false;
    Paintbox.OnMouseDown:=PanelDownMouse;
    Paintbox.OnDblClick:=nil;
    Paintbox.OnMouseMove:=nil;
    GraphH:=Graph.InhaltskopiedesGraphen
    (TPfadgraph,TPfadknoten,TInhaltskante,true);
    Menuenabled(false);
    Bildloeschen;
    GraphH.ZeichneGraph(Paintbox.Canvas);
    Sliste:=TStringList.Create;
    TPfadgraph(GraphH).Kruskal(Ausgabe1,Sliste,Paintbox.Canvas);
    StringlistnachListbox(Sliste,Listbox);
    Sliste.Free;
    Sliste:=nil;
    GraphH.ZeichneGraph(Paintbox.Canvas);
    Menuenabled(true);
    Ausgabe1.Caption:='';
    if Graph.Abbruch then begin Bildloeschen;Graph.ZeichneGraph
    (Paintbox.Canvas);Aktiv:=true;ShowMessage('Abbruch!');end;
  except
    Menuenabled(true);
    AbbruchClick(Sender);
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenformular.NetzwerkzeitplanClick(Sender:
TObject);
var Sliste:TStringList;
    Netzgraph:TNetzgraph;
    Oberflaeche:TForm;
    Graph:TInhaltsgraph;
begin
  try
    Graph:=self.Graph;
    if Graph.Leer then exit;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.AnzahlSchlingen>0 then
    begin
      ShowMessage('Der Graph hat Schlingen!');
      exit;
    end;
```

```
  Sliste:=TStringList.Create;
    Netzgraph:=TNetzgraph(Graph.InhaltskopiedesGraphen
    (TNetzgraph,TNetzknoten,
    TNetzkante,false));
    GraphH:=Netzgraph;
    Oberflaeche:=TForm(self);
   Paintbox.OnMouseDown:=NetzMouseDown;;
   Paintbox.ONmousemove:=nil;
   Paintbox.OnDblclick:=nil;
   Netzgraph.Netz(Graph,Oberflaeche,Paintbox.Canvas,Ausgabe1,Sliste);
    Sliste.Insert(0,'Knoten:');
    StringlistnachListbox(Sliste,Listbox);
    SListe.Free;
    SListe:=nil;
    GraphH:=Netzgraph.InhaltskopiedesGraphen
    (TInhaltsgraph,TNetzknoten,TNetzKante,false);;
    Aktiv:=false;
    Menuenabled(true);
    Netzgraph.Freeall;
    Netzgraph:=nil;
    if Graph.Abbruch then
      ShowMessage('Abbruch!');
  except
    ShowMessage('Fehler');
    AbbruchClick(Sender);
    Menuenabled(true);
  end;
end;

procedure TKnotenformular.HamiltonkreiseClick(Sender: TObject);
var Sliste:TStringList;Hamiltongraph:THamiltongraph;
begin
  try
    if Graph.Leer then exit;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.AnzahlKomponenten>1 then
    begin
      ShowMessage('Mehrere Komponenten!');
      exit;
    end;
    Menuenabled(false);
    if MessageDlg('Ungerichteten Graph untersuchen?',
       mtConfirmation, [mbYes, mbNo], 0) = mrYes
    then
      Hamiltongraph:=THamiltongraph(Graph.InhaltskopiedesGraphen
     (THamiltongraph,TInhaltsknoten,TInhaltskante,true))
    else
     Hamiltongraph:=THamiltongraph(Graph.InhaltskopiedesGraphen
     (THamiltongraph,TInhaltsknoten,TInhaltskante,false));
```

```
    Sliste:=TStringList.Create;
    GraphH:=Hamiltongraph;
    Aktiv:=false;
    Paintbox.OnMouseDown:=PanelDownMouse;
     Paintbox.Onmousemove:=nil;
     Paintbox.OnDblclick:=niL;
     Bildloeschen;
    Hamiltongraph.ZeichneGraph(Paintbox.Canvas);
   Hamiltongraph.Hamiltonkreise(Paintbox.Canvas,Ausgabe1,Sliste);
    StringlistnachListbox(Sliste,Listbox);
    GraphH:=Hamiltongraph.InhaltskopiedesGraphen
    (TInhaltsgraph,TInhaltsknoten,
    TInhaltsKante,false);;
    Menuenabled(true);
    Hamiltongraph.Freeall;
    Hamiltongraph:=nil;
    if Graph.Abbruch then
      ShowMessage('Abbruch!');
  except
    Menuenabled(true);
    AbbruchClick(Sender);
    ShowMessage('Fehler');
  end;
end;



procedure TKnotenformular.EulerliniegeschlossenClick(Sender:
TObject);
label Ende;
var Sliste:TStringList;
    Eulergraph:TEulergraph;
begin
  try
    if Graph.Leer then exit;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.AnzahlKomponenten>1 then
     begin
      ShowMessage('Mehrere Komponenten!');
      exit;
     end;
      Menuenabled(false);
      if MessageDlg('Ungerichteten Graph untersuchen?',
      mtConfirmation, [mbYes, mbNo], 0) = mrYes then
      begin
      if not Graph.GraphhatgeschlosseneEulerlinie(false) then
        begin
        ShowMessage('Der ungerichtete Graph hat keine ge
         schlossene Eulerlinie!');
```

```
          goto Ende;
        end
        else
        Eulergraph:=TEulergraph(Graph.InhaltskopiedesGraphen
        TEulergraph,TInhaltsknoten,TInhaltskante,true))
      end
      else
      begin
      if (not Graph.GraphhatgeschlosseneEulerlinie(false)) or
         (not Graph.GraphhatgeschlosseneEulerlinie(true)) then
        begin
        ShowMessage('Der gerichtete Graph hat keine geschlosse
          ne Eulerlinie! '+
        chr(13)+'(oder der Graph hat ungerichtete Kanten)');
          goto Ende;
        end
        else
        Eulergraph:=TEulergraph(Graph.InhaltskopiedesGraphen
        (TEulergraph,TInhaltsknoten,TInhaltskante,false));
      end;
      Sliste:=TStringList.Create;
       Bildloeschen;
      GraphH:=Eulergraph;
       Aktiv:=false;
      Eulergraph.ZeichneGraph(Paintbox.Canvas);
      Eulergraph.Eulerlinie
       (Paintbox.Canvas,Ausgabe1,Sliste,Eulergraph.letzterMausklickknoten,
      Eulergraph.letzterMausklickknoten);
      GraphH:=Eulergraph.InhaltskopiedesGraphen
      (TInhaltsgraph,TInhaltsknoten,
      TInhaltsKante,false);;
      StringlistnachListbox(Sliste,Listbox);
      Paintbox.OnMouseDown:=PanelDownMouse;
      Paintbox.ONmousemove:=nil;
      Paintbox.OnDblclick:=nil;
      Eulergraph.Freeall;
      Eulergraph:=nil;
       Ende:
      Menuenabled(true);
      if Graph.Abbruch then
       begin
         Aktiv:=true;
         Bildloeschen;
        Graph.zeichneGraph(Paintbox.Canvas);
        ShowMessage('Abbruch');
       end;
  except
    Menuenabled(true);
    AbbruchClick(Sender);
    ShowMessage('Fehler');
```

```
  end;
end;


procedure TKnotenformular.FrbbarkeitClick(Sender: TObject);
var AnzahlFarben:Integer;
    Farbgraph:TFarbgraph;
    Sliste:TStringList;
    Index:Integer;
begin
  try
    if Graph.Leer then exit;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    Menuenabled(false);
    Bildloeschen;
    BildzeichnenClick(Sender);
    Sliste:=TStringList.Create;
    Farbgraph:=TFarbgraph
    (Graph.InhaltskopiedesGraphen(TFarbgraph,TFarbknoten,
    TInhaltskante,false));
    GraphH:=TInhaltsgraph(Farbgraph);
    Aktiv:=false;
    Paintbox.OnMouseDown:=PanelDownMouse;
    Paintbox.Onmousemove:=nil;
    Paintbox.OnDblclick:=nil;
    Farbgraph.FaerbeGraph(Paintbox.Canvas,Ausgabe1,Sliste);
    Sliste.Insert(0,'Farben:');
    StringlistnachListbox(Sliste,Listbox);
    GraphH:=Farbgraph.InhaltskopiedesGraphen
    (TInhaltsgraph,TFarbknoten,TInhaltsKante,false);
    if Sliste.Count>1 then
     begin
      GraphH.Knotenwertposition:=2;
       Aktiv:=false;
      ShowMessage('Lösung: '+Sliste.strings[Sliste.Count-1]);
     end
     else
      ShowMessage('Keine Lösung!');
    Ausgabeloeschen(true);
    Menuenabled(true);
    Farbgraph.Freeall;
    Farbgraph:=nil;
    if Graph.Abbruch then
      ShowMessage('Abbruch!');
    Sliste.Free;
    Sliste:=nil;
  except
    Menuenabled(true);
    AbbruchClick(Sender);
```

```
    ShowMessage('Fehler');
  end;
end;



procedure TKnotenformular.EulerlinieoffenClick(Sender: TObject);
label Ende;
var Sliste:TStringList;
    Eulergraph:TEulergraph;
    Kno1,Kno2:TInhaltsknoten;
begin
  try
    if Graph.Leer then exit;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.AnzahlKomponenten>1 then
     begin
      ShowMessage('Mehrere Komponenten!');
      exit;
     end;
      Menuenabled(false);
      Eulergraph:=TEulergraph.Create;
      if MessageDlg('Ungerichteten Graph untersuchen?',
      mtConfirmation, [mbYes, mbNo], 0) = mrYes then
        begin
        Eulergraph:=TEulergraph(Graph.InhaltskopiedesGraphen
        (TEulergraph,TInhaltsknoten,TInhaltskante,true));
        if not Eulergraph.GraphhatoffeneEulerlinie
        (Tknoten(Kno1),TKnoten(Kno2),false)
          then
          begin
          ShowMessage('Der ungerichtete Graph hat keine
           offene Eulerlinie!');
            goto Ende;
        end
      end
      else
      begin
        Eulergraph:=TEulergraph
       (Graph.InhaltskopiedesGraphen(TEulergraph,
        TInhaltsknoten,TInhaltskante,false));
        if (not Eulergraph.GraphhatoffeneEulerlinie
        (TKnoten(Kno1),TKnoten(Kno2),false)) or
        (not Eulergraph.GraphhatoffeneEulerlinie
        (TKnoten(Kno1),TKnoten(Kno2),true) ) then
          begin
          ShowMessage('Der gerichtete Graph hat keine
          offene Eulerlinie! '+chr(13)+ '(oder der Graph hat
           ungerichtete Kanten)');
```

```
       goto Ende;
        end
      end;
      Sliste:=TStringList.Create;
      Bildloeschen;
      GraphH:=Eulergraph;
      Aktiv:=false;
     Paintbox.OnMouseDown:=PanelDownMouse;
      Paintbox.OnDblClick:=nil;
      Paintbox.ONmousemove:=nil;
      Eulergraph.zeichneGraph(Paintbox.Canvas);
      Eulergraph.Eulerlinie
      (Paintbox.Canvas,Ausgabe1,Sliste,Eulergraph.Graphknoten(Kno1),
      Eulergraph.Graphknoten(Kno2));
      StringlistnachListbox(Sliste,Listbox);

      GraphH:=Eulergraph.InhaltskopiedesGraphen
      (TInhaltsgraph,TInhaltsknoten,
      TInhaltsKante,false);;
      Ende:
      Menuenabled(true);
      if Graph.Abbruch then
      begin
        Aktiv:=true;
        Bildloeschen;
       Graph.ZeichneGraph(Paintbox.Canvas);
      end;
      if Graph.Abbruch then
        ShowMessage('Abbruch!');
      Eulergraph.Freeall;
      Eulergraph:=nil;
  except
    Menuenabled(true);
    AbbruchClick(Sender);
    ShowMessage('Fehler');
  end;
end;

procedure TKnotenformular.EndlicherAutomatClick(Sender:
TObject);
var Sliste:TStringList;
    Ende:Boolean;


procedure AutomatmitTAutomatengraph;
label Endproc;
var Automatengraph:TAutomatengraph;
begin
  if Graph.Leer then exit;
  if Graph.AnzahlKomponenten>1 then
```

```
begin
  ShowMessage('Mehrere Komponenten!');
  exit;
end;
Menuenabled(false);
Bildloeschen;
Graph.ZeichneGraph(Paintbox.Canvas);
if Graph.AnzahlungerichteteKanten>0 then
begin
  ShowMessage('Der Graph hat ungerichtete Kanten!');
  Menuenabled(true);
  exit;
end;
if Graph.AnzahlKnoten<2 then
begin
  ShowMessage('Graph hat nur einen Knoten!');
  Menuenabled(true);
  exit;
end;
Automatengraph:=TAutomatengraph
(Graph.InhaltskopiedesGraphen(TAutomatengraph,
TAutomatenknoten,TInhaltskante,false));
Automatengraph.SetzeAnfangswertknotenart;
GraphH:=Automatengraph;
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;
Automatengraph.MomentanerKnoten:=Automatengraph.BestimmeAnfangsknoten;
Automatengraph.ZeichneGraph(Paintbox.Canvas);
Eingabe.Visible:=true;
Button.Visible:=true;
Ausgabe1.Caption:='Kante eingeben:';
```

```
  Automatengraph.MomentaneKantenliste:=TKantenliste.Create;
  Eingabe.Setfocus;
  Endproc:
  GraphH:=Automatengraph;
  Aktiv:=false;
  if GraphH.Abbruch then
  begin
    Bildloeschen;
    Aktiv:=true;
    Graph.ZeichneGraph(Paintbox.Canvas);
    GraphH:=nil;
    ShowMessage('Abbruch!');
  end;
end;

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;
       Bildloeschen;
      Showmessage('Neuen Graph erzeugen.Danach Button Start an
       wählen!');
      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;
      Knotenformular.GraphladenClick(Sender);
       NeuerGraph:=false;
        Bildloeschen;
      AutomatenNeuGraph.zeichneGraph(Paintbox.Canvas);
  end
  else
  begin
    NeuerGraph:=true;
    Bildloeschen;
    Automatenneugraph:=TAutomatenneugraph
   (Graph.InhaltskopiedesGraphen(TAutomatenneugraph,
    TAutomatenknoten,TInhaltskante,false));
    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?
   (sonst Neubeginn!)',mtConfirmation,[mbYes,mbNo],0)=mrno))
    or (NeuerGraph)
 then
 begin
  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';
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;

begin
  try
    if (not Automatenneugraphaktiv) and
    (MessageDlg('Laden oder Speichern von
    Zwischenzuständen?',mtConfirmation,
    [mbYes,mbNo],0)=mrNo) then
      AutomatmitTautomatengraph
     else
     AutomatmitTAutomatenneugraph;
  except
    BildzeichnenClick(Sender);
    AbbruchClick(Sender);
    ShowMessage('Fehler');
  end;
end;


procedure TKnotenformular.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
  try
    if ord(key)=13 then
     begin
      Eingabe.Setfocus;
      Automatengraph:=TAutomatengraph(GraphH);
      if Automatengraph.Abbruch then
        begin
         Buttonclick(Sender);
         ShowMessage('Abbruch!');
          Menuenabled(true);
          goto Endproc;
        end;
       Str:=Eingabe.Text;
        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;
  end;
if (Ka2<>nil) and ((Ka2.Wert=Str) or (Ka2.Wert=' '+Str))
  then
  begin
  TKnoten(ZKno):=Ka2.Zielknoten
  (Automatengraph.MomentanerKnoten);
  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);
  Automatengraph.ZeichneGraph(Paintbox.Canvas);
   Automatengraph.Demopause;
   Ka2.Farbe:=clblack;
   Ka2.Stil:=pssolid;
  Ka2.Pfadrichtung:=ZKno;
  Automatengraph.MomentanerKnoten:=ZKno;
  Automatengraph.MomentanerKnoten.Farbe:=clred;
  Automatengraph.MomentanerKnoten.Stil:=psdot;
  T:=TInhaltsgraph(Automatengraph.MomentaneKantenliste.
    Kopie.Graph);
  Ausgabe2.Caption:='Zustandsfolge:
  '+T.InhaltallerKnoten(ErzeugeKnotenstring);
   Ausgabe2.Refresh;
   S:='Eingabefolge:
  '+T.InhaltallerKantenoderKnoten(ErzeugeKantenstring);
  Automatengraph.ZeichneGraph(Paintbox.Canvas);
   Eingabe.Text:='';
   if ZKno.Knotenart in [2,3] then
     begin
     if MessageDlg('Endzustand erreicht! Weiter?',
      mtConfirmation, [mbYes, mbNo], 0) = mrYes
       then goto Endproc;
      if Automatenneugraphaktiv
       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;
        Eingabe.Visible:=false;
        Button.Visible:=false;
         Listbox.Clear;
        Listbox.Items.Add(Ausgabe2.Caption);
        Listbox.Items.Add(S);
        Ausgabeloeschen(false);
        Graph.ZeichneGraph(Paintbox.Canvas);
         Aktiv:=true;
         Menuenabled(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:
    if not GraphH.Abbruch
    then
      GraphH.ZeichneGraph(Paintbox.Canvas)
    else
    begin
      Bildloeschen;
      Aktiv:=true;
      Graph.ZeichneGraph(Paintbox.Canvas);
      GraphH:=nil;
    end;
  except
    BildzeichnenClick(Sender);
    AbbruchClick(Sender);
    ShowMessage('Fehler');
  end;
end;
```

```
procedure TKnotenformular.ButtonClick(Sender: TObject);
begin
  try
    if Automatenneugraphaktiv
    then
    begin
      if Button.Caption=’Abbruch’ then
      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;
        Automatenneugraph:=nil;
      end;
      if Button.Caption=’Start’ then
      begin
        Button.Caption:=’Abbruch’;
        EndlicherAutomatClick(Sender);
        exit;
      end
    end;
    Eingabe.Visible:=false;
    Button.Visible:=false;
    Ausgabeloeschen(false);
    Graph.ZeichneGraph(Paintbox.Canvas);
    Aktiv:=true;
    Menuenabled(true);
  except
    BildzeichnenClick(Sender);
    AbbruchClick(Sender);
    ShowMessage(‘Fehler’);
  end;
end;


procedure TKnotenformular.GraphalsRelationClick(Sender:
TObject);
var Relationsgraph:TRelationsgraph;
    Reflexiv,Urspruenglich,Transitiv,Ordnung:Boolean;
    Sliste:TStringList;
    S:string;
begin
  try
    if Graph.Leer then exit;
    Bildloeschen;
```

```
Graph.ZeichneGraph(Paintbox.Canvas);
 if Graph.Leer then exit;
 if Graph.AnzahlungerichteteKanten>0 then
 begin
   ShowMessage('Ungerichtete Kanten!');
   exit;
 end;
 Menuenabled(false);
 Bildloeschen;
 BildzeichnenClick(Sender);
 Sliste:=TStringList.Create;
 Relationsgraph:=TRelationsgraph.Create;
 Relationsgraph:=TRelationsgraph
 (Graph.InhaltskopiedesGraphen(TRelationsgraph,
 TRelationsknoten,TInhaltskante,false));
 GraphH:=TInhaltsgraph(Relationsgraph);
 Aktiv:=false;
 if MessageDlg('Reflexivität  untersuchen und danach
 reflexive Hülle
 erzeugen?',mtConfirmation,[mbYes,mbNo],0)=mryes
  then
   Reflexiv:=true
  else
   Reflexiv:=false;
 if MessageDlg('Transitivität untersuchen und danach
 transitive Hülle
 erzeugen?',mtConfirmation,[mbYes,mbNo],0)=mryes
  then
   Transitiv:=true
 else
   Transitiv:=false;
if MessageDlg('Ursprünglichen Graph
wiederherstellen?',mtConfirmation,[mbYes,mbNo],0)=mryes
 then
   Urspruenglich:=true
 else
   Urspruenglich:=false;
 Bildzeichnenclick(Sender);
 Paintbox.OnMouseDown:=PanelDownMouse;
 Paintbox.OnDblClick:=nil;
 Paintbox.ONmousemove:=nil;
 if not  Graph.GraphhatKreise then
 begin
  Relationsgraph.ErzeugeOrdnung(Paintbox.Canvas);
  Relationsgraph.ErzeugeErgebnis(Sliste);
  Relationsgraph.FaerbeGraph(clblack,pssolid);
  S:='Die reflexive und transitive  Hülle der Relation ist
  antisymmetrisch!';
   Sliste.Add(S);
 end
```

```
else
   begin
     ShowMessage('Der Graph hat Kreise bzw. die
       Relation'+chr(13)+
      'ist nicht antisymmetrisch!'+chr(13)+
      'Deshalb gibt es keine Ordnungsrelation!');
       Sliste.Add
      ('Die reflexive und transitive Hülle der Relation
      ist nicht antisymmetrisch!');
   end;
  if Reflexiv then Relationsgraph.Schlingenerzeugen(Sliste);
  if Transitiv then Relationsgraph.Warshall(Sliste);
  if Relationsgraph.Relationistsymmetrisch
   then
   begin
     S:='Die ursprüngliche Relation ist symmetrisch!';
     Sliste.Add(S);
   end
   else
   begin
     S:='Die ursprüngliche Relation ist nicht symmetrisch!';
     Sliste.Add(S);
   end;
   if  Urspruenglich then
   begin
     GraphH:=TInhaltsgraph(Relationsgraph);
     Aktiv:=false;
   end
   else
   begin
     Graph.Freeall;
     Graph:=TInhaltsgraph.Create;
  Graph:=Relationsgraph.InhaltskopiedesGraphen(TInhaltsgraph,TInhaltsknoten,
     TInhaltskante,false);
     Graph.FaerbeGraph(clblack,pssolid);
   end;
   StringlistnachListbox(Sliste,Listbox);
   Sliste.Free;
   Sliste:=nil;
   Bildzeichnenclick(Sender);
   if not  Graph.GraphhatKreise then
     Relationsgraph.Knotenwertposition:=2;
   Relationsgraph.ZeichneGraph(Paintbox.Canvas);
   GraphH:=Relationsgraph.InhaltskopiedesGraphen
   (TInhaltsgraph,TRelationsknoten,
   TInhaltsKante,false);;
   if not  Graph.GraphhatKreise then
     GraphH.Knotenwertposition:=2;
   Aktiv:=false;
   Menuenabled(true);
```

```
    Relationsgraph.Freeall;
    Relationsgraph:=nil;
    if GraphH.Abbruch then
    begin
      Aktiv:=true;
      Bildloeschen;
      Graph.zeichneGraph(Paintbox.Canvas);
      ShowMessage('Abbruch!');
    end;
  except
    AbbruchClick(Sender);
    Fehler;
  end;
end;


procedure TKnotenformular.MaximalerNetzflussClick(Sender:
TObject);
var Sliste:TStringList;
    MaxFlussgraph:TMaxFlussgraph;
    Kant:TKantenliste;
    Ende:Boolean;
    Startknoten,Zielknoten:TMaxflussknoten;
    Gesamtfluss:Extended;
begin
  try
    if Graph.Leer then exit;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.AnzahlKomponenten>1 then
    begin
      ShowMessage('Mehrere Komponenten!');
      exit;
    end;
    if Graph.AnzahlKnoten<2 then
    begin
      ShowMessage('Nur ein Knoten!');
      exit;
    end;
    if Graph.AnzahlungerichteteKanten>0 then
    begin
      ShowMessage('Ungerichtete Kanten');
      exit;
    end;
    Menuenabled(false);
    MaxFlussgraph:=TMaxflussgraph
    (Graph.InhaltskopiedesGraphen(TMaxflussgraph,
    TMaxflussknoten,TMaxflusskante,false));
    MaxFlussgraph.FaerbeGraph(clblack,pssolid);
    MaxFlussgraph.ZeichneGraph(Paintbox.Canvas);
```

```
    GraphH:=TInhaltsgraph(MaxFlussgraph);
    Paintbox.OnMouseDown:=PanelDownMouse;
    Paintbox.OnDblClick:=nil;
    Paintbox.ONmousemove:=nil;
     MaxFlussgraph.StartFluss(Paintbox.Canvas,Gesamtfluss,Knotenformular);
    if Gesamtfluss>0 then
     begin
      Sliste:=TStringList.Create;
      MaxFlussgraph.BestimmeErgebnis(Sliste);
      Sliste.Insert(0,'Kanten,Schranken und Fluss:');
      Sliste.Add('maximaler Gesamtfluss: '+
      RundeZahltoString(Gesamtfluss,Graph.Kantengenauigkeit));
      if not Graph.Abbruch then
       begin
        StringlistnachListbox(Sliste,Listbox);
        ShowMessage('Der maximale Gesamtfluss beträgt '+
        RundeZahltoString(Gesamtfluss,Graph.Kantengenauigkeit));
        end;
       Bildloeschen;
      MaxFlussgraph.Knotenwertposition:=0;
      MaxFlussgraph.Kantenwertposition:=2;
      MaxFlussgraph.ZeichneGraph(Paintbox.Canvas);
      GraphH:=TInhaltsgraph(MaxFlussgraph);
      GraphH.Knotenwertposition:=0;
      GraphH.Kantenwertposition:=2;
       Aktiv:=false;
      GraphH:=MaxFlussgraph.
      InhaltskopiedesGraphen(TInhaltsgraph,TMaxflussknoten,
      TMaxflusskante,false);
      GraphH.Kantenwertposition:=2;
       Aktiv:=false;
       Sliste.Free;
       Sliste:=nil;
      MaxFlussgraph.Freeall;
      MaxFlussgraph:=nil;
    end else begin Aktiv:=true;Bildloeschen;Graph.ZeichneGraph
(Paintbox.Canvas);Showmessage('Der Gesamtfluss ist Null!Keine Lösung');end
    Menuenabled(true);if GraphH.Abbruch then
    begin
      Aktiv:=true;
      Bildloeschen;
      Graph.ZeichneGraph(Paintbox.Canvas);
      ShowMessage('Abbruch!');
    end;
  except
    AbbruchClick(Sender);
    Fehler;
  end;
end;
```

```
procedure TKnotenformular.MaximalesMatchingClick(Sender:
TObject);
label Endproc;
var Matchgraph:TMatchgraph;
    Sliste:TStringList;
    Anfangsmatching:Boolean;
begin
  try
    if Graph.Leer then exit;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if MessageDlg('AnfangsMatching?',
    mtConfirmation,[mbYes,mbNo],0)=mryes
     then
      Anfangsmatching:=true
     else
      Anfangsmatching:=false;
    Menuenabled(false);
    Bildloeschen;
    BildzeichnenClick(Sender);
    Sliste:=TStringList.Create;
    Matchgraph:=TMatchgraph.Create;
    Matchgraph:=TMatchgraph(Graph.InhaltskopiedesGraphen
    (TMatchgraph,TMatchknoten,
    TInhaltskante,false));
    GraphH:=TInhaltsgraph(Matchgraph);
    Aktiv:=false;
    Matchgraph.InitialisierealleKnoten;
    GraphH:=Matchgraph;
    Paintbox.OnMouseDown:=PanelDownMouse;
    Paintbox.OnDblClick:=nil;
    Paintbox.ONmousemove:=nil;
    if Anfangsmatching then
     begin
      Matchgraph.ErzeugeAnfangsmatching;
      Bildloeschen;
      Matchgraph.ZeichneGraph(Paintbox.Canvas);
      ShowMessage('Anfangsmatching Kantenzahl: '+Integertostring
      ((Matchgraph.AnzahlKnoten-
      Matchgraph.AnzahlexponierteKnoten)DIV 2));
     end;
   GraphH:=Matchgraph.InhaltskopiedesGraphen
   (TInhaltsgraph,TMatchknoten,
    TInhaltsKante,false);;
    if GraphH.Abbruch then
     begin
      Menuenabled(true);
      Ausgabeloeschen(true);
      exit;
     end;
```

```
     Matchgraph.BestimmeMaximalesMatching(Paintbox.Canvas,Ausgabe1,Graph);
    Bildloeschen;
    Matchgraph.ZeichneGraph(Paintbox.Canvas);
    if GraphH.Abbruch then
     begin
      Menuenabled(true);
      Ausgabeloeschen(true);
      goto Endproc;
     end;
    Sliste.Add('Kanten des maximalen Matchings:');
    Matchgraph.ErzeugeListe(Sliste);
   StringlistnachListbox(Sliste,Listbox);
    if GraphH.Demo then Ausgabe1.Caption:='Maximales Matching
     erreicht!';
    if GraphH.Demo then Ausgabe1.Refresh;
    GraphH.Demopause;
    ShowMessage('Kantenzahl des maximalen
    Matchings: '+Integertostring
    ((Matchgraph.AnzahlKnoten-Matchgraph.
    AnzahlexponierteKnoten)DIV 2));
    if Matchgraph.AnzahlexponierteKnoten=0 then ShowMessage('Das
    Matching ist perfekt!');
    Ausgabeloeschen(true);
    Menuenabled(true);
    Aktiv:=false;
     Endproc:
    Sliste.Free;
    Sliste:=nil;
    Matchgraph.Freeall;
    Matchgraph:=nil;
    if GraphH.Abbruch then
     begin
      Aktiv:=true;
      Bildloeschen;
      Graph.zeichneGraph(Paintbox.Canvas);
      ShowMessage('Abbruch!');
     end;
  except
    AbbruchClick(Sender);
    Fehler;
  end;
end;


procedure TKnotenformular.GleichungssystemClick(Sender:
TObject);
label Endproc;
var Sliste:TStringList;
    Gleichungsgraph:TGleichungssystemgraph;
    T:TInhaltsgraph;
```

```
    Kno,Kno1:TInhaltsKnoten;
    AnzahlKnoten:Integer;
    Ende,Beenden:Boolean;
    Zaehl:Integer;
    St:string;
    Loesung:Extended;
    Oberflaeche:TForm;
    G:TInhaltsgraph;
begin
  try
    if Graph.Leer then exit;
    Menuenabled(false);
    Beenden:=false;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.Leer then exit;
    if (Graph.AnzahlungerichteteKanten>0) or
    (Graph.AnzahlparallelerKanten>0) then
     begin
      ShowMessage('Der Graph hat ungerichtete Kanten oder
      Parallelkanten!');
      Menuenabled(true);
      exit;
     end;
    Sliste:=TStringList.Create;
    Ende:=false;
    T:=Graph.InhaltskopiedesGraphen
    (TInhaltsgraph,TGleichungssystemknoten,TInhaltskante,false);
     repeat
      Gleichungsgraph:=TGleichungssystemgraph
      (Graph.InhaltskopiedesGraphen
      (TGleichungssystemgraph,TGleichungssystemknoten,
      TInhaltskante,false));
      GraphH:=Graph.InhaltskopiedesGraphen(TInhaltsgraph,
      TInhaltsknoten,TInhaltskante,false);
      Gleichungsgraph.ErgaenzeKanten;
      Gleichungsgraph.ErgaenzeSchlingen;
      Gleichungsgraph.NumeriereKnoten;
      Gleichungsgraph.ZeichneGraph(Paintbox.Canvas);
      GraphH:=TInhaltsgraph(Gleichungsgraph);
      if GraphH.Abbruch or beenden then
       begin
         Bildloeschen;
         Aktiv:=true;
       Graph.ZeichneGraph(Paintbox.Canvas);
        Ausgabeloeschen(false);
         goto Endproc;
      end;
      Ausgabe1.Caption:='Knoten mit Mausklick bestimmen!';
      Ausgabe1.Refresh;
```

```
ShowMessage('Knoten mit Mausklick bestimmen');
  repeat
  GraphH:=TInhaltsgraph(Gleichungsgraph);
   if GraphH.Abbruch or Beenden then
     begin
      Bildloeschen;
      Aktiv:=true;
    Graph.ZeichneGraph(Paintbox.Canvas);
     Ausgabeloeschen(false);
      Menuenabled(true);
       goto Endproc;;
      end;
    GraphH.Zustand:=false;
   Paintbox.OnMouseDown:=GleichungssystemMouseDown;
    Paintbox.OnDblClick:=nil;
    Paintbox.ONmousemove:=nil;
      repeat
       Menuenabled(false);
     Ausgabe1.Caption:='Knoten durch Mausklick bestim
        men!';
      if GraphH.Abbruch or Beenden then
        begin
          Bildloeschen;
          Aktiv:=true;
        Graph.ZeichneGraph(Paintbox.Canvas);
          Ausgabeloeschen(false);
           goto Endproc;
        end;
     Application.Processmessages;
    Paintbox.OnMouseDown:=GleichungssystemMouseDown;
      Paintbox.OnDblClick:=nil;
      Paintbox.ONmousemove:=nil;
        if Aktiv then
         begin
           Aktiv:=false;
           Bildloeschen;
        Gleichungsgraph.ZeichneGraph(Paintbox.Canvas);
          end
 until GraphH.Zustand;
 Menuenabled(true);
Kno:=GraphH.LetzterMausklickknoten;
  St:='';
Kno1:=TInhaltsknoten.Create;
 Kno1.X:=Kno.X;
 Kno1.Y:=Kno.Y;
Oberflaeche:=TForm(Self);
  G:=Graph;
Gleichungsgraph.EliminiereKnoten
(TKnoten(Kno),Paintbox.Canvas,Ausgabe1,Loesung,
 Beenden,G,Oberflaeche);
```

```
    if Gleichungsgraph.AnzahlKnoten=0 then
      begin
        St:='Lösung:
      x'+Inttostr(TGleichungssystemknoten(Kno).Nummer)
        +' = '+Rundezahltostring
      (Loesung,Graph.Knotengenauigkeit);
      Sliste.Add('x'+Inttostr(TGleichungssystemKnoten
        (Kno).Nummer)+' = '+
      Rundezahltostring(Loesung,Graph.Knotengenauigkeit));
      TGleichungssystemknoten(T.Graphknoten(Kno1)).Ergebnis:=
      Rundezahltostring(Loesung,Graph.Knotengenauigkeit);
      end;
     Bildloeschen;
    Gleichungsgraph.ZeichneGraph(Paintbox.Canvas);
    GraphH:=TInhaltsgraph(Gleichungsgraph);
     Aktiv:=false;
     if St<>'' then
      begin
      Graph.ZeichneGraph(Paintbox.Canvas);
        GraphH:=Graph;
        if not beenden then ShowMessage(St);
      end;
   GraphH:=Gleichungsgraph.InhaltskopiedesGraphen
   (TInhaltsgraph,TInhaltsknoten,TInhaltsKante,false);;
     if Beenden then
      begin
      ShowMessage('Keine oder keine eindeutige Lösung!');
        Bildloeschen;
      Graph.ZeichneGraph(Paintbox.Canvas);
        Aktiv:=true;
      end
    until (Gleichungsgraph.AnzahlKnoten=0) or beenden;
    Ausgabe1.Caption:='Lösung:';
     Ausgabe1.Refresh;
    StringlistnachListbox(Sliste,Listbox);
     Ausgabe2.Caption:='';
     for Zaehl:=0 to Sliste.Count-1 do
      Ausgabe2.Caption:=Ausgabe2.Caption+'
       '+Sliste.Strings[Zaehl];
     Ausgabe2.Refresh;
    Graph.ZeichneGraph(Paintbox.Canvas);
      if not Beenden then
     if MessageDlg('Weitere Lösung bestimmen?',
     mtConfirmation, [mbYes, mbNo], 0) = mrNo then
      Ende:=true;
 until Ende or beenden;
Ausgabeloeschen(false);
GraphH:=Graph;
Bildloeschen;
GraphH:=T;
```

```
     GraphH.Knotenwertposition:=2;
     Graph.Abbruch:=GraphH.Abbruch;
      Endproc:
      if not GraphH.Abbruch
      then
        GraphH.ZeichneGraph(Paintbox.Canvas)
      else
      begin
        Aktiv:=true;
        Bildloeschen;
        Graph.ZeichneGraph(Paintbox.Canvas);;
        GraphH:=nil;
      end;
      Menuenabled(true);
      Sliste.Free;
      Sliste:=nil;
      Gleichungsgraph.Freeall;
      Gleichungsgraph:=nil;
      if G.Abbruch then
        ShowMessage('Abbruch!');
  except
    AbbruchClick(Sender);
    Fehler;
  end;
end;



procedure TKnotenformular.MarkovketteabsClick(Sender: TObject);
var Sliste:TStringList;
    Markovgraph:TMarkovgraph;
begin
  try
    if Graph.Leer then exit;
    Menuenabled(false);
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.Leer then exit;
    if Graph.AnzahlungerichteteKanten>0 then
     begin
      ShowMessage('Der Graph hat ungerichtete Kanten!');
      Menuenabled(true);
       exit;
    end;
    Sliste:=TStringList.Create;
    Markovgraph:=TMarkovgraph(Graph.InhaltskopiedesGraphen
    (TMarkovgraph,TMarkovknoten,
    TMarkovkante,false));
    GraphH:=Markovgraph;
    Aktiv:=false;
```

```
   Markovgraph.Markovkette(Paintbox.Canvas,Ausgabe1,Ausgabe2,SListe,TForm(self
    if GraphH.Abbruch then
     begin
      Menuenabled(true);
      ShowMessage('Abbruch');
       exit;
     end;
    StringlistnachListbox(Sliste,Listbox);
    Paintbox.OnDblClick:=nil;
    Paintbox.ONmouseMove:=nil;
    Paintbox.OnMouseDown:=MarkovMouseDown;
    GraphH:=Markovgraph.InhaltskopiedesGraphen
    (TInhaltsgraph,TMarkovknoten,
    TMarkovKante,false);
    if Sliste.count>0 then
      GraphH.Knotenwertposition:=8;
    Bildloeschen;Graph.Knotengenauigkeit:=GraphH.Knotengenauigkeit;
    GraphH.ZeichneGraph(Paintbox.Canvas);
    Aktiv:=false;
    Menuenabled(true);
    Sliste.Free;
    Sliste:=nil;
    Markovgraph.Freeall;
    Markovgraph:=nil;
    if Graph.Abbruch then
      ShowMessage('Abbruch!');
  except
    AbbruchClick(Sender);
    Fehler;
  end;
end;


procedure TKnotenformular.MarkovkettestatClick(Sender: TObject);
var Sliste:TStringList;
    Markovgraph:TMarkovstatgraph;
begin
  try
    if Graph.Leer then exit;
    Menuenabled(false);
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.Leer then exit;
    if (Graph.AnzahlungerichteteKanten>0)or
    (TMarkovstatgraph(Graph).AnzahlRandknoten>0) then
     begin
      ShowMessage('Der Graph hat ungerichtete Kanten oder Rand
       knoten!');
      Menuenabled(true);
      exit;
```

```
    end;
    Sliste:=TStringList.Create;
    Markovgraph:=TMarkovstatgraph(Graph.InhaltskopiedesGraphen
    (TMarkovstatgraph,TMarkovknoten,TMarkovkante,false));
    GraphH:=Markovgraph;
    Markovgraph.Markovkettestat
    (Paintbox.Canvas,Ausgabe1,Ausgabe2,SListe,TForm(self));
    if GraphH.Abbruch then
     begin
      Menuenabled(true);
       exit;
     end;
    StringlistnachListbox(Sliste,Listbox);
    Paintbox.OnDblClick:=nil;
    Paintbox.ONmousemove:=nil;
    Paintbox.OnMouseDown:=MarkovMouseDown;
    GraphH:=Markovgraph.InhaltskopiedesGraphen
    (TInhaltsgraph,TMarkovknoten,
    TMarkovKante,false);Graph.Knotengenauigkeit:=GaphH.Knotengenauigkeit;
    if Sliste.Count>0 then
      GraphH.Knotenwertposition:=8;
    Bildloeschen;
    GraphH.ZeichneGraph(Paintbox.Canvas);
    Menuenabled(true);
    Sliste.Free;
    Sliste:=nil;
    Markovgraph.Freeall;
    Markovgraph:=nil;
    if Graph.Abbruch then
      ShowMessage('Abbruch!');
  except
    AbbruchClick(Sender);
    Fehler;
  end;
end;


procedure TKnotenformular.GraphreduzierenClick(Sender: TObject);
label Fertig,Aus;
var Sliste:TStringList;
    Markovgraph:TMarkovreduzieregraph;
    T:TInhaltsgraph;
    Kno,Kno1:TInhaltsKnoten;
    AnzahlKnoten:Integer;
    Ende,Beenden:Boolean;
    Zaehl:Integer;
    St:string;
    Loesung:Extended;
    Markov:Boolean;
    Oberflaeche:TForm;
```

```
    Graph:TInhaltsgraph;

begin
  try
    Graph:=self.Graph;
    if Graph.Leer then exit;
    Menuenabled(false);
    Beenden:=false;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.Leer then exit;
    if Graph.AnzahlungerichteteKanten>0 then
    begin
      ShowMessage('Der Graph hat ungerichtete Kanten!');
      Menuenabled(true);
      exit;
    end;
    if MessageDlg('Graph als Markovkette?',
         mtConfirmation, [mbYes, mbNo], 0) = mrYes
    then
      Markov:=true
    else
      Markov:=false;
    Sliste:=TStringList.Create;
     Ende:=false ;
    T:=Graph.InhaltskopiedesGraphen
    (TInhaltsgraph,TMarkovreduziereknoten,TInhaltskante,false);
     repeat
      Ausgabe1.Caption:='Berechnung läuft';
    Markovgraph:=TMarkovreduzieregraph(Graph.InhaltskopiedesGraphen
      (TMarkovreduzieregraph,TMarkovreduziereknoten,
      TMarkovkante,false));
        GraphH:=Graph.InhaltskopiedesGraphen(TInhaltsgraph,TInhaltsknoten,
      TInhaltskante,false);;
       if Markov then
        if Markovgraph.Fehler then
        begin
         ShowMessage('Zuerst Fehler beheben!');
          Menuenabled(true);
           goto Aus;
        end;
      GraphH:=Markovgraph;
      Oberflaeche:=TForm(self);
      Markovgraph.GraphInit(Markov,Paintbox.Canvas,
      Oberflaeche,Graph,Ausgabe2);
      GraphH:=TInhaltsgraph(Markovgraph);
      if GraphH.Abbruch or Beenden then
        begin
         Bildloeschen;
         Aktiv:=true;
```

```
 Graph.ZeichneGraph(Paintbox.Canvas);
  Ausgabeloeschen(false);
   goto Fertig;
end;
Ausgabe1.Caption:=’Knoten mit Mausklick bestimmen!’;
Ausgabe1.Refresh;
ShowMessage(‘Knoten mit Mausklick bestimmen’);
repeat
  GraphH:=TInhaltsgraph(Markovgraph);
  if GraphH.Abbruch or Beenden then
  begin
    Bildloeschen;
    Aktiv:=true;
  Graph.ZeichneGraph(Paintbox.Canvas);
   Ausgabeloeschen(false);
    Menuenabled(true);
    goto Fertig;
  end;
  GraphH.Zustand:=false;
  Paintbox.OnMouseDown:=GleichungssystemMouseDown;
  Paintbox.OnDblClick:=nil;
  Paintbox.ONmousemove:=nil;
  repeat
    Menuenabled(false);
    Ausgabe1.Caption:=’Knoten durch Mausklick bestimmen!’;
    if GraphH.Abbruch or beenden then
    begin
      Bildloeschen;
      Aktiv:=true;
      Graph.ZeichneGraph(Paintbox.Canvas);
      Ausgabeloeschen(false);
      goto Fertig;
    end;
    Application.Processmessages;
    Paintbox.OnMouseDown:=GleichungssystemMouseDown;
    Paintbox.OnDblClick:=nil;
    Paintbox.OnMousemove:=nil;
    if Aktiv and (not GraphH.Abbruch) then
    begin
      Aktiv:=false;
      Bildloeschen;
      Markovgraph.ZeichneGraph(Paintbox.Canvas);
    end;
    GraphH:=Markovgraph;
    if GraphH.Abbruch
    then
    begin
      Bildloeschen;
      Graph.ZeichneGraph(Paintbox.Canvas);
      goto Fertig;
```

```
      end;
    until GraphH.Zustand;
    Menuenabled(true);
    GraphH:=Markovgraph;
   Kno:=GraphH.LetzterMausklickknoten;
     St:='';
   Oberflaeche:=TForm(self);
   Kno1:=TInhaltsknoten.Create;
    Kno1.X:=Kno.X;
    Kno1.Y:=Kno.Y;
   Markovgraph.LoescheKnotengraphreduzieren
   (TKnoten(Kno),Paintbox.Canvas,Sliste,
   Ausgabe1,Ausgabe2,Loesung,beenden,
    Graph,Oberflaeche);
     Bildloeschen;
    Markovgraph.ZeichneGraph(Paintbox.Canvas);
    GraphH:=TInhaltsgraph(Markovgraph);
    Aktiv:=false;
    if Markovgraph.Abbruch then goto Fertig;
    if Markovgraph.AnzahlKnoten=0 then
    begin
      if Markov then
      begin
       St:='Wahrscheinlichkeit:
       w'+Graph.Graphknoten(Kno1).Wert+' = '+
      Rundezahltostring(Loesung,Graph.Knotengenauigkeit);
      Sliste.Add('w'+Graph.Graphknoten(Kno1).Wert+' = '+
      Rundezahltostring(Loesung,Graph.Knotengenauigkeit));
      end
      else
      begin
      St:='Fluss: Markovgraph'+Graph.Graphknoten(Kno1).
         Wert+' = '+
      Rundezahltostring(Loesung,Graph.Knotengenauigkeit);
      Sliste.Add('Markovgraph'+Graph.Graphknoten(Kno1).Wert+'
        = '+Rundezahltostring
      (Loesung,Graph.Knotengenauigkeit));
      end;
   TMarkovreduziereknoten(T.Graphknoten(Kno1)).Ergebnis:=
   Rundezahltostring(Loesung,Graph.Knotengenauigkeit);
   end;
 Bildloeschen;
Markovgraph.ZeichneGraph(Paintbox.Canvas);
GraphH:=TInhaltsgraph(Markovgraph);
 Aktiv:=false;
Ausgabe2.Caption:='';
Ausgabe2.Refresh;
 if St<>'' then
 begin
 Graph.ZeichneGraph(Paintbox.Canvas);
```

```
      GraphH:=Graph;
     if not Beenden then ShowMessage(St);
     end;
   GraphH:=TInhaltsgraph(Markovgraph);
    if Beenden then
     begin
     ShowMessage('Keine oder keine eindeutige Lösung!');
       Bildloeschen;
     Graph.ZeichneGraph(Paintbox.Canvas);
       Aktiv:=true;
     end;
   if GraphH.Abbruch then goto Fertig;
 until (Markovgraph.AnzahlKnoten=0) or Beenden;
 Ausgabe1.Caption:='Lösung:';
 Ausgabe1.Refresh;
 StringlistnachListbox(Sliste,Listbox);
 Ausgabe2.Caption:='';
 for Zaehl:=0 to Sliste.Count-1 do
  Ausgabe2.Caption:=Ausgabe2.Caption+'
  '+Sliste.Strings[Zaehl];
 Ausgabe2.Refresh;
 Graph.ZeichneGraph(Paintbox.Canvas);
  if not beenden then
   if MessageDlg('Weitere Lösung bestimmen?',
   mtConfirmation, [mbYes, mbNo], 0) = mrNo then Ende:=true;
   if GraphH.Abbruch then goto Fertig;
until Ende or Beenden;
GraphH:=Graph;
 Fertig:
Ausgabeloeschen(false);
Bildloeschen;
if not GraphH.Abbruch then
 begin
   GraphH:=T;
  GraphH.Knotenwertposition:=2;
 end;
Graph.Abbruch:=GraphH.Abbruch;
if not GraphH.Abbruch
 then
   GraphH.ZeichneGraph(Paintbox.Canvas)
 else
 begin
   Aus:
  Ausgabeloeschen(false);
  Graph.Knotenwertposition:=0;
   Aktiv:=true;
   Bildloeschen;
  Graph.ZeichneGraph(Paintbox.Canvas);
  Paintbox.OnMouseDown:=PaintboxMousedown;
  Paintbox.OnDblClick:=PaintboxDblclick;
```

```
     Paintbox.OnMousemove:=PaintboxMousemove;
      GraphH:=nil;
     end;
    Menuenabled(true);
    Sliste.Free;
    Sliste:=nil;
    Markovgraph.Freeall;
    Markovgraph:=nil;
    if Graph.Abbruch then
      ShowMessage('Abbruch!');
  except
    AbbruchClick(Sender);
    Fehler;
  end;
end;


procedure TKnotenformular.MinimaleKostenClick(Sender: TObject);
label Endproc;
var Sliste:TStringList;
    Kostengraph:TMinimaleKostengraph;
    Startknoten,Zielknoten:TMaxflussknoten;
    Gesamtkosten:Real;
    Quelle,Senke:TInhaltsknoten;
    Ganzzahlig,Maximal:Boolean;
    Oberflaeche:TForm;
    Graph:TInhaltsgraph;

begin
  try
    Graph:=self.Graph;
    if Graph.Leer then exit;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.AnzahlKomponenten>1 then
     begin
      ShowMessage('Mehrere Komponenten!');
      exit;
     end;
    if Graph.AnzahlKnoten<2 then
     begin
      ShowMessage('Graph hat nur einen Knoten!');
       exit;
     end;
    if Graph.AnzahlungerichteteKanten>0 then
     begin
      ShowMessage('Der Graph hat ungerichtete Kanten');
        exit;
    end;
   Kostengraph:=TMinimaleKostengraph
```

```
(Graph.InhaltskopiedesGraphen(TMinimaleKostengraph,
TInhaltsknoten,TMinimaleKostenKante,false));
if Kostengraph.BestimmeQuelleundSenke
(Quelle,Senke,Paintbox.Canvas,true)=false then
 begin
  Kostengraph.Freeall;
  Kostengraph:=nil;
   exit;
 end;
if MessageDlg('Maximale Kosten (statt
minimale Kosten)?',mtConfirmation,[mbYes,mbNo],0)=mryes
 then
   Maximal:=true
 else
  Maximal:=false;
if MessageDlg('Nur ganzzahlige
Rechnung?',mtConfirmation,[mbYes,mbNo],0)=mryes
 then
  Ganzzahlig:=true
 else
  Ganzzahlig:=false;
Sliste:=TStringList.Create;
Menuenabled(false);
Kostengraph.ZeichneGraph(Paintbox.Canvas);
GraphH:=TInhaltsgraph(Kostengraph);
Kostengraph.Ganzzahlig:=Ganzzahlig;
Oberflaeche:=TForm(self);
Ausgabe1.Caption:='Eingabe der Kantenkosten';
Ausgabe1.Refresh;
ShowMessage('Eingabe der Kosten der
Kanten!'+chr(13)+chr(13)+
'Kanten mit Mausklick bestimmen!');
Bildloeschen;
GraphH.Kantenwertposition:=2;
GraphH.ZeichneGraph(Paintbox.Canvas);
 repeat
  GraphH.Zustand:=false;
  Paintbox.OnMouseDown:=KostenMousedown;
  Paintbox.ONmousemove:=nil;
  Paintbox.OnDblclick:=nil;
   repeat
    Application.Processmessages;
    if GraphH.Abbruch then goto Endproc;
   until GraphH.Zustand;
   Bildloeschen;
  GraphH.zeichneGraph(Paintbox.Canvas);
until MessageDlg('Keine Kante! Weitere Kosten eingeben?',
mtConfirmation, [mbYes, mbNo], 0) = mrNo;
Paintbox.OnMouseDown:=PanelDownMouse;
Ausgabeloeschen(true);
```

```
    ShowMessage('Kosten der Kanten');
    GraphH.Kantenwertposition:=0;
    Kostengraph.MinimaleKosten
    (Paintbox.Canvas,Graph,Oberflaeche,Ausgabe1,
    Maximal,Quelle,Senke,'m',Sliste);
    StringlistnachListbox(Sliste,Listbox);
     Endproc:
    GraphH.Kantenwertposition:=0;
    Ausgabeloeschen(true);
    GraphH:=Kostengraph.InhaltskopiedesGraphen
    (TInhaltsgraph,TInhaltsknoten,
    TMinimalekostenKante,false);;
    GraphH.Kantenwertposition:=1;
    Aktiv:=false;
    Menuenabled(true);
    if GraphH.Abbruch then
     begin
      Aktiv:=true;
      Bildloeschen;
      Graph.ZeichneGraph(Paintbox.Canvas);
     end;
    Sliste.Free;
    SListe:=nil;
    Kostengraph.Freeall;
    Kostengraph:=nil;
    if Graph.Abbruch then
      ShowMessage('Abbruch!');
  except
    AbbruchClick(Sender);
    Fehler;
  end;
end;


procedure TKnotenformular.TransportproblemClick(Sender:
TObject);
label Endproc;
var Sliste:TStringList;
    Kostengraph:TMinimaleKostengraph;
    Startknoten,Zielknoten:TMaxflussknoten;
    Gesamtkosten:Real;
    Quelle,Senke:TInhaltsknoten;
    Ganzzahlig,Maximal,Hitch:Boolean;
    Fluss:Real;
    Flussok:Boolean;
    Oberflaeche:TForm;
    Graph:TInhaltsgraph;
begin
  try
    Graph:=self.Graph;
```

```
if Graph.leer then exit;
Bildloeschen;
Graph.ZeichneGraph(Paintbox.Canvas);
if Graph.AnzahlKomponenten>1 then
 begin
  ShowMessage('Mehrere Komponenten!');
    exit;
 end;
if Graph.AnzahlKnoten<2 then
 begin
  ShowMessage('Graph hat nur einen Knoten!');
   exit;
 end;
if Graph.AnzahlungerichteteKanten>0 then
 begin
  ShowMessage('Der Graph hat ungerichtete Kanten');
    exit;
 end;
if MessageDlg('Maximale Kosten (statt
minimale Kosten)?',mtConfirmation,[mbYes,mbNo],0)=mryes
 then
   Maximal:=true
 else
   Maximal:=false;
if MessageDlg('Nur ganzzahlige
Rechnung?',mtConfirmation,[mbYes,mbNo],0)=mryes
 then
  Ganzzahlig:=true
 else
  Ganzzahlig:=false;
if MessageDlg('Hitchcockproblem
lösen??',mtConfirmation,[mbYes,mbNo],0)=mryes
 then
   Hitch:=true
 else
   Hitch:=false;
Sliste:=TStringList.Create;
Menuenabled(false);
Kostengraph:=TMinimaleKostengraph
(Graph.InhaltskopiedesGraphen(TMinimaleKostengraph,
TInhaltsknoten,TMinimaleKostenKante,false));
Kostengraph.ZeichneGraph(Paintbox.Canvas);
GraphH:=TInhaltsgraph(Kostengraph);
if Hitch then Kostengraph.SetzeSchrankenMax;
Kostengraph.Ganzzahlig:=Ganzzahlig;
Oberflaeche:=TForm(self);
Ausgabe1.Caption:='Eingabe der Kantenkosten';
Ausgabe1.Refresh;
ShowMessage('Eingabe der Kosten der
Kanten!'+chr(13)+chr(13)+
```

```
    'Kanten mit Mausklick bestimmen!');
    Bildloeschen;
    GraphH.Kantenwertposition:=2;
    GraphH.zeichneGraph(Paintbox.Canvas);
     repeat
      GraphH.Zustand:=false;
      Paintbox.OnMouseDown:=KostenMousedown;
      Paintbox.ONmousemove:=nil;
      Paintbox.OnDblclick:=nil;
       repeat
        Application.Processmessages;
        if GraphH.Abbruch then goto Endproc;
      until GraphH.Zustand;
      Bildloeschen;
     GraphH.zeichneGraph(Paintbox.Canvas);
    until MessageDlg('Keine Kante! Weitere Kosten eingeben?',
    mtConfirmation, [mbYes, mbNo], 0) = mrNo;
    Paintbox.OnMouseDown:=PanelDownMouse;
    Ausgabeloeschen(true);
    ShowMessage('Kosten der Kanten');
    GraphH.Kantenwertposition:=0;
    Kostengraph.MinimaleKosten
    (Paintbox.Canvas,Graph,Oberflaeche,Ausgabe1,
    Maximal,Quelle,Senke,'t',Sliste);
    StringlistnachListbox(Sliste,Listbox);
     Endproc:
    Ausgabeloeschen(true);
    GraphH:=Kostengraph.InhaltskopiedesGraphen
    (TInhaltsgraph,TInhaltsknoten,
    TMinimaleKostenKante,false);;
    GraphH.Kantenwertposition:=1;
    Aktiv:=false;
    Menuenabled(true);
    if GraphH.Abbruch then
     begin
       Aktiv:=true;
      Graph.ZeichneGraph(Paintbox.Canvas);
      ShowMessage('Abbruch!');
     end;
    Sliste.Free;
    SListe:=nil;
    Kostengraph.Freeall;
    Kostengraph:=nil;
  except
    AbbruchClick(Sender);
    Fehler;
  end;
end;
```

```
procedure TKnotenformular.OptimalesMatchingClick(Sender:
TObject);
label Endproc;
var Sliste:TStringList;
    Kostengraph:TMinimaleKostengraph;
    Startknoten,Zielknoten:TMaxflussknoten;
    Gesamtkosten:Real;
    Quelle,Senke:TInhaltsknoten;
    Ganzzahlig,Maximal,optimal:Boolean;
    Fluss:Real;
    Flussok:Boolean;
    Oberflaeche:TForm;
    Graph:TInhaltsgraph;
begin
  try
    Graph:=self.Graph;
    if Graph.Leer then exit;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.AnzahlKomponenten>1 then
     begin
      ShowMessage('Der Graph hat mehrere Komponenten!');
       exit;
     end;
    if Graph.AnzahlKnoten<2 then
     begin
      ShowMessage('Graph hat nur einen Knoten!');
       exit;
     end;
    if Graph.AnzahlungerichteteKanten>0 then
     begin
      ShowMessage('Der Graph hat ungerichtete Kanten');
       exit;
     end;
    if not Graph.Graphistpaar then
     begin
      ShowMessage('Der Graph ist nicht paar!');
       exit;
     end;
    if MessageDlg('Maximale Kosten (statt
    minimale Kosten)?',mtConfirmation,[mbYes,mbNo],0)=mryes
     then
       Maximal:=true
     else
      Maximal:=false;
    Ganzzahlig:=true;
    Sliste:=TStringList.Create;
    Menuenabled(false);
   Kostengraph:=TMinimaleKostengraph(Graph.InhaltskopiedesGraphen
    (TMinimaleKostengraph,
```

```
TInhaltsknoten,TMinimaleKostenKante,false));
Kostengraph.ZeichneGraph(Paintbox.Canvas);
GraphH:=TInhaltsgraph(Kostengraph);
Kostengraph.SetzeSchrankenMax;
Kostengraph.Ganzzahlig:=Ganzzahlig;
Oberflaeche:=TForm(self);
Ausgabe1.Caption:=’Eingabe der Kantenkosten’;
Ausgabe1.Refresh;
ShowMessage(‘Eingabe der Kosten der
Kanten!’+chr(13)+chr(13)+
‘Kanten mit Mausklick bestimmen!’);
Bildloeschen;
GraphH.Kantenwertposition:=2;
GraphH.ZeichneGraph(Paintbox.Canvas);
 repeat
  GraphH.Zustand:=false;
  Paintbox.OnMouseDown:=KostenMousedown;
  Paintbox.ONmousemove:=nil;
  Paintbox.OnDblclick:=nil;
    repeat
    Application.Processmessages;
    if GraphH.Abbruch then goto Endproc;
  until GraphH.Zustand;
  Bildloeschen;
 GraphH.zeichneGraph(Paintbox.Canvas);
until MessageDlg(‘Keine Kante! Weitere Kosten eingeben?’,
mtConfirmation, [mbYes, mbNo], 0) = mrNo;
Paintbox.OnMouseDown:=PanelDownMouse;
Ausgabeloeschen(true);
ShowMessage(‘Kosten der Kanten’);
GraphH.Kantenwertposition:=0;
Kostengraph.MinimaleKosten
(Paintbox.Canvas,Graph,Oberflaeche,Ausgabe1,
Maximal,Quelle,Senke,’o’,Sliste);
StringlistnachListbox(Sliste,Listbox);
Kostengraph.LoescheNichtMatchkanten;
Bildloeschen;
Kostengraph.ZeichneGraph(Paintbox.Canvas);
Endproc:
Ausgabeloeschen(true);
GraphH:=Kostengraph.InhaltskopiedesGraphen
(TInhaltsgraph,TInhaltsknoten,
TMinimaleKostenKante,false);;
GraphH.Kantenwertposition:=1;
Aktiv:=false;
Menuenabled(true);
if GraphH.Abbruch then
 begin
  Aktiv:=true;
  Bildloeschen;
```

```
     Graph.ZeichneGraph(Paintbox.Canvas);
     ShowMessage('Abbruch!');
     end;
    Sliste.Free;
    Sliste:=nil;
    Kostengraph.Freeall;
    Kostengraph:=nil;
  except
    AbbruchClick(Sender);
    Fehler;
  end;
end;


procedure TKnotenformular.ChinesischerBrieftrgerClick(Sender:
TObject);
label Endproc,Euler,Ende;
var Sliste,Esliste:TStringList;
    Kostengraph:TMinimaleKostengraph;
    Startknoten,Zielknoten:TMaxflussknoten;
    Gesamtkosten:Real;
    Quelle,Senke:TInhaltsknoten;
    Ganzzahlig,Maximal:Boolean;
    Fluss:Real;
    Flussok:Boolean;
    M:TEulergraph;
    Index:Integer;
    Oberflaeche:TForm;
    Graph:TInhaltsgraph;

begin
  try
    Graph:=self.Graph;
    if Graph.Leer then exit;
    Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    if Graph.Leer then exit;
    if Graph.AnzahlKomponenten>1 then
     begin
     ShowMessage('Mehrere Komponenten!');
        exit;
      end;
     if Graph.AnzahlKnoten<2 then
      begin
       ShowMessage('Graph hat nur einen Knoten!');
         exit;
      end;
     if Graph.AnzahlungerichteteKanten>0 then
       begin
        ShowMessage('Der Graph hat ungerichtete Kanten');
```

```
    exit;
   end;
  ShowMessage('Minimales Kostenproblem mit Weglängen als Ko
   sten ');
  Sliste:=TStringList.Create;
  Menuenabled(false);
 Kostengraph:=TMinimaleKostengraph(Graph.InhaltskopiedesGraphen
 (TMinimaleKostengraph,TInhaltsknoten,TMinimaleKostenKante,false));
  Kostengraph.ZeichneGraph(Paintbox.Canvas);
  GraphH:=TInhaltsgraph(Kostengraph);
  if Kostengraph.GraphhatgeschlosseneEulerlinie(true) then
   begin
    ShowMessage('Graph hat geschlossene Eulerlinie!');
     goto Euler;
   end;
  Ganzzahlig:=true;
  Maximal:=false;
  Kostengraph.Kantenwertposition:=0;
  Kostengraph.SpeichereSchrankenalsKosten;
  Kostengraph.Kantenwertposition:=2;
  Bildloeschen;
  Kostengraph.ZeichneGraph(Paintbox.Canvas);
  Ausgabe1.Caption:='Kanten-Kosten';
  GraphH.Demopause;
  Kostengraph.Kantenwertposition:=0;
  Kostengraph.SetzeSchrankenMax;
  Kostengraph.Ganzzahlig:=Ganzzahlig;
  Oberflaeche:=TForm(self);
  Kostengraph.MinimaleKosten
  (Paintbox.Canvas,Graph,Oberflaeche,Ausgabe1,
  Maximal,Quelle,Senke,'b',Sliste);
  StringlistnachListbox(Sliste,Listbox);
  if GraphH.Abbruch then goto Ende;
  Kostengraph.Kantenwertposition:=0;
  Kostengraph.ErzeugeunproduktiveKanten;
   Euler:
  Bildloeschen;
  Kostengraph.ZeichneGraph(Paintbox.Canvas);
  M:=TEulergraph(Kostengraph.InhaltskopiedesGraphen
  (TEulergraph,TInhaltsknoten,
  TInhaltskante,false));
  Aktiv:=false;
  GraphH:=M;
  ShowMessage('Eulerlinie suchen');
  Esliste:=TStringList.Create;
  if M.GraphhatgeschlosseneEulerlinie(false) then
   begin
    Bildloeschen;
    Aktiv:=false;
   M.ZeichneGraph(Paintbox.Canvas);
```

```
   M.Eulerlnie(Paintbox.Canvas,Ausgabe1,Esliste,
   GraphH.LetzterMausklickknoten,
   GraphH.LetzterMausklickknoten);
    if SListe.Count>=0 then
    for Index:=0 to SListe.count-1 do
     Esliste.Add(SListe.Strings[Index]);
    StringlistnachListbox(esliste,Listbox);
     Esliste.Free;
     Esliste:=nil;
    Paintbox.OnMouseDown:=PanelDownMouse;
    Paintbox.ONmousemove:=nil;
    Paintbox.OnDblclick:=nil;
   end
   else
    ShowMessage('Es gibt keine geschlossene Eulerlinie');
  Menuenabled(true);
   Endproc:
  Ausgabeloeschen(true);
  Kostengraph.Kantenwertposition:=0;
  if M.GraphhatgeschlosseneEulerlinie(false) then
   begin
    GraphH:=M.InhaltskopiedesGraphen
    (TInhaltsgraph,TInhaltsknoten,TInhaltsKante,false);
      M.Freeall;
      M:=nil;
     Kostengraph.Freeall;
     Kostengraph:=nil;
   end
   else
   begin
    GraphH:=Kostengraph.InhaltskopiedesGraphen
    (TInhaltsgraph,TInhaltsknoten,
    TMinimaleKostenKante,false);;
     Kostengraph.Freeall;
     Kostengraph:=nil;
    end;
   Aktiv:=false;
   Menuenabled(true);
   Ende:
  SListe.Free;
  SListe:=nil;
  if GraphH.Abbruch then
   begin
     Aktiv:=true;
     Bildloeschen;
    Graph.ZeichneGraph(Paintbox.Canvas);
    ShowMessage('Abbruch!');
   end;
except
  AbbruchClick(Sender);
```

```
    Fehler;
  end;
end;

procedure TKnotenformular.AbbruchClick(Sender: TObject);
begin
  try
   inherited;
   if Automatenneugraphaktiv
    then
    begin
      Automatenneugraphaktiv:=false;
      Knotenformular.Graph:=Automatenneugraph.
      InhaltskopiedesGraphen(TInhaltsgraph,
      TInhaltsknoten,TInhaltskante,false);
      Knotenformular.GraphH:=Knotenformular.Graph;
      Automatenneugraph.Freeall;
      Automatenneugraph:=nil;
      Knotenformular.Graph.FaerbeGraph(clblack,pssolid);
      Mainmenu.Items[5].enabled:=true;
      Mainmenu.Items[6].enabled:=true;
      Eingabe.Visible:=false;
      Button.Visible:=false;
      Ausgabeloeschen(false);
    end;
  except
    AbbruchClick(Sender);
    Fehler;
  end;
end;


end.
```

Unit UForm: (für EWK)

```
unit UForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs,
  UKNOTEN, Menus, Buttons, StdCtrls,
  ExtCtrls,UList,UGraph,UInhgrph,UAusgabe,
  UKante;

type
  TKnotenformular = class(TKnotenform)
```

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


var
  Knotenformular: TKnotenformular;

implementation

{$R *.DFM}


end.
```

Projektquelltext für DWK und EWK:

(Bei EWK fehlen die Units UPfad,UMath1 und UMath2 in der uses Anweisung und werden bei Bedarf eingefügt)

```
program Kprojekt;

uses
  Forms,
  UList in 'ULIST.PAS',
  UGraph in 'UGRAPH.PAS',
  UInhGrph in 'UINHGRPH.PAS',
  UKante in 'UKANTE.PAS' {Kantenform},
  UAusgabe in 'UAUSGABE.PAS' {Ausgabeform},
  UPfad in 'UPFAD.PAS',
  UMath1 in 'UMATH1.pas';
  UMath2 in 'UMATH2.PAS';
  UKnoten in 'UKNOTEN.PAS' {Knotenform},
  UForm in 'UForm.pas' {Knotenformular};
  {$R *.RES}

begin
  Application.CreateForm(TKnotenformular, Knotenformular);
  Application.CreateForm(TAusgabeform, Ausgabeform);
  Application.CreateForm(TKantenform, Kantenform);
  Application.CreateForm(TKnotenform, Knotenform);
  Application.Run;
end.
```

8)Beschreibung der Methoden der Programme Knotengraph DWK und der objektorientierten Entwicklungsumgebung EWK

Beschreibung der Unit UList:

Die Unit UList stellt mit TListe den grundlegenden Objekttyp einer erweiterbaren und zur Speicherung beliebiger (Objekt-)Datentypen verwendbaren Liste (ohne Bezugnahme auf spezielle Inhalte) quasi in der Form eines abstrakten Datentyps bereit,die sich vom vordefinierten Delphi-Datentyp TList durch Vererbung ableitet.Aufbauend auf der Datenstruktur TListe werden in der Unit UGraph die Objekttypen TPfadliste, TKantenliste, TKante, TKnotenliste und TKnoten durch Vererbung abgeleitet,aus denen sich dann die grundlegende Datenstruktur eines Graphen TGraph aufbauen läßt.

Methoden von TListe:

Der Datentyp TListe ist eine Liste von Objekten des Datentyps TObject.Die Elemente der Liste sind mittels der Property Element als Element(Index) (mit 0<Index<Anzahl der Elemente der Liste-1) vom Typ TElement anzusprechen.Da TListe keine neuen Felder deklariert, ruft der Constructor die Vorgängermethode TList auf.

Constructor TListe.Create

Diese Methode ist der Constructor von TListe.

Procedure TListe.Free

Diese Methode ist der einfache Destructor von TListe und entfernt Instanzen des Datentyps aus dem Speicher.

Procedure TListe.Freeall

Diese Methode ist der erweiterte Destructor von TListe und entfernt Instanzen des Datentyps einschließlich der einzelnen Listenelemente aus dem Speicher.

Function TListe.Element(Index:Integer):TElement

Die Funktionsmethode gibt das Element mit der Nummer Index (0<=Index<=Anzahl der Elemente der Liste -1) zurück.Wenn der Index außerhalb des zulässigen Bereichs liegt,wird dies durch einen Hinweis angezeigt.

Property Items[Index:Integer]:TElement read Element

Diese Property ermöglicht es,ein Element der Liste mittels

Element(Index) (mit 0<=Index<=Anzahl der Elemente der Liste-1) anzusprechen.

Procedure TListe.AmEndeanfuegen(Ob:TObject)

Diese Methode fügt ein Datenelement Ob vom Typ TObject am Ende an die Liste an.

Procedure TListe.AmAnfanganfuegen(Ob:TObject)

Diese Methode fügt ein Datenelement Ob vom Typ TObject am Anfang an die Liste an.

Procedure TListe.AnPositioneinfuegen(Ob:TObject;Index:Integer)

Die Procedure fügt ein Datenelement Ob vom Typ TObject an der Position Index in die Liste ein.

Procedure TListe.LoescheanderPosition(var Ob:TObject;N:Integer)

Diese Methode löscht ein Element an der Position N aus der Liste,falls es diese Position gibt.
Das Element wird als Referenzparameter Ob zurückgegeben.

Procedure TListe.AmAnfangloeschen(var Ob:TObject);

Diese Methode löscht das Anfangselement der Liste,und gibt es als Referenzparameter Ob zurück.

Procedure TListe.AmEndeloeschen(var Ob:TObject)

Diese Methode löscht das Endelement der Liste, und gibt es als Referenzparameter Ob zurück.

Procedure TListe.LoescheElement(Ob:TObject)

Diese Methode sucht nach einem Element Ob in der Liste und löscht es,falls es existiert.Das Element wird als Referenz-parameter zurückgegeben.

Procedure TListe.VertauscheElemente(Index1,Index2:Integer)

Diese Methode vertauscht die Elemente an den Positionen Index1 und Index2 in der Liste miteinander.

Procedure TListe.VerschiebeElement(Index1,Index2:Integer)

Diese Methode verschiebt ein Element von der Position Index1 an die Position Index2 in der Liste.

Procedure TListe.FuerjedesElement(Vorgang:TVorgang)

Für jedes Element der Liste vom ersten bis zum letzten Element wird die Procedure Vorgang vom Typ TVorgang ausgeführt. (TVorgang= procedure(Ob:TObject))

Procedure TListe.FuerjedesElementzurueck(Vorgang:TVorgang)

Für jedes Element der Liste vom letzten bis zum ersten Element (d.h. rückwärts) wird die Procedure Vorgang vom Typ TVorgang ausgeführt.(TVorgang=procedure(Ob:TObject))

Procedure TListe.FueralleElemente(Ob:TObject;Handlung:THandlung)

Für alle Elemente der Liste vom ersten bis zum letzten Element (d.h. vorwärts) wird die Procedure Handlung vom Typ THandlung ausgeführt.Die Procedure Handlung benötigt zur Übergabe den Werte-Parameter Ob vom Typ TObject.(THandlung=procedure(Ob1,Ob2:TObject))

Procedure TListe.FueralleElementezurueck (Ob:TObject;Handlung:THandlung)

Für alle Elemente der Liste vom letzten bis zum ersten Element (d.h. rückwärts) wird die Procedure Handlung vom Typ THandlung ausgeführt.Die Procedure Handlung benötigt zur Übergabe den Werteparameter Ob vom Typ TObject.(THandlung=procedure(Ob1,Ob2:TObject))

Procedure TListe.Loeschen

Diese Methode löscht alle Elemente aus der Liste.

Procedure TListe.Sortieren(Vergleich:TVergleich;Wert:TWert)

Die Elemente der Liste werden sortiert.Das Vergleichkriterium wird vorgegeben durch die Funktion Vergleich vom Typ TVergleich.Diese Function benötigt eine Function Wert vom Typ TWert,die jedem Object der Liste vom Typ TObject einen Wert zuweist.(TVergleich=function(Ob1,Ob2:TObject;Wert:TWert):Boolean /TWert= function(Ob:TObject):Extended)

Function TListe.Anzahl:Integer

Diese Funktionsmethode gibt die Anzahl der Elemente der Liste zurück.

Function TListe.WertsummederElemente(Wert:TWert):Extended

Diese Funktionsmethode bestimmt die Summe der Elemente der Liste,wobei der Wert jedes Elements durch die Funktion Wert vom Typ TWert vorgegeben wird.Falls die Wertsumme absolut größer als 1.0 E 40 ist,wird ein Fehler (Division durch Null) ausgelöst.(TWert=function(Ob:TObject):Extended)

Function TListe.WertproduktderElemente(Wert:TWert):Extended

Diese Funktionsmethode bestimmt das Produkt der Elemente der Liste,wobei der Wert jedes Elements durch die Funktion Wert vom Typ TWert vorgegeben wird.Falls das Wertprodukt absolut größer als 1.0 E 40 ist, wird ein Fehler (Division durch Null) ausgelöst. (TWert=function(Ob:TObject):Extended)

Function TListe.Leer:Boolean

Diese Funktionsmethode bestimmt,ob die Liste leer ist.

Function TListe.Erstes:Integer

Diese Funktionsmethode gibt 0,d.h.die Position des ersten Elements der Liste zurück.

Function TListe.Letztes:Integer

Diese Funktionsmethode gibt die Position des letzten Elements der Liste (Anzahl der Elemente -1) zurück.

Function TListe.Position(Ob:TObject):Integer

Diese Funktionsmethode gibt die Position des Objectes Ob vom Typ TObject in der Liste zurück,wenn Ob in der Liste enthalten ist.Ansonsten ist der Rückgabewert -1.

Function TListe.ElementistinListe(Ob:TObject):Boolean

Diese Funktionsmethode gibt an,ob das Element Ob vom Typ TObject in der Liste enthalten ist.

FunctionTListe.ErsterichtigePosition(Bedingung:TBedingung):Integer

Diese Funktionsmethode bestimmt die erste Positionszahl (Index) eines Elements in der Liste,das die Bedingung Bedingung vom Typ TBedingung erfüllt,wobei die Liste vom ersten bis zum letzten Element d.h. vorwärts durchsucht wird.Falls kein Element die Bedingung erfüllt,wird -1 zurückgegeben (TBedingung=function(Ob:TObject):Boolean).

Function TListe.ErstefalschePosition
(Bedingung:TBedingung):Integer

Diese Funktionsmethode bestimmt die erste Positionszahl (Index) eines Elements in der Liste,das die Bedingung Bedingung vom Typ TBedingung nicht erfüllt,wobei die Liste vom ersten bis zum letzten Element,d.h. vorwärts durchsucht wird.Falls kein Element nicht die Bedingung erfüllt,wird -1 zurückgegeben.(TBedingung=function(Ob:TObject):Boolean)

Function TListe.LetzterichtigePosition
(Bedingung:TBedingung):Integer

Diese Funktionsmetode bestimmt die letzte Positionszahl (Index) eines Elements in der Liste,das (noch) die Bedingung Bedingung vom Typ TBedingung erfüllt,wobei die Liste vom ersten bis zum letzten Element,d.h. vorwärts durchsucht wird.Falls kein Element die Bedingung erfüllt wird -1 zurückgegeben. (TBedingung=function(Ob:TObject):Boolean)

Function TListe.ErstepassendePosition(Vergleich:TVergleich;
Ob:TObject;Wert:TWert):Integer

Diese Funktionsmethode bestimmt die erste Positionszahl (Index) eines Elements in der Liste,bei dem die Function Vergleich vom Typ TVergleich erfüllt ist,wobei die Liste vom ersten bis zum letzten Element, d.h. vorwärts durchsucht wird.Falls kein Element die Bedingung erfüllt wird -1 zurückgegeben.Die Function Vergleich benötigt die Function Wert vom Typ Wert,die den Wert eines Listenelements Ob vom Typ TObject bestimmt. (TVergleich = function(Ob1,Ob2:TObject;Wert:TWert):Boolean / TWert=function(Ob:TObject):Extended)

Function TListe.ErsteunpassendePosition(Vergleich:TVergleich;
Ob:TObject;Wert:TWert):Integer

Diese Funktionsmethode bestimmt die erste Positionszahl (Index) eines Elements in der Liste,bei dem die Function Vergleich vom Typ TVergleich nicht erfüllt ist,wobei die Liste vom ersten bis zum letzten Element d.h. vorwärts durchsucht wird.Falls kein Element nicht die Bedingung erfüllt wird -1 zurückgegeben.Die Function Vergleich benötigt die Function Wert vom Typ Wert,die den Wert eines Listenelements Ob vom Typ TObject bestimmt.(TVergleich=function(Ob1,Ob2:TObject;Wert:TWert):Boolean / TWert=function(Ob:TObject):Extended)

Function TListe.ErstebestePosition
(Vergleich:TVergleich;Wert:TWert):Integer

Diese Funktionsmethode bestimmt die erste Positionszahl (In-

dex) eines Elements in der Liste,bei dem die Function Vergleich vom Typ TVergleich am besten (maximal) erfüllt ist,wobei die Liste vom letzten bis zum ersten Element d.h.rückwärts durchsucht wird.Falls kein Element die Bedingung erfüllt wird -1 zurückgegeben.Die Function Vergleich benötigt die Function Wert vom Typ Wert,die den Wert eines Listenelements vom Typ TObject bestimmt.(
TVergleich=function(Ob1,Ob2:TObject;Wert:TWert):Boolean /
TWert=function(Ob:TObject):Extended)

Function TListe.LetztebestePosition(Vergleich:TVergleich; Wert:TWert):Integer

Diese Funktionsmethode bestimmt die letzte größte Positionszahl (Index) eines Elements in der Liste,bei dem die Function Vergleich vom Typ TVergleich am besten (maximal) erfüllt ist,wobei die Liste vom ersten bis zum letzten Element d.h.vorwärts durchsucht wird.Falls kein Element die Bedingung erfüllt wird -1 zurückgegeben.Die Function Vergleich benötigt die Function Wert vom Typ Wert,die den Wert eines Listenelements vom Typ TObject bestimmt.(
TVergleich=function(Ob1,Ob2:TObject;Wert:TWert):Boolean/
TWert=function(Ob:TObject):Extended)

Methoden von TElement:

Der Datentyp TElement leitet sich durch Vererbung von TListe ab. Andererseits enthält TListe eine Property Element vom Typ TElement.So definiert TElement den Datentyp für ein Element der Liste.TElement enthält zwei Felder:
Wertposition_: Speichert die Position des Datenfeldes der Wertliste,die als Rückgabewert der Funktion Wert dient.
Wertliste_:Eine Liste vom Typ TStringlist,die dazu dient,die Werte von Datenfeldern von TElement als Datentyp String zu speichern.
Beide Felder und ihre Methoden bzw. Propertys dienen der Vererbung an von TElement abgeleitete Klassen.

Constructor TElement.Create

Diese Methode ist der Constructor für TElement.

Procedure TElement.Free

Diese Methode ist der Destructor von TElement und entfernt Instanzen des Datentyps aus dem Speicher.

Procedure TElement.SetzeWertposition(P:Integer)

Setzt die Wertposition für die Wertliste auf den Wert von

P.Diese Methode dient der Definition der Property Position.

Function TElement.WelcheWertposition:Integer

Diese Funktionsmethode gibt die Wertposition zurück.Sie dient zur Definition der Property Position.

Property Position:Integer read WelcheWertposition write SetzeWertposition

Das ist die Property für das Datenfeld Wertposition.

Function TElement.Wertlisteschreiben:TStringlist

Diese Funktionsmethode gibt die Wertliste zurück.Sie ist geeignet zum Überschreibung durch eine entsprechende Methode der von TElement durch Vererbung abgeleiteten Klassen.Diese Methoden sollten die Elemente der Datenfelder des Datentyps in den Felder der Wertliste speichern.

Procedure TElement.Wertlistelesen;virtual;abstract

Diese Methode dient zur Überschreibung durch eine entsprechende Methode der von TElement durch Vererbung abgeleiteten Klassen.Diese Methoden sollten die Elemente der Wertliste in den Datenfelder des Datentyps speichern.

Function TElement.Wert:string

Diese Funktionsmethode gibt den Inhalt des durch Position in der Wertliste beschriebenen Datenfeldes als Rückgabewert vom Datentyp String zurück.

Proceduren und Funktionen,die von der Unit UListe exportiert werden:

Function GGT(A,B:Longint):Longint

Diese Funktion gibt den größten gemeinsamen Teiler der Zahlen A und B zurück.

Function Tan(X:Extended):Extended

Diese Funktion gibt den Tangens von X zurück.

Function StringtoReal(S:string):Extended

Diese Funktion wandelt einen String S,falls möglich und zulässig,in eine Zahl von Datentyp Extended (Real) um.Falls der String nicht als Zahl zu interpretieren ist,wird 0

zurückgegeben.Bei Bereichsüberschreitung wird eine Exception ausgelöst und eine Fehlermeldung ausgegeben.

Function RealtoString(R:Extended):string

Diese Funktion wandelt eine Zahl R vom Datentyp Extended (Real) in einen String um.Bei Bereichsüberschreitung wird eine Exception ausgelöst und eine Fehlermeldung ausgegeben.

Function Integertostring(I:Integer):string

Diese Funktion wandelt eine Zahl I vom Datentyp Integer,falls möglich, in einen String um.Falls der String nicht als Zahl zu interpretieren ist,wird 0 zurückgegeben.Bei Bereichsüberschreitung wird eine Exception ausgelöst und eine Fehlermeldung ausgegeben.

Function StringtoInteger(S:string):Integer

Diese Funktion wandelt eine Zahl vom Datentyp Integer in einen String um.Bei Bereichsüberschreitung wird eine Exception ausgelöst und eine Fehlermeldung ausgegeben.

Function StringistRealZahl(S:string):Boolean

Diese Funktion testet,ob ein String als Gleitkommazahl,Festkommazahl bzw. Integerzahl aufzufassen ist.

Function RundeStringtoString(S:string;Stelle:Integer):string

Falls Stelle größer gleich Null ist, faßt diese Funktion den String S als Gleitkomazahl,als Festkommazahl oder Intergerzahl auf,schneidet die Zahl auf die durch Stelle vorgegebene Stellenzahl ab,wandelt die Zahl wieder in einen String um und gibt den String als Ergebnis zurück.Wenn Stelle nicht negativ ist,wird die Zahl nach der entsprechenden Stellenzahl nach dem Komma abgeschnitten.Wenn Stelle negativ ist,wird der String s mit der Zeichenanzahl der absoluten Zahl Stelle ausgegeben.Stelle darf nicht größer als 7 sein.Bei größeren Zahlen als 7 wird Stelle auf 7 gesetzt.

Function RundeZahltoString(R:Real;Stelle:Integer):string

Diese Funktion wandelt die Zahl R gemäß der Function RealtoString in einen String um und ruft mit diesem String und Stelle als Parameter die Function RundeStringtoString auf.Der dabei erhaltene Rückgabewert wird zurückgegeben.

Function Minimum(R1,R2:Extended):Extended

Die Funktion bestimmt das Minimum der Zahlen R1 und R2.
Function Maximum(R1,R2:Extended):Extended

Die Funktion bestimmt das Maximum der Zahlen R1 und R2.

Procedure Pause(N:Longint)

Diese Procedure erzeugt eine Pause von N Millisekunden.

Beschreibung der Unit UGraph:

Die Unit Graph definiert die grundlegenden Objekttypen für die Verwaltung eines Graphen.Aufbauend auf der Datenstruktur TListe aus der Unit UList werden von dieser die Objekttypen TPfadliste, TKantenliste, TKante, TKnotenliste und TKnoten durch Vererbung abgeleitet.Der Objekttyp TGraph hat TObject als Superklasse und besteht im wesentlichen aus einer Knotenliste vom Typ TKnotenliste und aus einer Kantenliste vom Typ TKantenliste als Datenfelder.Die Unit UGraph enthält die Methoden eines Graphen,die sich ohne Bezug auf Knoteninhalte bzw. Kanteninhalte (dazu zählen auch Koordinaten und Zeichenfarben) realisieren lassen.(Abstrakte Datenstruktur ADT)

Methoden von TKantenliste

Der Datentyp TKantenliste leitet sich durch Vererbung von TListe ab.Der Datentyp TGraph enthält eine Kantenliste vom Typ TKantenliste als Datenfeld.
Die Elemente der TKantenliste sind mittels der Property Kante als Kante(Index) (mit 0<=Index<=Anzahl der Elemente der Kantenliste-1) vom Typ TKante anzusprechen.

Constructor TKantenliste.Create

Das ist der Constructor von TKantenliste.

Procedure TKantenliste.Free

Dieser einfache Destructor entfernt Instanzen des Datentyps TKantenliste aus dem Speicher.

Procedure TKantenliste.Freeall

Dieser erweiterte Destructor entfernt Instanzen des Datentyps TKantenliste samt den Elementen der Liste aus dem Speicher.

Function Kante(Index:Integer):TKante

Diese Funktionsmethode gibt das Element (die Kante) mit der Nummer Index (0<=Index<=Anzahl der Elemente der KantenListe -1) zurück.Sie definiert die Property Items.Wenn der Index außerhalb des zulässigen Bereichs liegt,wird dies durch einen Hinweis angezeigt.

Property Items[Index:Integer]:TKante read Kante

Diese Property ermöglicht es ein Element der Kantenliste mittels Kante(Index) (mit 0<=Index<=Anzahl der Elemente der Kantenliste-1) anzusprechen.

Function TKantenliste.Kopie:TKantenliste

Diese Funktionsmethode erzeugt eine Kopie der Kantenliste und gibt sie zurück.Dabei wird lediglich eine Kopie der Kantenliste vom Typ TKantenliste erzeugt,nicht jedoch werden die in der Kantenliste enthaltenden Elemente (Kanten) kopiert.Kopie und Orginal enhalten also Zeiger auf dieselben Kanten-Objekte.

Function TKantenliste.Graph:TGraph

Diese Funktionsmethode erzeugt einen Graphen,der die Kantenliste vom Typ TKantenliste als Datenfeld Kantenliste enthält.Dabei enthält die Kantenliste des neuen Graphen dieselben Kantenzeiger wie die Orginal-Kantenliste.Gleichzeitig wird eine neue Knotenliste mit den Anfangs-und Endknoten der Kanten der Kantenliste erzeugt und ist als Datenfeld Knotenliste_ im Graphen enthalten.Die Knotenliste enthält die Knoten in der Reihenfolge,in der die Pfadrichtung in der Kantenliste definiert ist.

Function TKantenliste.UGraph:TGraph

Diese Funktionsmethode erzeugt einen Graphen,der die Kantenliste vom Typ TKantenliste als Datenfeld Kantenliste_ enthält.Dabei enthält die Kantenliste des neuen Graphen dieselben Kantenzeiger wie die der Orginal-Kantenliste.Gleichzeitig wird eine neue Knotenliste mit den Anfangs-und Endknoten der Kanten der Kantenliste erzeugt und ist als Datenfeld Knotenliste_ im Graphen enthalten.Die Knotenliste ist nicht unbedingt gemäß der Pfadrichtung der Kantenliste geordnet,sondern enthält die beim Erzeugen gegebene Knotenfolge.

Function TKantenliste.Kantenlistealsstring:string

Die Werte der Kanten der Kantenliste werden mit jeweils einem Leerzeichen getrennt,gemäß der Reihenfolge,die von der Kanten-

liste vorgegeben wird, aneinandergereiht und in einen Gesamtstring umgewandelt,der von der Function zurückgegeben wird,sofern er eine Länge von 254 Zeichen nicht übersteigt.Ansonsten wird nur ein Teilstring von maximal 254 Zeichen zurückgegeben.

Methoden von TKante:

TKante stammt durch Vererbung von TKantenliste ab und hat folgende zusätzliche Datenfelder:

```
Anfangsknoten_:TKnoten;
EndKnoten_:TKnoten;
Pfadrichtung_:TKnoten;
Gerichtet_:Boolean;
Wertposition_:Integer;
Wertliste_:TStringlist;
Besucht_:Boolean;
Erreicht_:Boolean;
```

Anfangsknoten_ und Endknoten_ vom Typ TKnoten sind Zeiger auf die beiden Knoten einer Kante.Pfadrichtung_ vom Typ TKnoten beschreibt bei gerichteten Kanten die Pfeilrichtung,bei ungerichteten Kanten die Durchlaufrichtung und ist ein Zeiger auf den entsprechenden Knoten.Gerichtet_ vom Typ Boolean gibt an,ob es sich um eine gerichtete Kante oder nicht handelt.Wertposition_ vom Typ Integer beinhaltet die Position des Datenfeldes,das als Ausgabe für die Property Wert zur Verfügung steht.Wertliste_ vom Typ TStringlist enthält die Daten der Datenfelder (evtl. umgewandelt) als Strings. Besucht_ und Erreicht_ vom Typ Boolean sind Felder,die für Suchalgorithmen verwendet werden können,um die Kante zu markieren.

Darüberhinaus enthält der Datentyp eine Reihe von Propertys,um über sie auf die oben genannten Datenfelder zuzugreifen.

Propertys:

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

Durch die folgenden Methoden von TKante (Prozeduren und Functionen) wird auf die Propertys lesend und schreibend zugegriffen:

Function TKante.WelcherAnfangsknoten:TKnoten
Procedure TKante.SetzeAnfangsknoten(Kno:TKnoten)
Function TKante.WelcherEndknoten:TKnoten
Procedure TKante.SetzeEndknoten(Kno:TKnoten)
Function TKante.WelchePfadrichtung:TKnoten
Procedure TKante.SetzePfadrichtung(Kno:TKnoten)
Function TKante.Istgerichtet: Boolean
Procedure TKante.Setzegerichtet(Gerichtet:Boolean)
Function TKante.WelcheWertposition:Integer
Procedure TKante.SetzeWertposition(P:Integer)
Function TKante.WelcheWertliste:TStringlist
Procedure TKante.SetzeWertliste(W:TStringlist)
Function TKante.Istbesucht:Boolean
Procedure TKante.Setzebesucht(B:Boolean)
Function TKante.Isterreicht:Boolean
Procedure TKante.Setzeerreicht(E:Boolean)
Function TKante.Wertlesen:string
Procedure TKante.Wertschreiben(S:string)

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Function TKante.Wertlisteschreiben:TStringlist;virtual;abstract;

Diese Funktionsmethode wird als virtual und abstract definiert,damit durch Vererbung erzeugte Nachfolgerobjekte mittels ihrer Wertlisteschreiben-Methode durch die Property Wert ihre Datenfelder als Wert zurückgeben können.Durch die Property Position wird jeweils das entsprechende Datenfeld ausgewählt.Die Daten sind als strings in dem Feld (Liste) Wertliste_ gespeichert.

Procedure TKante.Wertlistelesen;virtual;abstract;

Diese Methode wird als virtual und abstract definiert,damit den Datenfelder der duch Vererbung erzeugten Nachfolgerobjekte mittels ihrer Werlistelesen-Methode durch die Property Wert Werte zugewiesen werden können.Durch die Property Position wird jeweils das entsprechende Datenfeld ausgewählt. Die Daten werden als strings in dem Feld (Liste) Wertliste_ gespeichert

Constructor TKante.Create

Das ist der Constructor von TKante.

Procedure TKante.Free

Dieser einfache Destructor löscht eine Instanz vom Datentyp TKante aus dem Speicher.Die Felder Anfangsknoten_ und Endknoten_ werden nicht gelöscht.

Procedure TKante.Freeall

Dieser erweiterte Destructor entfernt Instanzen des Datentyps TKante einschließlich den Feldern Anfangsknoten_ und Endknoten_ aus dem Speicher.

Function TKante.Zielknoten(Kno:TKnoten):TKnoten

Diese Funktionsmethode gibt den Zielknoten einer Kante zurück.Der Zielknoten ist bei gerichteten Kanten der Knoten,auf den die Richtung der Kante zeigt,bei ungerichteten Kanten ist es der zweite Knoten der Kante,der ungleich Kno ist.
.
Function TKante.Quellknoten(Kno:TKnoten):TKnoten

Diese Funktionsmethode gibt den Quellknoten einer Kante zurück.Der Quellknoten ist bei gerichteten Kanten der Knoten,auf den nicht die Richtung der Kante zeigt,bei ungerichteten Kanten ist es der Knoten,der ungleich Kno ist.

Function TKante.KanteistSchlinge:Boolean

Diese Funktionsmethode gibt zurück,ob eine Kante den gleichen Knoten als Anfangs- und Endknoten hat.

Function TKante.KanteistKreiskante:Boolean

Diese Funktionsmethode gibt zurück,ob die Kante Teil eines Kreises ist.

Methoden von TPfadliste:

TPfadliste ist Nachkomme von TListe.Die Datenstruktur dient dazu, um eine Anzahl von Pfaden als Liste zu speichern.Die einzelnen Pfade sind am günstigsten vom Datentyp TGraph (sonstige Möglichkeit: Objekte vom Typ TKantenliste) zu wählen.Auf diese Weise können die Methoden von TGraph und TInhaltsgraph wie z.B: ZeichneGraph auf den Datentyp Pfad angewendet werden.Auf die einzelnen Elemente der Pfadliste kann mittels der Functionsmethode Pfad und der Property Items mittels Pfad(Index)

zugegriffen werden.Der Datentyp TKnoten enthält ein Datenfeld Pfadliste_ vom Typ TPfadliste,in dem von diesem Knoten ausgehende Pfade als Liste gespeichert werden können.

Constructor TPfadliste.Create

Diese Methode ist der Constructor von TPfadliste.

Procedure TPfadliste.Free

Diese Methode ist der einfache Destructor und entfernt Instanzen von TPfadliste (ohne die in ihnen enthaltenden Pfade) aus dem Speicher.

Procedure TPfadliste.Freeall

Diese Methode ist der erweiterte Destructor und entfernt Instanzen von TPfadliste einschließlich der in ihnen gespeicherten Pfade aus dem Speicher.

Function TPfadliste.Pfad(Index:Integer):TPfad

Diese Funktionsmethode gibt das Element der Pfadliste zurück,das durch Index bestimmt ist.(0<=Index<=Zahl der Elemente der Liste -1)

Property Items[Index:Integer]:TPfad read Pfad

Diese Property ermöglicht es,ein Element der Pfadliste mittels Pfad(Index) (mit 0<=Index<=Anzahl der Elemente der Pfadliste-1) anzusprechen.

Function TPfadliste.Kopie:TPfadliste

Diese Funktionsmethode gibt eine Kopie der Pfadliste zurück,deren Elemente gleich der der ursprünglichen Pfadliste sind.(Nur die Liste liegt dabei als Kopie vor,nicht die Elemente der Liste selber.)

Methoden von TPfad:

TPfad ist Nachkomme von TPfadliste und beschreibt ein Element der Pfadliste.TPfad sollte zur Ausnutzung von Methoden wie Pfadlaenge vom Datentyp TGraph (sonstige Möglchkeit:Objekte vom Typ TKantenliste) gewählt werden.

Constructor TPfad.Create

Diese Methode ist der Constructor von TPfad.

Procedure TPfad.Free

Diese Methode ist der Destructor von TPfad und gibt den Speicher von Instanzen des Datentyps frei.

Function TPfad.Knotenliste:TKnotenliste

Diese Funktionsmethode gibt die Knoten des Pfades,der als Graph vom Typ TGraph aufgefaßt wird, als Knotenliste des Graphen vom Typ TKnotenliste zurück.Nur sinnvoll,wenn TPfad vom Typ TGraph ist.

Function TPfad.Kantenliste:TKantenliste

Diese Funktionsmethode gibt die Kanten des Pfades,der als Graph vom Typ TGraph aufgefaßt wird,als Kantenliste vom Typ TKantenliste zurück.Nur sinnvoll,wenn TPfad vom Typ TGraph ist

Function TPfad.Pfadlaenge:Extended

Diese Funktionsmethode gibt die Pfadlaenge,d.h. die Anzahl der Kanten eines als TGraph aufzufassenden Pfades zurück.Nur sinnvoll,wenn der TPfad vom Typ TGaph ist.Ansonsten ist der Rückgabewert 0.

Function TPfad.PfadSumme(Wert:TWert):Extended

Diese Funktionsmethode gibt die Pfadsumme,d.h. die Kantensumme eines als TGraph aufzufassenden Pfades zurück.Die Kantenbewertung Wert vom Typ TWert muß zum Datentyp Extended kompatibel sein.Nur sinnvoll,wenn der TPfad vom Typ TGaph ist.Ansonsten Rückgabewert 0. (TWert= function(Ob:TObject): Extended)

Function TPfad.Pfadprodukt (Wert:TWert): Extended

Diese Funktionsmethode gibt das Pfadprodukt,d.h. die Produkte der Kantenwerte eines als TGraph aufzufassenden Pfades zurück.Die Kantenbewertung Wert vom Typ TWert muß zum Datentyp Extended kompatibel sein.Nur sinnvoll,wenn der TPfad vom Typ TGraph ist.Ansonsten ist der Rückgabewert 0. (TWert=function(Ob:TObject):Extended)

Function TPfad.Pfadstring(Sk:TString):string

Diese Funktionsmethode erzeugt aus einem Pfad vom Typ TPfad einen String,wobei Sk vom Typ TString vorgibt,welchen Teilstring jede Kante des Pfades zum Gesamtstring beiträgt.Die Functionsmethode kann z.B. bei der Ausgabe eines Pfades als Folge von Kanten- bzw. Knotenwerten nützlich sein. (TString=function(Ob:TObject):string)

Function TPfad.Pfadstringliste(Sk:TString):TStringlist

Diese Funktionsmethode erzeugt aus einem Pfad vom Typ TPfad eine Stringliste vom Typ TStringlist (Delphi-Datentyp), wobei Sk vom Typ TString vorgibt,welcher Teilstring von jeder Kante des Pfades als Element in der Stringliste gespeichert wird.Die Function kann z.B. bei der Ausgabe eines Pfades als Folge von Kanten- bzw. Knotenwerten nützlich sein.(TString= function(Ob:TObject):string)

Methoden von TKnotenliste:

TKnotenliste ist Nachkomme von TListe.Die Elemente der Liste sind die Knoten vom Datentyp TKnoten,auf die mittels der Property Items und der Funktion Knoten vom Typ TKnoten mittels Knoten(Index) (mit 0<=Index<=Zahl der Elemente der Knotenliste - 1) zugegriffen werden kann.

Constructor TKnoten.Create

Diese Methode ist der Constructor von TKnoten.

Procedure TKnotenliste.Free

Diese Methode ist der einfache Destructor und entfernt Instanzen von TKnotenliste (ohne die in ihnen gespeicherten Knoten) aus dem Speicher.

Procedure TKnotenliste.Freeall

Diese Methode ist der erweiterte Destructor entfernt Instanzen von TKnotenliste einschließlich der in ihnen gespeicherten Knoten aus dem Speicher.

Function TKnotenliste.Knoten(Index:Integer):TKnoten

Diese Funktionsmethode gibt den Knoten der Knotenliste zurück,der durch Index bestimmt ist.(0<=Index<=Zahl der Elemente der Knotenliste -1)

Property Items[Index:Integer]:TKnoten read Knoten

Diese Property ermöglicht es ein Element der Knotenliste mittels Knoten(Index) (mit 0<=Index<=Anzahl der Elemente der Knotenliste-1) anzusprechen.

Function TKnotenliste.Kopie:TKnotenliste

Diese Funktionsmethode erzeugt eine Kopie der Knotenliste mit denselben Knoten als Elementen wie die ursprüngliche Liste.(Nur

die Liste wird kopiert,nicht die Knoten als Elemente.)

Function TKnotenliste.Knotenlistealsstring:string

Diese Funktionsmethode erzeugt aus den Knoten-Werten einer Knotenliste einen Gesamtstring durch Aneinanderfügen (mit Leerzeichen zwischen den Werten).

Methoden von TKnoten:

Der Datentyp TKnoten ist Nachkomme von TKnotenliste.Er dient zur Beschreibung und Speicherung der Knoten eines Graphen.Er enthält folgende zusätzliche Datenfelder:

```
Graph_: TGraph;
EingehendeKantenliste_:TKantenliste;
AusgehendeKantenliste_:TKantenliste;
Pfadliste_:TPfadliste;
Besucht_:Boolean;
Erreicht_Boolean;
Wertposition_:Integer;
Wertliste_:Tstringlist;
```

Graph_ ist ein Zeiger auf den Graphen vom Typ TGraph,zu dem der Knoten gehört.EingehendeKantenliste_ und AusgehendeKantenliste_ vom Typ TKantenliste sind Zeiger auf Kantenlisten,die die eingehenden und ausgehenden Kanten des Knoten speichern.Eine gerichtete Kante wird dabei nur in einer dieser Listen gespeichert.Eine ungerichtete Kante wird in beiden Kantenlisten gespeichert.Pfadliste_ vom Typ TPfadliste kann eine Liste von Pfaden,die zu diesem Knoten gehören,aufnehmen.
Besucht_ und Erreicht_ vom Typ Boolean sind Markierungen,die angeben,ob der Knoten (z.B: bei einem Suchlauf) besucht wurde.Wertposition_ vom Typ Integer speichert die Position des Datenfeldes,das bei der Property Wert für die Ausgabe benutzt wird.Schließlich enthält Wertliste_ vom Typ TStringlist den Inhalt aller Datenfelder (evtl. umgewandelt) als Strings.

Darüberhinaus enthält der Datentyp eine Reihe von Propertys,um über sie auf die oben genannten Datenfelder zuzugreifen.

Propertys:

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;

Durch die folgenden Methoden von TKnoten (Prozeduren und Functionen) wird auf die Propertys lesend und schreibend zugegriffen:

Function TKnoten.WelcherGraph:TGraph;
Procedure TKnoten.SetzeGraph(G:TGraph);
Function TKnoten.WelcheEingehendeKantenliste:TKantenliste;
Procedure TKnoten.SetzeEingehendeKantenliste(L:TKantenliste);
Function TKnoten.WelcheAusgehendeKantenliste:TKantenliste;
Procedure TKnoten.SetzeAusgehendeKantenliste(L:TKantenliste);
Function TKnoten.WelchePfadliste:TPfadliste;
Procedure TKnoten.SetzePfadliste(P:TPfadliste);
Procedure TKnoten.SetzeWertposition(P:Integer);
Function TKnoten.WelcheWertposition:Integer;
Function TKnoten.WelcheWertliste:TStringlist;
Procedure TKnoten.SetzeWertliste(W:TStringlist);
Function TKnoten.Istbesucht:Boolean;
Procedure TKnoten.Setzebesucht(B:Boolean);
Function TKnoten.Isterreicht:Boolean;
Procedure TKnoten.Setzeerreicht(E:Boolean);
Procedure TKnoten.Wertschreiben(S:string);
Function TKnoten.Wertlesen:string;

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Function TKnoten.Wertlisteschreiben:
TStringlist;virtual;abstract;

Diese Funktionsmethode wird als virtual und abstract definiert,damit durch Vererbung erzeugte Nachfolgerobjekte mittels ihrer Wertlisteschreiben-Methode durch die Property Wert ihre Datenfelder als Wert zurückgeben können.Durch die Property Position wird jeweils das entsprechende Datenfeld ausgewählt.Die Daten sind als Strings in dem Feld Wertliste_ gespeichert.

Procedure TKnoten.Wertlistelesen;virtual;abstract;

Diese Methode wird als virtual und abstract definiert,damit den

Datenfelder der duch Vererbung erzeugten Nachfolgerobjekte mittels ihrer Werlistelesen-Methode durch die Property Wert Werte zugewiesen werden können.Durch die Property Position wird jeweils das entsprechende Datenfeld ausgewählt. Die Daten werden als Strings in dem Feld Wertliste_ gespeichert

Constructor TKnoten.Create

Diese Methode ist der Constructor von TKnoten.

Procedure TKnoten.Free

Diese Methode ist der einfache Destructor von TKnoten und entfernt Instanzen des Datentyps (ohne die in der ausgehenden und eingehenden Kantenliste gespeicherten Kanten) aus dem Speicher.

Procedure TKnoten.Freeall

Diese Methode ist der erweiterte Destructor entfernt Instanzen von TKnoten einschließlich der in der ausgehenden und eingehenden Kantenliste gespeicherten Kanten aus dem Speicher.

Procedure TKnoten.FueralleausgehendenKanten(Handlung:THandlung)

Diese Methode führt die Anweisungen der Procedure Handlung vom Typ THandlung für alle Elemente der ausgehenden Kantenliste (für alle ausgehenden Kanten)
aus.(THandlung=procedure(Ob1,Ob2:TObject))

Procedure TKnoten.FueralleeingehendenKanten(Handlung:THandlung)

Diese Methode führt die Anweisungen der Procedure Handlung vom Typ THandlung für alle Elemente der eingehenden Kantenliste (für alle eingehenden Kanten)
aus.(THandlung=procedure(Ob1,Ob2:TObject))

Procedure TKnoten.FueralleKanten(Handlung:THandlung)

Diese Methode führt die Anweisungen der Procedure Handlung vom Typ THandlung für alle Elemente der ausgehenden und eingehenden Kantenliste (für alle Kanten) aus.
(THandlung=procedure(Ob1,Ob2:TObject))

Procedure TKnoten.FuerjedeausgehendeKante(Vorgang:TVorgang)

Diese Methode führt die Anweisungen der Procedure Vorgang vom Typ TVorgang für alle Elemente der ausgehenden Kantenliste (für alle ausgehenden Kanten) aus.(TVorgang=procedure(Ob:TObject))

Procedure TKnoten.FuerjedeeingehendeKante(Vorgang:TVorgang)

Diese Methode führt die Anweisungen der Procedure Vorgang vom Typ TVorgang für alle Elemente der eingehenden Kantenliste (für alle eingehenden Kanten) aus.(TVorgang=procedure(Ob:TObject))

Procedure TKnoten.FuerjedeKante(Vorgang:TVorgang)

Diese Methode führt die Anweisungen der Procedure Vorgang vom Typ TVorgang für alle Elemente der eingehenden und ausgehenden Kantenliste (für alle Kanten)
aus.(TVorgang=procedure(Ob:TObject))

Function TKnoten.Kantenzahlausgehend:Integer

Diese Funktionsmethode gibt die Anzahl der vom Knoten ausgehenden Kanten (Pfeilrichtung vom Knoten weg) zurück.Ungerichtete Kanten zählen dabei ebenfalls als ausgehend.

Function TKnoten.Kantenzahleingehend:Integer

Diese Funktionsmethode gibt die Anzahl der auf den Knoten hin eingehenden Kanten (Pfeilrichtung auf den Knoten hin) zurück.Ungerichtete Kanten zählen dabei ebenfalls als eingehend.

Function TKnoten.Kantenzahl:Integer

Diese Funktionsmethode gibt die Gesamtzahl aller Kanten,die den Knoten als Anfangs-oder Endknoten haben,zurück.

Function TKnoten.AnzahlSchlingen:Integer

Diese Funktionsmethode gibt die Anzahl der Schlingen(-Kanten) eines Knoten zurück.Schlingen sind solche Kanten,deren Anfangs- und Endknoten gleich ist.

Function TKnoten.Grad(Gerichtet:Boolean):Integer

Diese Funktionsmethode gibt den Grad eines Knoten zurück.Der Parameter gerichtet bestimmt,ob der Grad für einen Knoten in einem gerichteten Graph oder ungerichteten Graph bestimmt wird.

Function TKnoten.IstimPfad:Boolean

Diese Function gibt zurück,ob die Pfadliste des Knoten leer ist oder nicht.

Procedure TKnoten.LoeschePfad

Diese Methode löscht die Einträge in der Pfadliste des Knoten.

Procedure TKnoten.ErzeugeallePfade

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen Pfade zu allen anderen Knoten des Graphen,falls existent.Alle Pfade werden in der Pfadliste des Startknoten gespeichert.Außerdem enthält die Pfadliste aller anderen Knoten außer dem Startknoten den Pfad vom Startknoten zu diesem Knoten. Falls keine Pfade existieren sind die entsprechenden Pfade leer (die Pfadliste enthält keinen Eintrag).

Procedure TKnoten.ErzeugeallePfadeZielKnoten(Kno:TKnoten)

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen Pfade zu allen anderen Knoten des Graphen,falls existent.Alle Pfade zum Knoten Kno werden in der Pfadliste des Knotens Kno gespeichert.
Falls keine Pfade existieren ist die entsprechende Pfadliste leer (die Pfadliste enthält keinen Eintrag).

Function TKnoten.PfadzumZielknoten
(Kno:TKnoten;Ka:TKante):Boolean;

Diese Methode überprüft,ob ein Pfad vom Knoten zum Zielknoten Kno existiert.Dabei wird die Kante Ka vom Typ TKante als Kante eines Pfades vom Knoten zum Zielknoten gesperrt.Wenn Ka gleich nil ist,sind alle Kanten für Pfade zugelassen.

Procedure TKnoten.ErzeugeTiefeBaumPfade(Preorder:Boolean)

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen tiefen Baumpfade zu allen anderen Knoten des Graphen,falls existent.Alle Pfade werden in der Pfadliste des Startknoten gespeichert.Außerdem enthält die Pfadliste aller anderen Knoten außer dem Startknoten den Pfad vom Startknoten zu diesem Knoten. Falls keine Pfade existieren sind die entsprechenden Pfade leer (die Pfadliste enthält keinen Eintrag).Der Parameter Preorder bestimmt (Preorder=true),daß der Baum (Graph) gemäß der Ordnung Preorder durchlaufen wird,ansonsten wird die Ordnung Postorder benutzt.

Procedure TKnoten.ErzeugeWeiteBaumPfade

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen weiten Baumpfade zu allen anderen Knoten des Graphen,falls existent.Alle Pfade werden in der Pfadliste des Startknoten gespeichert.Außerdem enthält die Pfadliste aller anderen Knoten

außer dem Startknoten den Pfad vom Startknoten zu diesem Knoten. Falls keine Pfade existieren sind die entsprechenden Pfade leer (die Pfadliste enthält keinen Eintrag).

Procedure TKnoten.ErzeugeKreise

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen Kreise zu allen anderen Knoten des Graphen,falls existent.Alle Pfade werden in der Pfadliste des Startknoten gespeichert.
Falls keine Pfade existieren ist die Pfadliste leer (die Pfadliste enthält keinen Eintrag).

Procedure TKnoten.ErzeugeminimalePfade(Wert:TWert)

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen minimalen Pfade bezüglich der Funktion (Bewertung der Kanten bzw. Pfade) Wert vom Typ TWert zu allen anderen Knoten des Graphen,falls existent.Alle Pfade werden in der Pfadliste des Startknoten gespeichert.Außerdem enthält die Pfadliste aller anderen Knoten außer dem Startknoten den Pfad vom Startknoten zu diesem Knoten.Falls keine Pfade existieren sind die entsprechenden Pfade leer (die Pfadliste enthält keinen Eintrag).Der Algorithmus erzeugt alle Pfade zu den Knoten des Graphen und sortiert anschließend die Pfadlisten der einzelnen Knoten der Pfadlänge nach. (TWert=function(Ob:TObject):Extended)

Procedure TKnoten.ErzeugeminimalePfadenachDijkstra(Wert:TWert)

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen minimalen Pfade bezüglich der Funktion (Bewertung der Kanten bzw. Pfade) Wert vom Typ TWert zu allen anderen Knoten des Graphen,falls existent.Alle Pfade werden in der Pfadliste des Startknoten gespeichert.Außerdem enthält die Pfadliste aller anderen Knoten außer dem Startknoten den Pfad vom Startknoten zu diesem Knoten.Falls keine Pfade existieren sind die entsprechenden Pfade leer (die Pfadliste enthält keinen Eintrag).
Die Methode arbeitet nach dem Algorithmus von Dijkstra.
(TWert=function(Ob:TObject):Extended)

Procedure TKnoten.SortierePfadliste(Wert:TWert)

Diese Methode sortiert die Pfade der Pfadliste eines Knoten gemäß der Bewertung (Function) Wert vom Typ TWert für Kanten bzw. Pfade in aufsteigender Ordnung.
(TWert=function(Ob:TObject):Extended)

Function TKnoten.AnzahlPfadZielknoten: Integer

Diese Funktionsmethode gibt die Anzahl der Knoten im Graph

zurück,zu dem vom Knoten als Startknoten aus Pfade existieren.Der Startknoten wird dabei nicht mitgezählt.

Function TKnoten.MinimalerPfad(Wert:TWert):TPfad

Diese Funktionsmethode gibt den minimalen Pfad der Pfadliste eines Knoten bzgl. der Pfad- bzw. Kantenbewertung Wert vom Typ TWert zurück. (TWert=function(Ob:TObject):Extended)

Function TKnoten.MaximalerPfad(Wert:TWert):TPfad

Diese Funktionsmethode gibt den maximalen Pfad der Pfadliste eines Knoten bzgl. der Pfad- bzw. Kantenbewertung Wert vom Typ TWert zurück. (TWert=function(Ob:TObject):Extended)

Function TKnoten.KnotenistKreisknoten:Boolean

Diese Funktionsmethode gibt zurück,ob ein Knoten Teil eines Kreises im Graphen ist.

Methoden von TGraph:

TGraph ist Nachkomme von TObject.Der Datentyp enthält folgende Datenfelder:

```
Knotenliste_:TKnotenliste;
Kantenliste_:TKantenliste;
Wertposition_:Integer;
Wertliste_:TStringlist;
Unterbrechung_:Boolean;
```

Knotenliste_ vom Typ TKnotenliste und Kantenliste_ vom Typ TKantenliste sind Zeiger auf die Knotenliste und die Kantenliste des Graphen.Wertposition_ vom Typ Integer gibt die Position des Datenfeldes an,die von der Property Wert als String zurückgegeben wird (Property Position).

Wertliste_ vom Typ TStringlist enthält die Datenfelder aller Datenfelder (evtl. umgewandelt) als Strings.Die Wertliste_ steht für Datenfelder zur Verfügung,die ein Anwender später für einen von TGraph durch Vererbung abgeleiteten Datentyp erzeugt.Das Feld Unterbrechung_ dient dazu,um eine Markierung (Flag) für einen gewünschten Abbruch des momentan ausgeführten Algorithmus zu speichern (Abbruch_=true).

Darüberhinaus enthält der Datentyp eine Reihe von Propertys,um über sie auf die oben genannten Datenfelder zuzugreifen.

Propertys:

```
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 Abbruch:Boolean read WelcheUnterbrechung write
SetzeUnterbrechung;
Property Wert:string read Wertlesen write Wertschreiben;
```

Durch die folgenden Methoden von TGraph (Proceduren und Functionen) wird auf die Propertys lesend und schreibend zugegriffen:

```
Function TGraph.WelcheKnotenliste:TKnotenliste;
Procedure TGraph.SetzeKnotenliste(K:TKnotenliste);
Function TGraph.WelcheKantenliste:TKantenliste;
Procedure TGraph.SetzeKantenliste(K:TKantenliste);
Procedure TGraph.SetzeWertposition(P:Integer);
Function TGraph.WelcheWertposition:Integer;
Function TGraph.WelcheWertliste:TStringlist;
Procedure TGraph.SetzeWertliste(W:TStringlist);
Function TGraph.Wertlesen:string;
Procedure TGraph.Wertschreiben(S:string);
Procedure TGraph.SetzeUnterbrechung(Ub:Boolean);
Function TGraph.WelcheUnterbrechung:Boolean;
```

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

```
Function TGraph.Wertlisteschreiben:TStringlist;virtual;abstract;
```

Diese Funktionsmethode wird als virtual und abstract definiert,damit durch Vererbung erzeugte Nachfolgerobjekte mittels ihrer Wertlisteschreiben-Methode durch die Property Wert ihre Datenfelder zurückgeben können.Durch die Property Position wird jeweils das entsprechende Datenfeld ausgewählt.Die Daten sind als Strings in dem Feld Wertliste_ gespeichert.

```
Procedure TGraph.Wertlistelesen;virtual;abstract;
```

Diese Methode wird als virtual und abstract definiert,damit den Datenfelder der duch Vererbung erzeugten Nachfolgerobjekte mittels ihrer Werlistelesen-Methode durch die Property Wert Werte zugewiesen werden können.Durch die Property Position wird jeweils das entsprechende Datenfeld ausgewählt. Die Daten werden

als Strings in dem Feld Wertliste_ gespeichert.

Constructor TGraph.Create

Diese Methode ist der Constructor für TGraph.

Procedure TGraph.Free

Diese Methode ist der einfache Destructor von TGraph und entfernt Instanzen des Datentyps (ohne Knoten und Kanten) aus dem Speicher.

Procedure TGraph.FreeAll

Diese Methode ist der erweiterte Destructor und entfernt Instanzen von TGraph einschließlich der in der Knotenliste gespeicherten Knoten und der in der Kantenliste gespeicherten Kanten aus dem Speicher

Procedure TGraph.ImGraphKnotenundKantenloeschen

Diese Methode löscht alle Kanten und Knoten des Graphen aus dem Speicher und stellt einen leeren Graphen her.

Function TGraph.Leer:Boolean

Diese Function gibt zurück,ob der Graph leer ist,d.h. keine Knoten oder Kanten enthält.

Procedure TGraph.KnotenEinfuegen(Kno:TKnoten)

Diese Methode fügt den Knoten Kno in den Graph ein.

Procedure TGraph.Knotenloeschen(Kno:TKnoten)

Diese Methode löscht den Knoten Kno im Graph.Der Knoten (die Instanz) wird aus dem Speicher gelöscht.

Procedure TGraph.KanteEinfuegen
(Ka:TKante;Anfangsknoten,EndKnoten:TKnoten;Gerichtet:Boolean)

Diese Methode fügt eine Kante Ka vom Typ TKante mit dem Anfangsknoten Anfangsknoten und dem Endknoten Endknoten jeweils vom Typ TKnoten in den Graph ein.Dabei beschreibt Gerichtet vom Typ Boolean,ob es sich um eine gerichtete oder ungerichtete Kante handelt.Die Informationen über Anfangsknoten,Endknoten und Gerichtet,die in Ka gespeichert sind,werden durch die eben genannten Parameter der Procedure überschrieben.Wenn Ka,Anfangsknoten oder Endknoten nil sind,wird keine Kante eingefügt.

Procedure TGraph.EinfuegenKante(Ka:TKante)

Diese Methode fügt die Kante Ka vom Typ TKante in den Graph ein.Dazu enthält Ka alle notwendigen Informationen:Anfangsknoten,Endknoten vom Typ TKnoten und Gerichtet vom Typ Boolean.Wenn Ka,Anfangsknoten oder Endknoten nil sind,wird keine Kante eingefügt.

Procedure TGraph.Kanteloeschen(Ka:TKante)

Diese Methode löscht die Kante Ka aus dem Graph und (die Instanz) aus dem Speicher.

Procedure TGraph.LoescheKantenbesucht

Diese Methode setzt bei allen Kanten des Graphen die Markierung Besucht auf false.

Procedure TGraph.LoescheKantenerreicht

Diese Methode setzt bei allen Kanten des Graphen die Markierung Erreicht auf false.

Procedure TGraph.LoescheKnotenbesucht

Diese Methode setzt bei allen Knoten des Graphen die Markierung Besucht auf false.

Procedure TGraph.FuerjedenKnoten(Vorgang:TVorgang)

Diese Methode führt die Anweisungen der Procedure Vorgang vom Typ TVorgang für jeden Knoten des Graphen aus.(TVorgang=procedure(Ob:TObject))

Procedure TGraph.FuerjedeKante(Vorgang:TVorgang)

Diese Methode führt die Anweisungen der Procedure Vorgang vom Typ TVorgang für jede Kante des Graphen aus. (TVorgang=procedure(Ob:TObject))

Function TGraph.Anfangsknoten:TKnoten

Diese Funktionsmethode gibt den Anfangsknoten des Graphen zurück.Das ist der erste in die Knotenliste des Graphen eingefügte Knoten.Wenn der Graph leer ist,wird nil zurückgegeben.

Function TGraph.Anfangskante:TKante

Diese Funktionsmethode gibt die Anfangskante des Graphen zurück.Das ist die erste in die Kantenliste des Graphen einge-

fügte Kante.Wenn der Graph leer ist,wird nil zurückgegeben.

Function TGraph.AnzahlKanten:Integer

Diese Funktionsmethode gibt die Anzahl der Kanten des Graphen ohne die Schlingen des Graphen zurück.Schlingen sind Kanten mit gleichem Anfangs-und Endknoten.

Function TGraph.AnzahlKantenmitSchlingen: Integer

Die Funktionsmethode gibt die Anzahl der Kanten des Graphen einschließlich der Schlingen zurück.Schlingen sind Kanten mit gleichem Anfangs- und Endknoten.

Function TGraph.AnzahlKnoten:Integer

Diese Funktionsmethode gibt die Anzahl der Knoten des Graphen zurück.

Function TGraph.AnzahlgerichteteKanten:Integer

Diese Funktionsmethode gibt die Anzahl der gerichteten Kanten (einschließlich Schlingen.d.h. Kanten mit gleichem Anfangs-und Endknoten) des Graphen zurück.

Function TGraph.AnzahlungerichteteKanten:Integer

Diese Funktionsmethode gibt die Anzahl der ungerichteten Kanten (einschließlich Schlingen,d.h. Kanten mit gleichem Anfangs-und Endknoten) des Graphen zurück.

Function TGraph.AnzahlKomponenten:Integer

Diese Funktionsmethode gibt die Anzahl der Komponenten des Graphen zurück.

Function TGraph.AnzahlparallelerKanten:Integer

Diese Funktionsmethode gibt die Anzahl paralleler Kanten des Graphen zurück.Parallele Kanten verbinden dieselben Knoten und haben die gleiche Richtung.

Function TGraph.AnzahlantiparallelerKanten:Integer

Diese Funktionsmethode gibt die Anzahl antiparalleler Kanten des Graphen zurück.Antiparallele Kanten verbinden dieselben Knoten und haben entgegengesetzte Richtung.

Function TGraph.AnzahlparallelerKantenungerichtet:Integer

Diese Funktionsmethode gibt die Anzahl paralleler ungerichteter Kanten des Graphen zurück.Parallele ungerichtete Kanten verbinden dieselben Knoten und sind ungerichtet.

Function TGraph.Kantensumme(Wert:TWert):Extended

Diese Funktionsmethode gibt die Summe der Werte aller Kanten des Graphen zurück gemäß der Kantenbewertung (Function) Wert vom Typ TWert. (TWert=function(Ob:TObject):Extended)

Function TGraph.Kantenprodukt(Wert:TWert):Extended

Diese Funktionsmethode gibt das Produkt der Werte aller Kanten des Graphen zurück gemäß der Kantenbewertung (Function) Wert vom Typ TWert. (TWert=function(Ob:TObject):Extended)

Function TGraph.ListemitKnotenInhalt(Sk:TString):TStringlist

Diese Funktionsmethode gibt eine Stringliste vom Typ TStringlist zurück,deren Elemente aus den Rückgabewerten der Function Sk vom Typ TString angewendet auf jeden Knoten bestehen. (TString=function(Ob:TObject):string)

Function TGraph.InhaltallerKnoten(Sk:TString):string

Diese Funktionsmethode gibt einen String zurück,dessen Teilstrings aus den Rückgabewerten der Function Sk vom Typ TString angewendet auf jeden Knoten bestehen.Die Teilstrings werden durch Leerzeichen getrennt. (TString=function(Ob:TObject):string)

Function TGraph.ListemitInhaltKantenoderKnoten
(Sk:TString):TStringlist

Diese Funktionsmethode gibt eine Stringliste vom Typ TStringlist zurück,deren Elemente aus den Rückgabewerten der Function Sk vom Typ TString angewendet auf jede Kante bestehen.Statt des Kanteninhalts können beispielweise auch Anfangs-und/oder Endknoten der Kanten mittels der Function Sk ausgewählt werden (TString=function(Ob:TObject):string)

Function TGraph.InhaltallerKantenoderKnoten(Sk:Tstring):string

Diese Funktionsmethode gibt einen String zurück,dessen Teilstrings aus den Rückgabewerten der Function Sk vom Typ TString angewendet auf jede Kante bestehen.Statt des Kanteninhalts können beispielweise auch Anfangs-und/oder Endknoten der Kanten mittels der Function Sk ausgewählt werden.Die Teilstrings werden durch Leerzeichen getrennt.

(TString=function(Ob:TObject):string)

Procedure TGraph.Pfadlistenloeschen

Diese Methode löscht die Pfadlisten aller Knoten (erzeugt leere Pfadlisten).

Procedure TGraph.SortiereallePfadlisten(Wert:TWert)

Diese Methode sortiert die Pfadlisten aller Knoten des Graphen aufsteigend gemäß der Pfad-bzw.Kantenbewertung Wert vom Typ TWert. (TWert=function(Ob:TObject):Extended)

Function TGraph.BestimmeminimalenPfad
(Kno1,Kno2:TKnoten;Wert:Twert):TPfad

Diese Funktionmethode bestimmt den minimalen Pfad zwischen den Knoten Kno1 und Kno2 vom Datentyp TKnoten gemäß dem Algorithmus von Ford.Der Pfad wird als Graph,dessen Kantenliste der gesuchte minimale Pfad ist,vom Typ TGraph und gleichzeitig TPfad zurückgegeben.Als Bewertung der Pfadlängen dient dabei die Funktion Wert vom Typ TWert,wobei auch negative Kantenbewertungen (nur) auf gerichteten Graphen zulässig sind,allerdings keine negativen Kreise.Die Pfadliste aller Knoten enthält im Pfad(0) nach Abschluß der Methode die kürzesten Pfade von Kno1 zu Ihnen und der kürzeste Pfad zu Kno2 wird außerdem als Rückgabewert der Funktion zurückgegeben.
(TWert=function(Ob:TObject):Extended)

Function TGraph.GraphhatKreise:Boolean

Die Funktionsmethode testet,ob der Graph Kreise hat.

Function TGraph.GraphhatgeschlosseneEulerlinie
(Gerichtet:Boolean):Boolean

Die Funktionsmethode testet,ob der Graph mindestens einen geschlossene Eulerlinie hat.Der Parameter Gerichtet bestimmt,ob die Kanten des Graphen mit ihrer vorgegeben Richtung oder als ungerichtet untersucht werden.

Function TGraph.GraphhatoffeneEulerlinie(var Kno1,Kno2:TKnoten;
Gerichtet:Boolean):Boolean

Die Funktionsmethode testet,ob der Graph mindestens einen offenen Eulerlinienpfad zwischen zwei Knoten hat.Die Knoten werden als Referenzparameter Kno1 und Kno2 zurückgegeben.Der Parameter Gerichtet bestimmt,ob die Kanten des Graphen mit ihrer vorgegeben Richtung oder als ungerichtet untersucht werden.

Function TGraph.Kopie:TGraph

Diese Funktionsmethode gibt eine Kopie des Graphen zurück.Dabei werden nur die Knotenliste (-zeiger) und die Kantenliste (-zeiger) sowie der Graph (-zeiger) kopiert.Knoten und Kanten sind dieselben Elemente (Zeiger) wie im ursprünglichen Graphen.

Function TGraph.KanteverbindetKnotenvonnach (Kno1,Kno2:TKnoten):Boolean

Die Funktionsmethode testet,ob eine Verbindungskante zwischen den Knoten Kno1 und Kno2 vom Datentyp TKnoten besteht.

Function TGraph.ErsteKantevonKnotenzuKnoten (Kno1,Kno2:TKnoten):TKante

Diese Funktionsmethode gibt die als erstes in der ausgehenden Kantenliste von Kno1 eingefügte Kante,die als Endknoten Kno2 hat,zurück.Falls keine solche Kante existiert,wird nil zurückgegeben.

Function TGraph.ErsteSchlingezuKnoten(Kno:TKnoten):TKante

Diese Funktionsmethode gibt die als erstes in der ausgehenden Kantenliste von Kno eingefügte Schlinge,die als Endknoten Kno hat,zurück.Eine Schlinge hat als Anfangs-und Endknoten den gleichen Knoten.Falls keine solche Kante existiert,wird nil zurückgegeben.

Function TGraph.GerichteterGraphistpaarundohneisoliertenKnoten:Boolean

Diese Funktionsmethode testet,ob der Digraph paar und ohne isol.Knoten ist.

Function TGraph.GerichteterGraphistBinaerbaum:Boolean

Diese Funktionsmethode testet,ob der gerichtete Graph als Binärbaum aufgefasst werden kann.

Beschreibung der Unit UInhGrph:

Die Unit enthält alle Methoden,die für die Beschreibung eines Graphen,in dessen Kanten und Knoten Dateninhalte (dazu zählen auch Koordinaten,Farben und Zeichenstile) gespeichert sind,erforderlich sind und erweitert damit die Datentypen der Unit UGraph.Von den dortigen Datentypen TKnoten,TKante und TGraph leiten sich durch Vererbung die Datentypen TInhaltsknoten, TInhaltskante und TInhaltsgraph dieser Unit ab.

Methoden von TInhaltsknoten:

Der Datentyp TInhaltsknoten ist Nachkomme von TKnoten. Er dient zur Beschreibung und Speicherung der Inhalte der Knoten eines Graphen.Er enthält folgende zusätzliche Datenfelder:

```
X_,Y_:Integer;
Farbe_:TColor;
Stil_:TPenstyle;
Typ_:char;
Radius_:Integer;
Inhalt_:string;
```

Die Datenfelder X_ und Y_ speichern die x-bzw- y-Koordinate eines Knotenmittelpunkts auf der Zeichenfläche,das Feld Farbe_ speichert die Zeichenfarbe des Knotenkreises,das Feld Stil_ speichert den Zeichenstil (durchgezogen,gestrichelt usw.) des Knotenkreises,Typ_ dient zur Aufnahme der Kennzeichnung einer Typkennzeichnung des Knotens mittels des Datentyps Char (Typ wird im vorliegenden Programm nicht verwendet) und Radius_ speichert den Radius des Knotenkreises.Das Datenfeld Inhalt_ dient zum Speichern des Knoteninhalts als String.Zahlen,die gespeichert werden sollen,müssen zuerst konvertiert werden.Der Zugriff erfolgt über die Property Wert von TKnoten mittels Position und Wertliste.

Darüberhinaus enthält der Datentyp eine Reihe von Propertys,um über sie auf die oben genannten Datenfelder zuzugreifen:

Propertys:

```
Poperty X:Integer read Lesex write Schreibex;
Poperty Y:Integer read Lesey write Schreibey;
Poperty Radius:Integer read Welcherradius write Setzeradius;
Poperty Farbe:TColor read WelcheFarbe write SetzeFarbe;
Poperty Stil:Tpenstyle read WelcherStil write SetzeStil;
Poperty Typ:char read WelcherTyp write SetzeTyp;
```

Durch die folgenden Methoden von TInhaltsknoten (Proceduren und Functionen) wird auf die Propertys lesend und schreibend zugegriffen:

```
Function TInhaltsknoten.Lesex:Integer
Procedure TInhaltsknoten.Schreibex(X:Integer)
Function TInhaltsknoten.Lesey:Integer
Procedure TInhaltsknoten.Schreibey(Y:Integer)
Function TInhaltsknoten.Welcherradius:Integer
Procedure TInhaltsknoten.Setzeradius(R:Integer)
Function TInhaltsknoten.WelcheFarbe:TColor
Procedure TInhaltsknoten.SetzeFarbe(F:TColor)
Function TInhaltsknoten.WelcherStil:TPenstyle
Procedure TInhaltsknoten.SetzeStil(T:TPenstyle)
```

Function TInhaltsknoten.WelcherTyp:Char
Procedure TInhaltsknoten.SetzeTyp(Typ:Char)

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.
Function TInhaltsknoten.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TInhaltsknoten mit dem Inhalt von Inhalt_ als Element und gibt sie zurück.Allgemein enthält die Wertliste die Inhalte der Datenfelder des Datentyps (evtl. umgewandelt) als Strings.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TKnoten zugegriffen.

Procedure TInhaltsknoten.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps (hier: Inhalt_). Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TKnoten zugegriffen.

Constructor TInhaltsknoten.Create

Diese Methode ist der Constructor von TInhaltsknoten.

Procedure TInhaltsknoten.Free

Diese Methode ist der einfache Destructor von TInhaltsknoten entfernt Instanzen des Datentyps (ohne die in der aus-und eingehenden Kantenliste gespeicherten Kanten) aus dem Speicher.

Procedure TInhaltsknoten.Freeall

Diese Methode ist der erweiterte Destructor entfernt Instanzen von TInhaltsknoten einschließlich der in der ausgehenden und eingehenden Kantenliste gespeicherten Kanten aus dem Speicher.

Procedure TInhaltsknoten.ZeichneKnoten(Flaeche:TCanvas)

Diese Methode zeichnet den Knoten auf der durch Flaeche vom Typ TCanvas vorgegebenen Objekt-Zeichenfläche.

Procedure TInhaltsknoten.ZeichneDruckKnoten(Flaeche:TCanvas;
Faktor:Integer);

Diese Methode zeichnet den Knoten auf der durch Flaeche vom Typ TCanvas vorgegebenen Objekt-Zeichenfläche (geeignet für die Zeichenfläche des Druckers).Die X- und Y-Koordinaten und der Radius jedes Knoten werden um den Faktor Faktor gestreckt.

Procedure TInhaltsknoten.Knotenzeichnen
(Flaeche:TCanvas;Demo:Boolean;Pausenzeit:Integer)

Diese Methode zeichnet den Knoten auf der durch Flaeche vom Typ TCanvas vorgegebenen Objekt-Zeichenfläche zunächst rot markiert und danach wieder in der Farbe schwarz.Dazwischen wird bei Demo=true eine Pause der Länge Pausenzeit eingelegt,und es ertönt ein kuzer Ton.Bei Demo=false erfolgt keine Pause und kein Ton.

Procedure TInhaltsknoten.AnzeigePfadliste
(Flaeche:TCanvas;Ausgabe:TLabel;var Sliste:
TStringlist;Zeichnen:Boolean;LetzterPfad:Boolean)

Diese Methode zeichnet die zum Knoten gehörende Pfadliste (falls vom Typ TInhaltsgraph) auf der durch Fläche vom Typ TCanvas vorgegebenen Objekt-Zeichenfläche (für den Fall zeichnen=true).In dem Referenzparameter Sliste vom Typ TStringlist wird (falls vom Typ TInhaltsgraph) der Pfad als Liste von Kanteninhalten gespeichert und zurückgegeben.Der Werteparameter zeichnen bestimmt,ob die Pfadliste gezeichnet wird (zeichnen=true) oder nur in Sliste gespeichert wird (zeichnen=false).Wenn zeichnen true ist,wird die Pfadliste auf dem Label Ausgabe vom Typ TLabel ausgegeben,vorausgesetzt Ausgabe ist ungleich nil.Wenn LetzterPfad true ist,wird die markierte Anzeige (bei Zeichnen true) des letzten Pfades der Pfadliste nicht gelöscht.

Function TInhaltsknoten.ErzeugeminmaxKreise(Minmax:
Boolean):TKantenliste;

Diese Funktionsmethode bestimmt die Kreise mit der kleinsten oder mit der größten Kantenzahl (mit mehr als zwei Kanten),die durch den Knoten verlaufen und gibt sie jeweils als Kantenliste zurück.Der Parameter Minmax vom Datentyp Boolean bestimmt ob der kleinste oder der größte Pfad bestimmt wird.

Procedure TInhaltsknoten.ErzeugeKreisevonfesterLaenge
(Laenge:Integer);

Diese Methode bestimmt alle Kreise der festen Länge Länge durch den aktuellen Knoten und speichert die Pfade in der Pfadliste des Knotens.

Methoden von TInhaltskante:

Der Datentyp TInhaltskante ist Nachkomme von TKante.Er dient zur Beschreibung und Speicherung der Inhalte der Kanten eines Graphen.Er enthält folgende zusätzliche Datenfelder:

```
Farbe_:TColor;
Stil_:TPenstyle;
Weite_:Integer;
Typ_:char;
Inhalt_:string;
```

Die Datenfelder Farbe_ und Stil_ speichern die Farbe und den Stil (durchgezogen,gestrichelt usw.),in der die Kanten gezeichnet werden.Weite_ gibt die Kantenweite an.D.h. welche Abstand in Pixeln hat der Mittelpunkt der Kante von der Verbindungsstrcke zwischen den beiden Knotenmittelpunkten der zu der Kante gehörenden Knoten.Typ beschreibt den Datentyp des Kanten (’s’: string,’r’: real,’i’: Integer) als char-Zeichen.Das Datenfeld Inhalt_ dient zum Speichern des Kanteninhalts als String.Zahlen,die gespeichert werden sollen,müssen zuerst konvertiert werden.Der Zugriff erfolgt über die Property Wert von TKante mittels Position und Wertliste.

Darüberhinaus enthält der Datentyp eine Reihe von Propertys,um über sie auf die oben genannten Datenfelder zuzugreifen:

Propertys:

```
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;
```

Durch die folgenden Methoden von TInhaltskante (Proceduren und Functionen) wird auf die Propertys lesend und schreibend zugegriffen:

```
Function TInhaltskante.Welchertyp:char
Procedure TInhaltskante.Setzetyp(Typ:char)
Function TInhaltskante.Welcheweite:Integer
Procedure TInhaltskante.SetzeWeite(Weite:Integer)
Function TInhaltskante.WelcheFarbe:TColor
Procedure TInhaltskante.SetzeFarbe(F:TColor)
Function TInhaltskante.WelcherStil:TPenstyle
Procedure TInhaltskante.SetzeStil(T:TPenstyle)
```

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TInhaltskante.Create

Diese Methode ist der Constructor für Instanzen vom Typ TInhaltskante.

Procedure TInhaltskante.Free

Dieser einfache Destructor löscht eine Instanz vom Datentyp TInhaltskante aus dem Speicher.Die Felder Anfangsknoten_ und Endknoten_ werden nicht gelöscht.

Procedure TInhaltskante.Freeall

Dieser erweiterte Destructor entfernt Instanzen des Datentyps TInhaltskante einschließlich den Objekten Anfangsknoten_ und Endknoten_ aus dem Speicher.

Function TInhaltskante.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TInhaltskante mit dem Inhalt von Inhalt_ als Element und gibt sie zurück.Allgemein enthält die Wertliste die Inhalte der Datenfelder des Datentyps (evtl. umgewandelt) als Strings.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TKante zugegriffen.

Procedure TInhaltskante.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps (hier: Datenfeld Inhalt_). Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TKante zugegriffen.

Function TInhaltskante.MausklickaufKante(X,Y:Integer):Boolean;

Diese Funktionsmethode testet,ob sich in der Nähe des Bildschirmpunkt mit den Koordinaten X und Y (X und Y werden beispielsweise als Mausklickpunkt erzeugt) eine Kante vom Typ TInhaltskante befindet.

Procedure TInhaltskante.ZeichneKante(Flaeche:TCanvas)

Diese Methode zeichnet die Kante auf der durch Flaeche vom Typ TCanvas vorgegebenen Objekt-Zeichenfläche.

Procedure TInhaltskante.ZeichneDruckKante
(Flaeche:TCanvas;Faktor:Integer);

Diese Methode zeichnet die Kante auf der durch Flaeche vom Typ TCanvas vorgegebenen Objekt-Zeichenfläche (geeignet als Druckerfläche).Um den Faktor Faktor wird die Weite der Kante gestreckt.

Procedure TInhaltskante.Kantezeichnen

(Flaeche:TCanvas;Demo:Boolean;Pausenzeit:Integer)

Diese Methode zeichnet die Kante auf der durch Flaeche vom Typ TCanvas vorgegebenen Objekt-Zeichenfläche zunächst rot markiert und danach wieder in der Farbe schwarz.Dazwischen wird bei Demo=true eine Pause der Länge Pausenzeit eingelegt,und es ertönt ein kurzer Ton.Bei Demo=false erfolgt keine Pause und kein Ton.

Methoden von TInhaltsgraph:

Der Datentyp TInhaltsgraph ist Nachkomme von TGraph.Er dient zur Beschreibung und Speicherung eines Graphen mit Knoten- und Kanteninhalten sowie (möglichen zusätzlichen) Inhaltsfeldern des Graphen.Er enthält folgende Datenfelder:

```
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;
```

Die Felder Knotenwertposition_ und Kantenwertposition_ speichern die Position des zur Ausgabe der Property Wert von Knoten und Kanten verwendeten Datenfeldes der Wertliste des Graphen.Demo_ speichert eine Markierung (Flag),ob der Ablauf von Algorithmen im Demo-Modus mit einer Verzögerungszeit,die im Feld Pausenzeit_ enthalten ist,ablaufen soll.Zustand_ und Stop_ sind Felder,die den Ablauf eines Algorithmus bei ereignisorientierten Programmierung steuern (z.B. abbrechen).Die Felder Knotengenauigkeit- und Kantengenauigkeit_ speichern die Stellenzahlen,mit der Wert von Knoten und Kanten des Graphen angezeigt wird. Radius_ enthält den Radius,mit dem die Kreise von Knoten gezeichnet werden.Liniendicke_speichert die Liniendicke,mit der Knoten und Kanten gezeichnet werden.Graphistgespeichert_ ist eine Markierung (Flag),die festhält,ob der aktuell (veränderte) Graph schon gespeichert ist.
Im Feld Inhaltsknotenclass_ wird der Objekttyp eines von

TInhaltsknoten vererbten Objekttyps (oder TInhaltsknoten selber,das ist die Voreinstellung) und im Feld Inhaltskantenclass_ wird der Objecttyp eines von TInhaltskante vererbten Objecttyps (oder TInhaltskante selber,das ist die Voreinstellung) gespeichert.Durch die Verwendung von virtuellen Constructoren für Instanzen von Typ TInhaltsknoten/TKnoten bzw. vom Typ TInhaltskante/TKante und TInhaltsgraph/TGraph ist es so möglich,den Datentyp der Knoten und Kanten des Graphen nachträglich festzulegen und trotzdem auf die vordefinierten Methoden von TInhaltsknoten, TInhaltskante und TInhaltsgraph zuzugreifen.Die Felder MomentaneKnotenliste_ und MomentaneKantenliste_ dienen dazu,Knoten- bzw. Kantenlisten zwischenspeichern zu können.Dateiname_ enthält den Dateinamen des aktuell geladenen Graphen.Die Knoten K1_, K2_, K3_, K4_ sind Hilfsvariablen,die bei der Programmierung von Ereignisorientierten Methoden oder zur Speicherung von Zwischenergebnissen wie dem letzten mit der Maus angeklickten Knoten nützlich sind.

Darüberhinaus enthält der Datentyp eine Reihe von Propertys,um über sie auf die oben genannten Datenfelder zuzugreifen.

Propertys:

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;

Durch die folgenden Methoden von TInhaltskante (Proceduren und Functionen) wird auf die Propertys lesend und schreibend zugegriffen:

Function TInhaltsgraph.WelcheKnotenwertposition:Integer
Procedure TInhaltsgraph.SetzeKnotenwertposition(P:Integer)
Function TInhaltsgraph.WelcheKantenwertposition:Integer
Procedure TInhaltsgraph.SetzeKantenwertposition(P:Integer)
Function TInhaltsgraph.WelcheWartezeit:Integer
Procedure TInhaltsgraph.SetzeWartezeit(Wz:Integer)
Function TInhaltsgraph.WelcherDemomodus:Boolean
Procedure TInhaltsgraph.SetzeDemomodus(D:Boolean)
Function TInhaltsgraph.WelcherEingabezustand:Boolean
Procedure TInhaltsgraph.SetzeEingabezustand(Ezsd:Boolean)
Function TInhaltsgraph.WelcherStopzustand:Boolean
Procedure TInhaltsgraph.SetzeStopzustand(Stop:Boolean)
Function TInhaltsgraph.WelcheKnotengenauigkeit:Integer
Procedure TInhaltsgraph.SetzeKnotengenauigkeit(G:Integer)
Function TInhaltsgraph.WelcheKantengenauigkeit:Integer
Procedure TInhaltsgraph.SetzeKantengenauigkeit(G:Integer)
Function TInhaltsgraph.WelcherKnotenradius:Integer
Procedure TInhaltsgraph.SetzeKnotenradius(R:Integer)
Function WelcheLiniendicke:Integer;
Procedure SetzeLiniendicke(D:Integer);
Function TInhaltsgraph.IstGraphgespeichert:Boolean
Procedure TInhaltsgraph.SetzeGraphgespeichert(Gesp:Boolean)
Function TInhaltsgraph.WelcherDateiname:string
Procedure TInhaltsgraph.SetzeDateiname(N:string)
Procedure TInhaltsgraph.SetzeletztenMausklickKnoten
(Kno:TInhaltsknoten)
Function TInhaltsgraph.WelcherletzteMausklickKnoten:
TInhaltsknoten
Function TInhaltsgraph.WelcheKnotenclass:TInhaltsknotenclass
Procedure TInhaltsgraph.SetzeKnotenclass
(Inhaltsclass:TInhaltsknotenclass)
Function TInhaltsgraph.WelcheKantenclass:TInhaltskanteclass

Procedure TInhaltsgraph.SetzeKantenclass
(Inhaltsclass:TInhaltskanteclass)
Function TInhaltsgraph.WelcheKnotenliste:TKnotenliste
Procedure TInhaltsgraph.SetzeKnotenliste(L:TKnotenliste)
Function TInhaltsgraph.WelcheKantenliste:TKantenliste
Procedure TInhaltsgraph.SetzeKantenliste(L:TKantenliste)
Function TInhaltsgraph.WelcherK1:TInhaltsknoten
Procedure TInhaltsgraph.SetzeK1(Kno:TInhaltsknoten)
Function TInhaltsgraph.WelcherK2:TInhaltsknoten
Procedure TInhaltsgraph.SetzeK2(Kno:TInhaltsknoten)
Function TInhaltsgraph.WelcherK3:TInhaltsknoten
Procedure TInhaltsgraph.SetzeK3(Kno:TInhaltsknoten)
Function TInhaltsgraph.WelcherK4:TInhaltsknoten
Procedure TInhaltsgraph.SetzeK4(Kno:TInhaltsknoten)

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TInhaltsgraph.Create

Das ist der Constructor für Instanzen vom Typ TInhaltsgraph.

Procedure TInhaltsGraph.FreeAll

Diese Methode ist der einfache Destructor von Tinhaltsgraph und entfernt Instanzen des Datentyps (ohne die Knoten und die Kanten der Knoten-.und Kantenliste) aus dem Speicher.

Procedure TInhaltsGraph.FreeAll

Diese Methode ist der erweiterte Destructor entfernt Instanzen von TInhaltsgraph einschließlich der in der Knotenliste gespeicherten Knoten und der in der Kantenliste gespeicherten Kanten aus dem Speicher

Function TInhaltsgraph.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die leere Wertliste von TInhaltsgraph und gibt sie zurück.Allgemein enthält die Wertliste die Inhalte der Datenfelder des Datentyps (evtl. umgewandelt) als Strings.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsgraph zugegriffen. Die Methode dient zum Überschreiben durch spätere vererbte Objekttypen.

Procedure TInhaltsgraph.Wertlistelesen

Diese Methode liest allgemein die Einträge der Wertliste als

Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps (hier: Inhalt_). Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TGraph zugegriffen.Die Methode dient zum Überschreiben durch Methoden von späteren davon abgeleiteten Objekttypen.

Procedure TInhaltsgraph.Demopause

Diese Methode erzeugt eine Pause mit der durch das Feld Pausenzeit_ vorgegebenen Anzahl von Sekunden.Bei negativer Pausenzeit_ wird eine Endlosschleife solange durchlaufen,bis der Wert wieder größer gleich Null ist.(Einzelschrittmodus)

Procedure TInhaltsgraph.FuegeKnotenein(Kno:TInhaltsknoten)

Diese Methode fügt einen Knoten Kno vom Typ TInhaltsknoten in den Graph ein.

Procedure TInhaltsgraph.FuegeKanteein(Kno1,Kno2:TInhaltsknoten; Gerichtet:Boolean;Ka:TInhaltskante)

Diese Methode fügt eine Kante Ka vom Typ TInhaltskante zwischen den Knoten Kno1 und Kno2 als Anfangs- und Endknoten vom Typ TInhaltsknoten in den Graph ein.Wenn Kno1 und Kno2 nicht im Graph vorhanden sind,werden diese Knoten ebenfalls eingefügt.Anfangs- und Endknoten,auf die Ka eventuell verweist,werden durch Kno1 und Kno2 ersetzt.Der Parameter Gerichtet bestimmt,ob eine gerichtete Kante (Gerichtet=true) oder ungerichtete Kante (Gerichtet=false) eingefügt wird.Die Richtung der Kante ist bei Gerichtet= true von Kno1 nach Kno2.Ein Inhalt der Kante muß zuvor in der Instanz Ka gespeichert worden sein (Ebenso für Kno1 und Kno2, falls nicht schon im Graph vorhanden.).Wenn Kno1, Kno2 oder Ka nil sind,wird keine Kante eingefügt.

Procedure TInhaltsgraph.EinfuegenKante(Ka:TInhaltskante)

Diese Methode fügt eine Kante Ka vom Typ TInhaltskante in den Graph ein.Anfangs- und Endknoten sowie Kanteninhalt und Richtung müssen durch die entsprechenden Datenfelder von Ka vorgegeben sein.Wenn Anfangsknoten,Endknoten oder Ka nil sind,wird keine Kante eingefügt.

Procedure TInhaltsgraph.LoescheKante(Kno1,Kno2:TInhaltsknoten)

Diese Methode durchsucht die Kantenliste des Graphen vorwärts und löscht die erste gefundene Kante,die als Anfang- bzw. Endknoten Kno1 bzw. Kno2 oder Kno2 bzw. Kno1 hat.

Procedure TInhaltsgraph.LoescheInhaltsKante(Ka:TInhaltskante)

Diese Methode durchsucht die Kantenliste des Graphen vorwärts und löscht die erste gefundene Kante,deren Zeiger mit dem Zeiger auf Kante Ka übereinstimmt.

Procedure TInhaltsgraph.ZeichneGraph(Flaeche:TCanvas)

Diese Methode zeichnet den Graphen auf der durch Flaeche vom Typ TCanvas bestimmten Objekt-Zeichenfläche.Die Methode benutzt die Property Wert für die Ausgabe von Knoten- und Kanteninhalten.Dabei werden die in Wertknotenposition und Wert-kantenposition bestimmten Positionen für die Ausgabe mittels der Property Wert für alle Knoten und Kanten berücksichtigt.Die Farben und Stile (sowie die Kantenweite),mit denen Knoten und Kanten gezeichnet werden,richtet sich dagegegen nach den in den einzelnen Knoten bzw. Kanten gespeicherten Werten.

Procedure TInhaltsgraph.ZeichneDruckGraph
(Flaeche:TCanvas;Faktor:Integer)

Diese Methode zeichnet den Graphen auf der durch Flaeche vom Typ TCanvas bestimmten Objekt-Zeichenfläche (geignet als Druckfläche).Die Methode benutzt die Property Wert für die Aus-gabe von Knoten- und Kanteninhalten.Dabei werden die in Wertknotenposition und Wertkantenposition bestimmten Positionen für die Ausgabe mittels der Property Wert für alle Knoten und Kanten berücksichtigt.Die Farben und Stile (sowie die Kantenweite),mit denen Knoten und Kanten gezeichnet werden,richtet sich dagegegen nach den in den einzelnen Knoten bzw. Kanten gespeicherten Werten.Die X- und Y-Koordinaten und der Radius aller Knoten und die Weite aller Kanten werden um den Faktor Faktor gestreckt.

Procedure TInhaltsgraph.Graphzeichnen
(Flaeche:TCanvas;Ausgabe:TLabel;Wert:TWert;
Sliste:TStringlist;Demo:Boolean;Pausenzeit:Integer;
Kantengenauigkeit:Integer)

Diese Methode zeichnet den Graphen auf der durch Flaeche vom Typ TCanvas bestimmten Object-Zeichenfläche zunächst rot mar-kiert und danach wieder schwarz.Dazwischen wird bei Demo=true eine Pause der Länge Pausenzeit eingelegt,und es ertönt ein kur-zer Ton.Bei Demo=false gibt es keine Pause,und es ertönt kein Ton.Die Kantenliste des Graphen wird im Label Ausgabe in der Property Caption gespeichert und der Liste Sliste vom Typ TStringlist hinzugefügt.Die Bewertung der Kanten erfolgt nach der Funktion Wert vom Typ TWert,und die verwendete Kanten-genauigkeit wird durch Kantengenauigkeit vorgegeben.

Procedure TInhaltsgraph.FaerbeGraph(F:TColor;T:TPenstyle)

Diese Methode setzt die Farbe und den Stil bei allen Knoten und Kanten eines Graphen auf die durch F und T vorgegebenen Werte.

Function TInhaltsgraph.FindezuKoordinatendenKnoten(var A,B:Integer;var Kno:TInhaltsknoten):Boolean

Diese Funktionsmethode testet,ob sich in der Nähe des Punktes der Zeichenfläche mit den Koordinaten A und B der Mittelpunkt eines Knoten befindet (Abstand ca. Knotenradius).Falls dies der Fall ist,wird der Knoten an den Referenzparameter Kno vom Typ TInhaltsknoten zurückgegeben.Der Punkt kann z.B. durch die Koordinaten eines Mausklicks vorgegeben werden.Die Knotenliste wird bei dieser Funktionsmethode vorwärts durchsucht,bis eine geeigneter Knoten gefunden ist,oder das Ende der Liste erreicht ist.

Function TInhaltsgraph.FindedenKnotenzuKoordinaten(var A,B:Integer;Var Kno:TInhaltsknoten):Boolean

Diese Funktionsmethode testet,ob sich in der Nähe des Punktes der Zeichenfläche mit den Koordinaten A und B der Mittelpunkt eines Knoten befindet (Abstand ca. Knotenradius).Falls dies der Fall ist,wird der Knoten an den Referenzparameter Kno vom Typ TInhaltsknoten zurückgegeben.Der Punkt kann z.B. durch die Koordinaten eines Mausklicks vorgegeben werden.Die Knotenliste wird bei dieser Funktionsmethode rückwärts durchsucht,bis eine geeigneter Knoten gefunden ist,oder der Anfang der Liste erreicht ist.

Function TInhaltsgraph.Graphknoten (Kno:TInhaltsknoten):TInhaltsknoten

Diese Funktionsmethode bestimmt zu einem Knoten Kno einen Zeiger auf einen Knoten im Graph,der die gleichen X-und Y-Datenfeldinhalte (Koordinaten auf der Zeichenfläche) wie der Knoten Kno hat und gibt diesen Zeiger zurück.Falls kein solcher Knoten existiert,wird nil zurückgegeben.

Procedure TInhaltsgraph.SpeichereGraph(Dateiname:string)

Diese Methode speichert den Graphen,der durch die Instanz von TInhaltsgraph oder (Objekt-) Nachfolgern von TInhaltsgraph vorgegeben ist,unter dem Namen Dateinamen,der auch den Pfad des (Speicher-)Verzeichnisses enthalten kann, auf Festplatte oder Diskette ab.Voraussetzung für die Benutzung dieser Methode für (Objekt-) Nachfolger von TInhaltsgraph ist,dass der Graph sowie dessen Kanten und Knoten jeweils die Methoden Wertlisteschreiben und Werlistelesen definieren,damit die Inhalte der

in vererbten Objekttypen definierten zusätzlichen Felder abgespeichert werden können.

Procedure TInhaltsgraph.LadeGraph(Dateiname:string)

Diese Methode lädt den Graph, der durch Dateiname (Dateiname enthält auch den Pfad des (Lade-) Verzeichnisses) vorgegeben ist,von Festplatte oder Diskette.Danach ist der Graph als der Instanz von TInhaltsgraph oder Nachfolgern von TInhaltsgraph verfügbar.Voraussetzung für die Benutzung dieser Methode für Nachfolger von TInhaltsgraph ist,daß der Graph sowie die Kanten und Knoten jeweils die Methoden Wertlisteschreiben und Werliste lesen definieren,damit die Inhalte der in vererbten Objekttypen definierten zusätzlichen Felder geladen werden können.

Procedure TInhaltsgraph.EingabeKante(var Ka:Tinhaltskante;var Aus:Boolean;var Abbruch:Boolean)

Diese Methode wird von der Funktionsmethode BeiMausklickKantezeichnen (X,Y:Integer):Boolean aufgerufen und regelt die Eingabe der Kantendaten.Sie ruft eine Eingabeform (Kantenform) in der Unit UKante auf,in der die Kantendaten eingegeben werden können.Die Kante Ka wird (mit Inhalt) als Referenzparameter Ka zurückgegeben.Der Referenzparameter Aus vom Datentyp Boolean gibt an,ob es sich bei einer gerichteten Kante um eine aus- oder einlaufende Kante handelt.(bezogen auf die Reihenfolge in der die Knoten in der Methode BeiMausklickKantezeichnen (X,Y:Integer;Flaeche:TCanvas):Boolean mit der Maus angewählt wurden.) Der Referenzparameter Abbruch vom Datentyp Boolean gibt an,ob das Verfahren der Eingabe der Kantendaten abgebrochen worden ist,und keine Kante erzeugt werden soll.
Diese Methode ist virtuell und kann für eigene Kantendatenmethoden überschrieben werden,die dann automatisch von der Methode BeiMausklickKantezeichnen (X,Y:Integer;Flaeche:TCanvas):Boolean aufgerufen werden.

Function TInhaltsgraph.Kantezeichnen(X,Y:Integer):Boolean

Das Erzeugen einer Kante zwischen zwei Knoten kann dadurch bewirkt werden,dass zuerst der erste Knoten und danach der zweite Knoten mit der linken Maus angeklickt werden.Schließlich wird die Methode EingabeKante gestartet,um das Eingeben der Kantendaten einzuleiten.Die Methode Kantezeichnen kann alle drei Vorgänge verarbeiten.Nachdem nach Mausklick die Bildschirmkoordinaten X und Y des Mausklickpunktes feststehen,wird die Functionsmethode zum ersten Mal mit diesen Koordinaten als Parameter aufgerufen.Zu den Koordinaten X und Y wird ein geigneter

Anfangsknoten gesucht,dessen Mittelpunkt in unmittelbarer Nähe des Mausklickpunktes liegen muß.Die Funktionsmethode liefert bei Auswahl eines Anfangsknoten als Resultat false zurück und erzeugt ein Ausgabefenster,dass der zweite Knoten (Endknoten) mit der Maus angewählt werden sollte.Nachdem die Koordinaten eines zweiten Mausklickpunkts feststehen,wird die Methode Kantezeichnen mit diesen Mausskoordinaten erneut aufgerufen und startet,falls ein weiterer Knoten als Endknoten bestimmt werden kann,der sich wiederum in der unmittelbarer Nähe dieses Mausklickpunktes befinden muß,daraufhin erneut die Methode EingabeKante.Nach Eingabe der Kantendaten und Beendigung der Methode EingabeKante (mit Abbruch=false) gibt die Funktionsmethode Kantezeichnen true zurück und fügt die Kante in den Graph ein.
Falls jeweils kein Knoten bestimmt werden konnte,ist der Rückgabewert der Funktionsmethode true,und es wird eine Fehlermeldung ausgegeben.
Der Gebrauch der Methode wird demonstriert in der Methode TKnotenform.KanteerzeugenMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer),die von der Methode TKnotenform.KantenerzeugenClick(Sender: TObject) (jeweils Unit UKnoten) aufgerufen wird.
Die ungewöhnliche Nutzung und Aufbau der Function (zweimaliger Aufruf) erweist sich als geeignetes und vorteilhaftes Mittel bei der ereignisorientierten Programmierung,wie sie in Delphi unter Windows üblich ist.

Function TInhaltsgraph.Inhaltskanteloeschen (X,Y:Integer):Boolean

Diese Funktionsmethode löscht eine Kante,die sich in der unmittelbaren Nähe des Zeichenflächenpunktes mit den Koordinaten X und Y befindet.Das Resultat der Funktion ist true,wenn die Kante gelöscht wurde,sonst false.Im letzten Fall wird die Meldung „Keine Kante“ ausgegeben.

Function TInhaltsgraph.Knoteninhaltzeigen(X,Y:Integer):Boolean

Die Werte X und Y sind die Koordinaten eines Zeichenflächenpunktes,der z.B. das Ergebnis eines Mausklicks ist.Wenn der Punkt in der Nähe des Mittelpunkts eines Knotens des Graphen liegt,werden dessen Koordinaten und dessen Inhalt,der durch die Propertys Wert und Wertposition festgelegt ist,in einem Anzeigefenster ausgegeben.Wenn ein Knoten des Graphen gefunden wurde,ist die Rückgabe der Funktionsmethode true sonst false.

Function TInhaltsgraph.Knotenverschieben(X,Y:Integer):Boolean

Die Werte X und Y sind die Koordinaten eines

Zeichenflächenpunktes,der z.B. das Ergebnis eines Mausklicks ist.Wenn der Punkt in der Nähe des Mittelpunkts eines Knotens liegt,wird dieser Knoten durch die Funktionsmethode ausgewählt,und der Mittelpunkt des Knoten auf den Punkt mit den Koordinaten X und Y auf der Zeichenfläche verschoben.Wenn ein Knoten ausgewählt werden konnte,ist das Resultat der Funktion true,sonst false.

Function TInhaltsgraph.Kanteverschieben(X,Y:Integer):Boolean

Die Werte X und Y sind die Koordinaten eines Zeichenflächenpunktes,der z.B. das Ergebnis eines Mausklicks ist.Wenn der Punkt in der Nähe des Mittelpunkts einer Kante liegt,wird diese Kante durch die Funktionsmethode ausgewählt und der Mittelpunkt der Kante senkrecht zur Verbindungslinie von Anfangs-und Endknoten der Kante auf den Punkt mit den Koordinaten X und Y hin verschoben.Wenn eine Kante ausgewählt werden konnte,ist das Resultat der Funktion true,sonst false.

Procedure TInhaltsgraph.EditiereKnoten(var Kno:TInhaltsknoten;var Abbruch:Boolean)

Diese Methode ist virtual und wird von der Function TInhaltsgraph.Knoteneditieren (X,Y:Integer) aufgerufen.Der Inhalt des Knoten Kno (bestimmt durch die Propertys Wert und Position) kann durch diese Methode verändert werden.Dazu wird ein Anzeigefenster geöffnet,in dem der alte Knoteninhalt durch einen neuen ersetzt werden kann.Der Werteparameter Abbruch bestimmt,ob der Knoteninhalt verändert werden soll (Abbruch=false) oder nicht (Abbruch=true).Die Methode kann durch eine Methode eines von TInhaltsgraph vererbten Objects überschrieben werden und so an die neuen Datenfelder eines erweiterten Knotenobjekts angepaßt werden.

Function TInhaltsgraph.Knoteneditieren(X,Y:Integer):Boolean

Die Werte X und Y sind die Koordinaten eines Zeichenflächenpunktes,der z.B. das Ergebnis eines Mausklicks ist.Wenn der Punkt in der Nähe des Mittelpunkts eines Knoten Kno liegt,wird dieser Knoten durch die Functionsmethode ausgewählt und die Methode EditiereKnoten(var Kno:TInhaltsknoten) aufgerufen.Dadurch kann der Knoteninhalt (festgelegt durch die Propertys Wert und Position) verändert werden.Das Ergebnis der Functionsmethode ist true,wenn ein Knoten ausgewählt wurde,sonst false.

Procedure TInhaltsgraph.EditiereKante(var Ka:TInhaltskante;var Aus:Boolean;var Abbruch:Boolean)

Diese Methode wird von der Function Kanteeditieren(X,Y:Integer;

Flaeche:TCanvas):Boolean aufgerufen und regelt das Verändern der Kantendaten.Sie ruft eine Eingabeform (Kantenform) in der Unit UKante auf,in der die Kantendaten verändert werden können.Die Kante Ka wird als Referenzparameter Ka zurückgegeben.Der Referenzparameter Aus vom Datentyp Boolean gibt an,ob es sich bei einer gerichteten Kante um eine aus- oder einlaufende Kante handelt (bezogen auf die Anfangs-und Endknoten der Kante).Der Referenzparameter Abbruch vom Datentyp Boolean gibt an,ob das Verfahren der Eingabe der Kantendaten abgebrochen worden ist,und keine Kante verändert werden soll.
Diese Methode ist virtuell und kann für eigene Kantendaten-methoden überschrieben werden,die dann automatisch von der Funk-tion BeiMausklickKanteeditieren(X,Y:Integer):Boolean aufgerufen werden.

Function TInhaltsgraph.Kanteeditieren(X,Y:Integer):Boolean

Die Werte X und Y sind die Koordinaten eines Zeichenflächenpunktes,der z.B. das Ergebnis eines Mausklicks ist. Wenn der Punkt in unmittelbarer Nähe einer Kante Ka liegt,wird diese Kante durch die Funktionsmethode ausgewählt und die Methode EditiereKante(var Ka:TInhaltskante;var Aus:Boolean;var Abbruch:Boolean) aufgerufen.Dadurch können die Kantendaten verändert werden.Flaeche vom Typ TCanvas ist die Objekt-Zeichenfläche,auf der der Knoten gezeichnet ist.Das Re-sultat der Funktionsmethode ist true,wenn eine Kante ausgewählt wurde,sonst false.

Function TInhaltsgraph.Kanteninhaltzeigen(X,Y:Integer):Boolean

Die Werte X und Y sind die Koordinaten eines Zeichenflächenpunktes,der z.B. das Ergebnis eines Mausklicks ist.Wenn der Punkt in unmittelbarer Nähe einer Kante liegt,werden deren Anfangs- und Endknoten und deren Inhalt,der durch die Propertys Wert und Position festgelegt ist,in einem Anzeigefenster ausgegeben.Das Resultat der Function ist true,wenn eine Kante ausgewählt wurde,sonst false.

Procedure TInhaltsgraph.EingabeKnoten(Var Kno:TInhaltsknoten; Var Abbruch:Boolean)

Diese Methode wird von der Methode Knotenzeichnen(X,Y:Integer): Boolean aufgerufen und regelt die Eingabe des Knoteninhalts (festgelegt durch die Propertys Wert und Position)).Sie zeigt ein Fenster an,in dem der bisherige Knoteninhalt angezeigt und durch einen neuen überschrieben werden kann.Der Knoten Kno wird als Referenzparameter Kno zurückgegeben.Der Referenzparameter Abbruch vom Datentyp Boolean gibt an,ob das Verfahren der Einga-

be der Knotendaten abgebrochen worden ist, und kein Knoten erzeugt werden soll.
Diese Methode ist virtuell und kann für die Eingabe von zusätzlichen Datenfelder bei vererbten Knotenobjekten überschrieben werden,die dann automatisch von der Methode Knotenzeichnen(X,Y:Integer):Boolean aufgerufen werden.

Function TInhaltsgraph.ZweiKnotenauswaehlen(X,Y:Integer; var Kno1,Kno2:TInhaltsKnoten;var Gefunden:Boolean):Boolean

Wenn diese Funktion zum erstemal aufgerufen wird,wird,wenn sich ein Zeichenflächenpunkt mit den Koordinaten X und Y in der Nähe des Mittelpunkts eines Knoten des Graphen befindet,dieser Knoten als erster Knoten ausgewählt.Falls ein erster Knoten ausgewählt wurde,gibt die Funktionsmethode als Resultat false zurück,sonst true.Bei einem zweiten Aufruf der Funktion mit den Koordinaten X und Y wird,falls sich wiederum der zugehörige Zeichenflächenpunkt in der Nähe des Mittelpunktes eines zweiten Knoten des Graphen befindet,dieser zweite Knoten ebenfalls ausgewählt.Falls ein zweiter Knoten gewählt wurde,gibt die Funktion bei Beendigung true zurück,sonst false.Ebenfalls true wird zurückgegeben wenn jeweils kein Knoten ausgewählt wurde.Der Referenzparameter Gefunden ist true,wenn zwei Knoten des Graphen bestimmt wurden,sonst false.
Die ungewöhnliche Nutzung und Aufbau der Function (zweimaliger Aufruf) erweist sich als geeignetes und vorteilhaftes Mittel bei der ereignisorientierten Programmierung,wie sie in Delphi unter Windows üblich ist.

Function TInhaltsgraph.Knotenzeichnen(X,Y:Integer):Boolean

Die Werte X und Y sind Koordinaten eines Zeichenflächenpunktes,der z.B. durch einen Mausklick bestimmt werden kann.An dieser Stelle wird durch die Funktionsmethode ein Knoten mit dem Punkt als Mittelpunkt gezeichnet. Außerdem wird die Methode EingabeKnoten(Var Kno:TInhaltsknoten;Var Abbruch:Boolean) aufgerufen,so daß ein Knoteninhalt (festgelegt durch die Propertys Wert und Position) eingegeben werden kann.Danach wird der Knoten in den Graph eingefügt.Das Resultat der Funktionsmethode ist true,wenn ein Knoten gezeichnet (d.h. in den Graph eingefügt) wurde,sonst false.

Function TInhaltsgraph.Inhaltsknotenloeschen (X,Y:Integer):Boolean

Diese Funktionsmethode löscht einen Knoten,die sich in der Nähe des Zeichenflächenpunktes mit den Koordinaten X und Y befindet.Falls kein Knoten gefunden wurde,wird die Meldung „Kein Knoten“ ausgegeben. Das Resultat der Funktion ist true,wenn ein Knoten gefunden und gelöscht (d.h. aus dem Graph entfernt)

wurde,sonst false.

Function TInhaltsgraph.InhaltsKopiedesGraphen
(Inhaltsgraphclass:TInhaltsgraphclass;
Inhaltsknotenclass:TInhaltsknotenclass;Inhaltskanteclass:
TInhaltskanteclass;UngerichteterGraph:Boolean):TInhaltsgraph

Diese Funktionsmethode erzeugt eine Inhaltskopie des Graphen,d.h. es wird ein neuer Graph erzeugt,dessen Struktur gleich der Struktur des vorgegebenen Graphen ist,und dessen Knoten und Kanten dieselben Inhalte wie die Knoten und Kanten im vorgegebenen Graphen enthalten.(aber nicht dieselben Knoten und Kanten sind!)
Außerdem ist mittels dieser Funktionsmethode das Erzeugen eines neuen Graphen möglich,dessen Knoten-,Kanten- und dessen Graph-Objekttyp sich jeweils von den Knoten-,Kanten-und Graphtypen des Ursprungsgraph durch Vererbung ableiten oder umgekehrt.

Die Parameter bedeuten:

Inhaltsgraph vom Typ TInhaltsgraphclass ist der Objektyp in der Kopie des Graphen.
Inhaltsknotenclass vom Typ TInhaltsknotenclass ist der Objekttyp der Knoten in der Kopie des Graphen.
Inhaltskanteclass vom Typ TInhaltskanteclass ist der Objekttyp der Kanten in der Kopie des Graphen.
Der Benutzer ist selber dafür verantwortlich,daß die Objekttypen jeweils in einer Vererbungsrelation zu den ursprüngliche Typen stehen.Dies wird von der Funktionsmethode nicht überprüft.
Ungerichtet=true bedeutet,dass die Kopie ungerichtete Kanten erhält.
Die Funktionsmethode gibt die Kopie des Graphen zurück.

Function TInhaltsgraph.AnzahlTypknoten(Typ:char):Integer

Diese Funktionsmethode gibt die Anzahl der Knoten,die vom Datentyp Typ sind, zurück.
Function TInhaltsgraph.AnzahlTypkanten(Typ:char):Integer

Diese Funktionsmethode gibt die Anzahl der Kanten,die vom Datentyp Typ sind,zurück.

Function TInhaltsgraph.AnzahlBruecken(var Sliste:TStringlist,
Ausgabe:Tlabel;Flaeche:TCanvas):Integer

Diese Funktionsmethode gibt die Anzahl der Brücken des Graphen zurück und speichert die Kanten jeweils in der Form

„Anfangsknoten.Wert - Endknoten.Wert“ (Inhalt wird durch die Propertys Wert und Position festgelegt) in dem Referenzparameter Sliste als Strings einer Liste vom Typ TStringlist.Im Label Ausgabe vom Typ TLabel wird die gerade untersuchte Kante angezeigt.Die Kanten werden im Ausgabefenster als Kanten ausgegeben und im Graphen markiert angezeigt.
Wenn sich beim Löschen einer Kante in einem Graph die Anzahl der Komponenten vergrößert,heißt die Kante Brücke.

Function TInhaltsgraph.AlleKnotenbestimmen:TStringlist

Diese Funktionsmethode speichert die Knoten-Wert(e) (bestimmt durch die Propertys Wert und Position) als Strings einer Liste vom Typ TStringlist und gibt die Liste zurück.

Function TInhaltsgraph.AlleKantenbestimmen:TStringlist

Diese Funktionsmethode speichert die Kanten-Werte(e) (bestimmt durch die Propertys Wert und Position) als Strings einer Liste vom Typ TStringlist und gibt die Liste zurück.

Function TInhaltsgraph.AnzahlKnotenkleinsterKreis(var St:string; Flaeche:TCanvas):Integer

Diese Funktionsmethode bestimmt einen Kreis mit kleinster Kantenzahl,falls er existiert,und gibt die Kantenzahl als Resultat zurück.Der Referenzparameter St vom Datentyp string enthält dann die Kanten-Wert(e) jeweils durch Leerzeichen getrennt.Falls ein Kreis nicht existiert,wird als Kantenzahl -1 und für St der Leerstring zurückgegeben.
Der Kreis wird markiert auf der Objektfläche Fläche gezeichnet.

Function TInhaltsgraph.AnzahlKnotengroesterKreis(var St:string; Flaeche:TCanvas):Integer

Diese Funktionsmethode bestimmt einen Kreis mit größter Kantenzahl,falls er existiert und gibt die Kantenzahl als Resultat zurück.Der Referenzparameter St vom Datentyp string enthält dann die Kanten-Wert(e) jeweils durch Leerzeichen getrennt.Falls ein Kreis nicht existiert,wird als Kantenzahl -1 und für St der Leerstring zurückgegeben.
Der Kreis wird markiert auf der Objektfläche Fläche gezeichnet.

Function TInhaltsgraph.KreisefesterLaenge(Laenge:Integer;var Sliste:TStringlist;Flaeche:TCanvas; Ausgabe:TLabel):Integer

Diese Funktionsmethode bestimmt alle Kreise mit der festen Länge Laenge im Graphen und gibt ihre Anzahl zurück.Die Kreise werden als Knotenfolge in der Liste Sliste gespeichert und im Label

Ausgabe angezeigt sowie im Demomodus auf der Objektfläche Flaeche markiert mit Verzügerung ausgegeben.

Proceduren und Functionen,die von der Unit UInhGrph exportiert werden:

Function Bewertung (Ob:TObject):Extended

Diese Function erzeugt eine Bewertung für eine Objekt Ob vom Typ TObject (oder Nachfolgern von TObject).Wenn Ob vom Typ TInhaltskante und außerdem der Typ der Kante ‘i’ (Integer) oder ‘r’ (Real) ist,ist die Bewertung der vom Datentyp String in eine Realzahl konvertierte Zahlenwert (vom Typ Extended) der Property Wert der Kante (festgelegt durch die Property Position).Beim Typ ‘s’ (string) ist die Bewertung 1.Wenn Ob vom Typ TInhaltsknoten ist,ist die Bewertung der vom Datentyp String in eine Realzahl konvertierte Zahlenwert (vom Typ Extended) der Property Wert des Knotens (festgelegt durch die Property Position).In allen anderen Fällen ist die Bewertung 0.

Function ErzeugeKnotenstring(Ob:TObject):string

Diese Function faßt Ob als vom Datentyp TInhaltsknoten auf (Typkonversion) und gibt den Wert der Property Wert des Knotens als String (festgelegt durch die Property Position) zurück.

Function ErzeugeKantenstring(Ob:TObject):string

Diese Function faßt Ob als vom Datentyp Tinhaltskante auf (Typkonversion) und gibt den Wert der Property Wert der Kante als String (festgelegt durch die Property Position) zurück.

Procedure LoescheBild(var G:TInhaltsgraph;var Oberflaeche:TForm)

Diese Procedure löscht das in der Objekt-Form vom Typ TForm gezeichnete Bild des Graphen (und nicht den Graphen selber).Dabei ist G der auf der Zeichenoberfläche gezeichnete Graph oder steht zu diesem in einer Vererbungsrelation.

Beschreibung der Unit UAusgabe:

Diese Unit deklariert eine Form des Objekttyps TAusgabeform,der sich von TForm durch Vererbung ableitet.
Durch die in dieser Unit globale Variable Ausgabeform wird ein Ausgabefenster definiert,das zur Ausgabe von Ergebnissen dienen kann. Der Typ TAusgabeform hat folgende Datenfelder:

Gitternetz: TStringGrid;

```
MainMenu: TMainMenu;
Ende: TMenuItem;
Kopieren: TMenuItem;
Drucken: TMenuItem;
PrintDialog: TPrintDialog;
private
Listenbox_:TListbox;
```

Gitternetz vom Typ TStringGrid ist ein Gitternetz,das aus einer Spalte und Zeilen besteht,in die mittels des Datenfeldes Listenbox_ vom Typ TListbox der Inhalt der Elemente von Listenbox_ in die Zeilen als Strings ausgegeben werden kann. MainMenu vom Typ TMainMenu ist das Hauptmenü der Form,das wiederum die Untermenüs Ende,durch das die Form geschlossen werden kann,Kopieren,durch das der Text der Listenbox in die Zwischenablage kopiert wird und Drucken,durch das der Text gedruckt werden kann (jeweils vom Datentyp TMenuItem) umfaßt.PrintDialog vom Typ TPrintDialog ist das Delphi-Auswahlfenster für Drucker.

Die Form kann durch die Anweisungen Ausgabeform:=TAusgabeform.Create(self) erzeugt und durch Ausgabeform.Showmodal initiert und dargestellt werden.

Darüberhinaus enthält der Datentyp eine Property Listenbox,um über sie auf das Feld Listenbox_ zuzugreifen.

Property:

```
Property Listenbox:TListbox read WelcheListbox write
SetzeListbox
```

Durch die folgenden Methoden von TAusgabeform (Proceduren und Functionen) wird auf die Property Listenbox lesend und schreibend zugegriffen:

```
Function TAusgabeform.WelcheListbox:TListbox
Procedure TAusgabeform.SetzeListbox(Lb:TListbox)
```

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

```
Procedure TAusgabeform.EndeClick(Sender:TObject)
```

Diese (Ereignis-)Methode schließt und beendet Instanzen von TAusgabeform bei Mausklick auf das Menü Ende.In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber,welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

```
Procedure TAusgabeform.KopierenClick(Sender:TObject)
```

Diese (Ereignis-)Methode kopiert den Text der Listenbox (d.h. maximal die ersten 255 Zeilen) bei Mausklick auf das Menü Kopieren.In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber, welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Procedure TAusgabeform.DruckenClick(Sender:TObject)

Diese (Ereignis-)Methode druckt den Text der Listenbox (d.h. maximal die ersten 255 Zeilen) bei Mausklick auf das Menü Drucken.Zuvor wird ein Druckerauswahlfenster gezeigt,in dem ein Drucker ausgewählt werden kann.In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber, welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Procedure TAusgabeform.FormPaint(Sender: TObject)

Diese (Ereignis-)Methode füllt die Zeilen des Gitternetzes Gitternetz vom Typ TStringGrid mit den Elemente der Listenbox vom Typ TListbox und stellt das Gitternetz in der Form Ausgabeform dar.Wenn Listenbox keine Element enthält,wird in dem Gitternetz die Ausgabe „leere Ausgabe“ erzeugt.In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber, welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Constructor TAusgabeform.Create(AOwner:TComponent)

Diese Methode ist der Constructor von TAusgabeform.

Beschreibung der Unit UKante:

Die Unit UKante definiert eine Form des Objekttyps TKantenform,die sich von TForm durch Vererbung ableitet.Durch die in dieser Unit globale Variable Kantenform vom Typ TKantenform wird ein Eingabe- und Editierfenster für Kantendaten definiert.Der Typ TAusgabeform hat folgende Datenfelder:

```
Checkausgehend: TCheckBox;
Checkeingehend: TCheckBox;
CheckInteger: TCheckBox;
CheckReal: TCheckBox;
Inhalt: TEdit;
Weite: TEdit;
ScrollBar: TScrollBar;
OK: TButton;
Abbruch: TButton;
Textausgehend: TLabel;
```

```
TextEingehend: TLabel;
TextInteger: TLabel;
TextReal: TLabel;
TextWeite: TLabel;
TextInhalt: TLabel;
private
Kanteein_,Kanteaus_:Boolean;
KantenInteger_,Kantenreal_:Boolean;
Kanteninhalt_:string;
Kantenweite_:Integer;
Kantenabbruch_:Boolean;
```

Kantenform enthält vier Checkboxen Checkausgehend,Checkeingehend, CheckInteger und CheckReal vom Typ TCheckbox zur Eingabe der Eigenschaften ausgehend(e Kante),eingehend(e Kante),Integer(Typ-Kante) und Real(Typ-Kante).Ausgehend und eingehend beschreiben die Richtung von gerichteten Kanten.Werden beide Eigenschaften angewählt oder keine,so wird eine ungerichtete Kante erzeugt.
Die Felder Inhalt und Weite vom Typ TEdit sind Eingabefelder für den Inhalt und die Weite der Kante (Abstand des Kantenmittelpunkts von der Verbindungslinie der beiden Endknoten).ScrollBar vom Typ TScrollBar ist ein Schiebregler zur Einstellung der Weite.
Die Buttons OK und Abbruch vom Typ TButton dienen zu Übernahme bzw. Nicht-Übernahme der Daten und Verlassen der Form.Die Felder Textausgehend, Texteingehend, TextInteger,TextReal,TextWeite und TextInhalt vom Datentyp TLabel stellen die (Erläuterungs-)Texte in der Form dar.
Die Datenfelder Kanteein_, Kanteaus_, KantenInteger_, Kantenreal_, Kanteninhalt_, Kantenweite_ dienen zum Zwischenspeichern der in der Kantenform in die Felder Checkausgehend, Checkeingehend, CheckInteger, CheckReal, Inhalt,Weite ein- bzw. ausgegebenen Werte.Kantenabbruch_ beschreibt,ob die Eingabe in die Kantenform mittels des Buttons Abbruch abgebrochen wurde.

Die Form kann durch die Anweisungen Kantenform:=TKantenform.Create(self) erzeugt und durch Kantenform.Showmodal initiert und dargestellt werden.

Darüberhinaus enthält der Datentyp eine Reihe von Propertys,um über sie auf die oben genannten (private) deklarierten Datenfelder zuzugreifen.

Property Kanteein:Boolean read WelchesKantenein write setzeKantenein
Property Kanteaus:Boolean read WelchesKantenaus write setzeKantenaus
Property KantenInteger:Boolean read WelchesKanteninteger write setzeKantenInteger

Property Kantenreal:Boolean read WelchesKantenreal write setzeKantenreal
Property Kanteninhalt:string read WelcherKanteninhalt write setzeKanteninhalt
Property Kantenweite:Integer read WelcheKantenweite write setzeKantenweite
Property Kantenabbruch:Boolean read WelcherKantenabbruch write setzeKantenabbruch

Durch die folgenden Methoden von TKante (Proceduren und Functionen) wird auf die Propertys lesend und schreibend zugegriffen:

Function TKante.WelchesKantenein:Boolean
Procedure TKante.SetzeKantenein(Ke:Boolean)
Function TKante.WelchesKantenaus:Boolean
Procedure TKante.SetzeKantenaus(Ka:Boolean)
Function TKante.WelchesKantenInteger:Boolean
Procedure TKante.SetzeKantenInteger(Ki:Boolean)
Function TKante.WelchesKantenreal:Boolean
Procedure TKante.SetzeKantenreal(Kr:Boolean)
Function TKante.WelcherKanteninhalt:string
Procedure TKante.SetzeKanteninhalt(Ki:string)
Function TKante.WelcheKantenweite:Integer
Procedure TKante.SetzeKantenweite(Kw:Integer)
Function TKante.WelcherKantenabbruch:Boolean
Procedure TKante.SetzeKantenabbruch(Ab:Boolean)

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Procedure TKantenform.OKClick(Sender: TObject)

Diese (Ereignis-)Methode wird aufgerufen,wenn der OK-Button mit der linken Maustaste angewählt wird.Sie speichert die vom Benutzer in die entsprechenden (oben beschriebenden) Datenfelder der Form eingegebenen Daten in den Datenfeldern Kanteein_, Kanteaus_, KantenInteger_, Kantenreal_, Kanteninhalt_, Kantenweite_ und beendet die Kantendateneingabe durch Schließen des Form-Fensters.Kantenabbruch_ wird auf false gesetzt.Die Zulässigkeit und Konvertierbarkeit von numerischen Eingaben als Strings im Feld Inhalt bzw. Kanteninhalt_ in Zahlen wird überprüft und gegebenfalls eine Fehlermeldung ausgegeben.In diesem Fall wird die Eingabe abgebrochen und das Eingabeformfenster geschlossen. In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber, welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Procedure TKantenform.InhaltChange(Sender:TObject)

Diese (Ereignis-)Methode reagiert auf Änderungen des Inhalts-Eingabefeldes Inhalt und speichern die neue Eingabe in dem Datenfeld Kanteninhalt_.
In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber, welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Procedure TKantenform.CheckIntegerClick(Sender:TObject)

Diese (Ereignis-)Methode setzt die Eigenschaft Integer der Kante (CheckInteger.checked) auf true und die Eigenschaft Real der Kante (CheckReal.checked) auf false.Nur eine der Eigenschaften Integer oder Real kann angewählt werden.
In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber,welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Procedure TKantenform.CheckRealClick(Sender:TObject)

Diese (Ereignis-)Methode setzt die Eignschaft Real (CheckReal.checked) auf true und die Eigenschaft Integer der Kante (CheckInteger.checked) auf false.Eine der Eigenschaften Real oder Integer kann nur angewählt werden.
In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber, welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Procedure TKantenform.InhaltKeyPress(Sender:TObject;var Key: Char)

Diese (Ereignis-)Methode reagiert auf das Drücken der Eingabe-Taste im Eingabefeld für den Kanteninhalt Inhalt und bewirkt dasselbe wie die Methode OKClick. Der Referenzparameter Key übergibt die gewählten Tasten.
In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber,welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Procedure TKantenform.FormActivate(Sender: TObject)

Diese (Ereignis-)Methode speichert die Werte der oben genannten Datenfelder Kanteein_, Kanteaus_, KantenInteger_, Kantenreal_, Kanteninhalt_, Kantenweite_ in den entsprechenden Datenfeldern der Instanz Kantenform von TKantenform zur dortigen Anzeige.Sie wird beim Aktivieren der Form aufgerufen.
In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber, welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Procedure TKantenform.AbbruchClick(Sender:TObject)

Diese (Ereignis-)Methode wird aufgerufen,wenn der Button Abbruch mit der linken Maustaste angeklickt wird und schließt und beendet die Instanz von TKantenform.Vorher wird Kantenabbruch_auf true gesetzt.
In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber, welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Procedure TKantenform.WeiteKeyPress(Sender: TObject; var Key: Char)

Diese Ereignismethode sclieẞt die Kantenform,wenn Return gedrückt wird und wenn das Weite-Eingabefeld den Focus hat.
In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber, welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Procedure TKantenform.ScrollBarChange(Sender:TObject)

Diese Methode überträgt die Position der Scrollbar an das Eingabefeld Weite und stellt Position dort als string dar.
In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber,welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Procedure TKantenform.WeiteChange(Sender:TObject)

Diese Methode überträgt den Inhalt des Eingabefeldes Weite an die Scrollbar und setzt entsprechend deren Position.
In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber, welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Constructor TKantenform.Create(AOwner: Tcomponent)

Diese Methode ist der Constructor von TKantenform.

Beschreibung der Unit UPfad:

Die Unit UPfad definiert die Objekt-Datentypen TPfadknoten und TPfadgraph, die sich durch Vererbung von TInhaltsknoten und TInhaltsgraph ableiten,sowie die zugehörigen Methoden,die bei den Anwendungsalgorithmen des Menüs Pfade benötigt werden.Das Menü Pfade enthält die folgenden Untermenüs:
Alle Pfade (von einem Knoten aus),(alle) Kreise (von einem Knoten aus),(alle) Minimale Pfade (von einem Knoten aus), Anzahl (der erreichbaren) Zielknoten (von einem Knoten aus),Tiefe(r)

Baum(pfade) (von einem Knoten aus),Weite(r) Baum(pfade) (von einem Knoten aus),(kürzester Pfad-) Abstand von zwei Knoten,Alle Pfade zwischen zwei Knoten,Minimales Gerüst des Graphen.

Die folgende Beschreibung ist sinnvollerweise nach den einzelnen Anwendungen gegliedert und enthält jeweils die von den einzelnen Anwendungen verwendeten Methoden von TPfadknoten und TPfadgraph in unmittelbarer Reihenfolge zusammen.Diese Einheiten stellen ein vollständiges Programmierungsprojekt dar,wenn noch die Menümethode aus der Unit Knoten zum Aufruf der entsprechenden TPfadgraphmethode hinzugefügt wird.Die Methoden im zweiten Teil werden nicht durch ein Menü des Programms Knotengraph direkt oder indirekt aufgerufen (Kennzeichnung ohne Menü) oder sonstwie verwendet und sind didaktisch-methodische Vereinfachung bzw. Zusätze der Methoden des ersten Teils,die den Quellcode der entsprechenden Anwendungen des Menüs Pfade darstellen.Sie werden hier nur aus didaktischen Gründen zum Vergleich bzw. um Alternativen darzustellen aufgeführt.Ihre Funktionalität ist im Programm Knotengraph schon durch den Einsatz der erstgenannten Methoden enthalten.
Eine Reihe der Methoden von TPfadknoten stehen auch schon (fast Quellcodegleich) als Methoden von TKnoten bzw. TGraph in der Unit UGraph für Anwendungen zur Verfügung und sind hier aus Gründen der Übersichtlichkeit nochmal vorhanden,um in Bezug auf Programmierungsprojekte alle Bestandteile einer geschlossenen Unterrichtseinheit zusammenzustellen.Die Methoden wurden in der Unit UKnoten den Objekttypen TKnoten und TGraph deswegen noch einmal extra hinzugefügt,um sie für die Erstellung auf Ihnen aufbauender Algorithmen verwenden zu können,falls man nicht die Unit UPfad zur Verfügung hat (wie in der Entwicklungsumgebung von Knotengraph EWK) oder Methoden dieser Unit nicht extra erstellen möchte.Z.B. werden durch sie sämtliche Pfade der Pfadalgorithmen-Methoden als Pfade der Pfadlisten der Endknoten dieser Pfade gespeichert und damit für mögliche andere darauf aufbauende Anwendungen fertig bereitgestellt.Da die Unit UGraph in der Entwicklungsumgebung EWK für neu zu beginnende Schülerprojekte nur in compilierter Form vorliegt,sind diese Methoden,falls es darum geht Teile der Unit Pfade als Unterrichtsprojekt zu programmieren,(ohne eine entsprechende Mitteilung des Lehrers) als Methoden von TKnoten und TGraph für Schüler verborgen,so dass kein Motivationsverlust bei der nochmaligen Programmierung einer schon vorhandenen funktionsgleichen Methode auftritt.

Die Datentypen TPfadknoten und TPfadgraph enthalten keine neuen Felder und auch keine Propertys.

Die Bewertung der Kanten geschieht in den Methoden dieser Unit durch die Funktionsmethode **Bewertung** vom Typ TWert aus der Unit UInhGrph.

Ein Punkt liegt in der (unmittelbaren) Nähe eines Knotens bedeutet im folgenden,dass der Punkt maximal um den Radius des Knotenkreises vom Mittelpunkt dieses Kreises entfernt ist.

Procedure TPfadknoten.Free

Diese Methode ist der einfache Destructor von TPfadknoten und entfernt Instanzen des Datentyps (ausschließlich der in der ausgehenden und eingehenden Kantenliste gespeicherten Kanten) aus dem Speicher.

Procedure TPfadknoten.Freeall

Diese Methode ist der erweiterte Destructor entfernt Instanzen von TPfadknoten einschließlich der in der ausgehenden und eingehenden Kantenliste gespeicherten Kanten aus dem Speicher.

Procedure TPfadgraph.FreeAll

Diese Methode ist der einfache Destructor von TPfadgraph entfernt Instanzen des Datentyps (ausschließlich der in der Knotenliste gespeicherten Knoten und der in der Kantenliste gespeicherten Kanten) aus dem Speicher.

Procedure TPfadgraph.FreeAll

Diese Methode ist der erweiterte Destructor entfernt Instanzen von TPfadgraph einschließlich der in der Knotenliste gespeicherten Knoten und der in der Kantenliste gespeicherten Kanten aus dem Speicher

Anwendung: AllePfade

Procedure TPfadknoten.ErzeugeallePfade

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen Pfade zu allen anderen Knoten des Graphen,falls existent.Alle Pfade werden in der Pfadliste des Startknoten gespeichert.Falls keine Pfade existieren,sind die entsprechenden Pfade leer (die Pfadliste enthält keinen Eintrag).

Procedure TPfadknoten.BinaererBaumInorder

Diese Methode durchläuft einen Graphen,der als Binärbaum aufgefaßt werden kann,in Inorder-Reihenfolge.Dabei ist die zuerst in die ausgehende Kantenliste eines Knotens eingefügte Kante die Kante zum linken Teilbaum und die danach eingefügte Kante die Kante zum rechten Teilbaum.Enthält ein Knoten nur eine ausgehende Kante ist dies automatisch die Kante zum linken Teilbaum.Soll

ein Knoten nur eine Kante zum rechten Teilbaum besitzen,muß als erste Kante eine Kante zu einem Dummy-Knoten mit leerem Inhalt eingefügt werden.Knoten von denen keine Kanten ausgehen,sind die Blätter.Der Pfadknoten ist als Wurzel des Binär(teil)baums der Ausgangspunkt des Inorderdurchlaufs.

Procedure TPfadgraph.AllePfadevoneinemKnotenbestimmen (X,Y:Integer;Ausgabe:TLabel; var Sliste:TStringlist; Flaeche:TCanvas)

Diese Methode bestimmt zunächst zu einem Zeichenflächenpunkt mit den Koordinaten X und Y einen Knoten des Graphen,dessen Mittelpunkt in der Nähe des Punktes liegt.Wenn kein solcher Knoten existiert,wird der Knoten gewählt,der als erstes Element in der Knotenliste des Graphen gespeichert ist.Anschließend werden alle möglichen Pfade des Graphen erzeugt,die den ausgewählten Knoten als Startknoten haben.Die Pfade werden als Liste von Strings,wobei jeder String jeweils die Knotenfolge des Pfads,die Kantensumme und das Kantenprodukt enthält, in dem Referenzparameter SListe vom Typ TStringlist gespeichert und außerdem in dem Label Ausgabe vom Typ TLabel jeweils beim Zeichnen der Pfade (Zeichnen=true) ausgegeben.Flaeche vom Typ TCanvas gibt die Objekt-Zeichenfläche an,auf der die Pfade (markiert) gezeichnet werden.Falls der Graph als Binärbaum aufgefaßt werden kann,kann die Option Inorder-Durchlauf vom Startknoten aus gewählt werden und die Methode BinaererBaumInorder(Beschreibung:siehe dort) wird aufgerufen.Die Pfade werden im Demo-Modus markiert auf der Zeichenoberfläche gezeichnet und als Knotenfolge des Pfades im Ausgabefenster angezeigt.

Anwendung: Kreise

Procedure TPfadknoten.ErzeugeKreise

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen Kreise zu allen anderen Knoten des Graphen,falls existent.Alle Pfade werden in der Pfadliste des Startknoten gespeichert.Falls keine Pfade existieren ist die Pfadliste leer (die Pfadliste enthält keinen Eintrag).

Procedure TPfadgraph.AlleKreisevoneinemKnotenbestimmen (X,Y:Integer; Ausgabe:TLabel;var Sliste:TStringlist;Flaeche:TCanvas)

Diese Methode bestimmt zunächst zu einem Zeichenflächenpunkt mit den Koordinaten X und Y einen Knoten des Graphen,dessen Mittelpunkt in der Nähe des Punktes liegt.Wenn kein solcher Knoten existiert,wird der Knoten gewählt,der als erstes Element in der Knotenliste des Graphen gespeichert ist.Anschließend werden alle möglichen Kreise des Graphen erzeugt,die den ausgewählten Knoten als Startknoten haben.Die Pfade werden als Liste von

Strings,wobei jeder String jeweils die Knotenfolge des Pfads,die Kantensumme und das Kantenprodukt enthält,in dem Referenz-parameter SListe vom Typ TStringlist gespeichert und außerdem in dem Label Ausgabe vom Typ TLabel jeweils beim Zeichnen ausgegeben.Flaeche vom Typ TCanvas gibt die Objekt-Zeichenfläche an,auf der die Pfade (markiert) gezeichnet werden.

Anwendung: Minimale Pfade

Procedure TPfadknoten.ErzeugeminimalePfadenachDijkstra (Flaeche:TCanvas)

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen minimalen Pfade bezüglich der Funktion Bewertung (Bewertung der Kanten bzw. Pfade) vom Typ TWert zu allen anderen Knoten des Graphen,falls existent.Alle Pfade werden in der Pfadliste des Startknoten gespeichert.Falls keine Pfade existieren sind die entsprechenden Pfade leer (die Pfadliste enthält keinen Eintrag).Die Methode arbeitet nach dem Algorithmus von Dijkstra.Flaeche vom Typ TCanvas ist die Objektzeichenfläche,auf der die Pfade im Demomodus gezeichnet werden.(TWert=function(Ob:TObject):Extended)

Procedure TPfadgraph.AlleminimalenPfadevoneinenKnotenbestimmen (X,Y:Integer;Ausgabe:TLabel;var Sliste:TStringlist;Flaeche:TCanvas)

Diese Methode bestimmt zunächst zu einem Zeichenflächenpunkt mit den Koordinaten X und Y einen Knoten des Graphen,dessen Mittelpunkt in der Nähe des Punktes liegt.Wenn kein solcher Knoten existiert,wird der Knoten gewählt,der als erstes Element in der Knotenliste des Graphen gespeichert ist.Anschließend werden jeweils alle minimalen Pfade (falls existent) zu den anderen Knoten des Graphen erzeugt,die den ausgewählten Knoten als Startknoten haben.Die Pfade werden als Liste von Strings,wobei jeder String jeweils die Knotenfolge des Pfads,die Kantensumme und das Kantenprodukt enthält,in dem Referenzparameter Sliste vom Typ TStringlist gespeichert und außerdem in dem Label Ausgabe vom Typ TLabel jeweils beim Zeichnen ausgegeben.Flaeche vom Typ TCanvas gibt die Objekt-Zeichenfläche an,auf der der die Pfade (markiert) gezeichnet werden.Bewertung vom Typ TWert bestimmt die Kanten- bzw. Pfadbewertung und bestimmt die Minimalität der Pfade.
(TWert=function(Ob:TObject):Extended)

Anwendung:Anzahl Zielknoten

Diese Anwendung benötigt außerdem die Methode:

ProcedureTPfadknoten.ErzeugeminimalePfadenachDijkstra

der Anwendung Minimale Pfade.

Function TPfadknoten.AnzahlPfadZielknoten:Integer

Diese Funktionsmethode gibt die Anzahl der Knoten im Graph zurück,zu dem vom Knoten als Startknoten aus Pfade existieren.Der Startknoten wird dabei nicht mitgezählt.

Anwendung :Tiefer Baum

Procedure TPfadnoten.ErzeugeTiefeBaumPfade(Preorder:Boolean)

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen tiefen Baumpfade zu allen anderen Knoten des Graphen,falls existent.Alle Pfade werden in der Pfadliste des Startknoten gespeichert.Falls keine Pfade existieren,sind die entsprechenden Pfade leer (die Pfadliste enthält keinen Eintrag).Der Parameter Preorder bestimmt (Preorder=true),dass der Baum gemäß der Ordnung Preorder durchlaufen wird,ansonsten wird die Ordnung Postorder benutzt.

Procedure TPfadgraph.AlletiefenBaumpfadevoneinemKnotenbestimmen (X,Y:Integer;;Ausgabe:TLabel;var Sliste:TStringlist;Flaeche:TCanvas)

Diese Methode bestimmt zunächst zu einem Zeichenflächenpunkt mit den Koordinaten X und Y einen Knoten des Graphen,dessen Mittelpunkt in der Nähe des Punktes liegt.Wenn kein solcher Knoten existiert,wird der Knoten gewählt,der als erstes Element in der Knotenliste des Graphen gespeichert ist.Anschließend werden alle möglichen tiefen Baumpfade des Graphen erzeugt,die den ausgewählten Knoten als Startknoten haben.Die Pfade werden als Liste von Strings,wobei jeder String jeweils die Knotenfolge des Pfads,die Kantensumme und das Kantenprodukt enthält,in dem Referenzparameter Sliste vom Typ TStringlist gespeichert und außerdem auf dem Label Ausgabe vom Typ TLabel jeweils beim Zeichnen ausgegeben.Flaeche vom Typ TCanvas gibt die Objekt-Zeichenfläche an,auf der der die Pfade (markiert) gezeichnet werden.

Anwendung: Weiter Baum

Procedure TPfadnoten.ErzeugeWeiteBaumPfade

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen weiten Baumpfade zu allen anderen Knoten des Graphen,falls existent.Alle Pfade werden in der Pfadliste des Startknoten gespeichert.Außerdem enthält die Pfadliste aller anderen Knoten

außer dem Startknoten den Pfad vom Startknoten zu diesem Knoten.Falls keine Pfade existieren sind die entsprechenden Pfade leer (die Pfadliste enthält keinen Eintrag).

Procedure TPfadgraph.AlleweitenBaumpfadevoneinemKnotenbestimmen (X,Y:Integer;Ausgabe:Tlabel;Var Sliste: TStringlist;Flaeche:TCanvas)

Diese Methode bestimmt zunächst zu einem Zeichenflächenpunkt mit den Koordinaten X und Y einen Knoten des Graphen,dessen Mittelpunkt in der Nähe des Punktes liegt.Wenn kein solcher Knoten existiert,wird der Knoten gewählt,der als erstes Element in der Knotenliste des Graphen gespeichert ist.Anschließend werden alle möglichen weiten Baumpfade des Graphen erzeugt,die den ausgewählten Knoten als Startknoten haben.Die Pfade werden als Liste von Strings,wobei jeder String jeweils die Knotenfolge des Pfads,die Kantensumme und das Kantenprodukt enthält,in dem Referenzparameter Sliste vom Typ TStringlist gespeichert und außerdem auf dem Label Ausgabe vom Typ TLabel jeweils beim Zeichnen (Zeichnen=true) ausgegeben.Flaeche vom Typ TCanvas gibt die Objekt-Zeichenfläche an,auf der die Pfade (markiert) gezeichnet werden.

Anwendung: Abstand von zwei Knoten

Function TPfadgraph.BestimmeminimalenPfad (Kno1,Kno2:TKnoten;Flaeche:TCanvas):TPfad

Diese Funktionmethode bestimmt den minimalen Pfad zwischen den Knoten Kno1 und Kno2 vom Datentyp TKnoten gemäß dem Algorithmus von Djkstra.Der Pfad wird als Graph,dessen Kantenliste der gesuchte minimale Pfad ist,vom Typ TGraph und gleichzeitig TPfad zurückgegeben.Als Bewertung der Pfadlängen dient dabei die Funktion Bewertung vom Typ TWert.Flaeche vom Typ TCanvas ist die Objektzeichenfläche,auf der die Pfade im Demodus gezeichnet werden. (TWert=function(Ob:TObject):Extended)

Procedure TPfadKnoten.BinaeresSuchen(Kno:TKnoten)

Falls der Graph als Binärbaum aufgefasst werden kann,wird von dem Pfadknoten als Wurzel eines Teilbaums aus nach einem Knoten binär gesucht,dessen Wert gleich dem des Knotens Kno ist.Dabei wird ein bezüglich der Knotenwerte vom Datentyp string Inordergeordneter Binärbaum vorrausgesetzt,wobei die Methode selber bestimmt,ob sich im linken oder rechten Teilbaum das kleinere Element befindet.Der Knoten Kno kann ein isolierter Knoten ausserhalb des Binärbaums sein.Falls sich kein Knoten mit gleichem Wert im Binärbaum befindet,ist die Pafdliste des Pfadknotens leer,ansonsten enthält sie den Pfad von der Wurzel zum

gesuchten Knoten.

Function TPfadgraph.MinimalenPfadzwischenzweiKnotenbestimmen (X,Y:Integer;Ausgabe:Tlabel;var Sliste:TStringlist;Flaeche:TCanvas):Boolean

Wenn diese Funktionsmethode zum erstemal aufgerufen wird, wird,wenn sich ein Zeichenflächenpunkt mit den Koordinaten X und Y in der (unmittelbaren) Nähe des Mittelpunkts eines Knoten des Graphen befindet,dieser Knoten als erster Knoten ausgewählt.Falls ein erster Knoten ausgewählt wurde,gibt die Function als Resultat false zurück,sonst true.Bei einem zweiten Aufruf der Funktion mit den Koordinaten X und Y eines weiteren Bildschirmpunktes wird,falls sich wiederum dieser Bildschirmpunkt mit den Koordinaten X und Y in der Nähe des Mittelpunktes eines zweiten Knoten des Graphen befindet, dieser zweite Knoten ebenfalls ausgewählt.Falls ein zweiter Knoten gewählt wurde,gibt die Funktion beim Beenden true zurück,sonst false.Zwischen den beiden gewählten Knoten wird der minimale (bezüglich der Pfad- bzw. Kanten-Bewertung Bewertung vom Typ TWert) Pfad im Graphen mit dem ersten Knoten als Anfangsknoten und dem zweiten Knoten als Endknoten erzeugt.Der Pfad wird (markiert) gezeichnet.Der minimale Pfade wird als String als Knotenfolge des Pfads mit Kantensumme und Kantenprodukt in dem Referenzparameter Sliste vom Typ TStringlist gespeichert und außerdem auf dem Label Ausgabe vom Typ TLabel jeweils beim Zeichnen (Zeichnen=true) ausgegeben.Flaeche vom Typ TCanvas gibt die Objekt-Zeichenfläche an,auf der die Pfade gezeichnet werden.Der Gebrauch der Funktionsmethode wird in der Methode TKnotenform. AbstandvonzweiKnotenMouseDown (Sender: TObject) demonstriert, die von TKnotenform.AbstandvonzweiKnotenClick (Sender: TObject) (Unit Ulnoten) aufgerufen wird.Die ungewöhnliche Nutzung und Aufbau der Funktionsmethode (zweimaliger Aufruf) erweist sich als geeignetes und vorteilhaftes Mittel bei der ereignisorientierten Programmierung,wie sie in Delphi unter Windows üblich ist.(TWert= function(Ob:TObject):Extended) Wenn der Graph als Binärbaum aufgefasst werden kann,wird die Methode BinaeresSuchen (Beschreibung: siehe dort) vom Anfangsknoten aus aufgerufen,die nach einem Knotenwert,der gleich dem Wert des Endknotens (Knoten Kno dieser Methode) ist,im Baum binär sucht.Falls ein entsprechender Knoten existiert,wird der Pfad markiert auf der Zeichenoberfläche und als Knotenfolge des Pfades im Ausgabefenster angezeigt.

Anwendung: Alle Pfade zwischen zwei Knoten

Procedure TPfadknoten.ErzeugeallePfadeZielKnoten(Kno:TKnoten)

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen Pfade zu allen anderen Knoten des Graphen,falls existent.Alle Pfade zum Knoten Kno werden in der Pfadliste des Knotens Kno gespeichert.
Falls keine Pfade existieren ist die entsprechende Pfadliste leer (die Pfadliste enthält keinen Eintrag).

Function TPfadgraph.AllePfadezwischenzweiKnotenbestimmen (X,Y:Integer;Ausgabe:TLabel;var Sliste:TStringlist; Flaeche:TCanvas):Boolean

Wenn diese Funktionsmethode zum erstenmal aufgerufen wird, wird,wenn sich ein Zeichenflächenpunkt mit den Koordinaten X und Y in der (unmittelbaren) Nähe des Mittelpunkts eines Knoten des Graphen befindet,dieser Knoten als erster Knoten ausgewählt.Falls ein erster Knoten ausgewählt wurde,gibt die Funktionsmethode als Resultat false zurück,sonst true.Bei einem zweiten Aufruf der Functionsmethode mit den Koordinaten X und Y wird,falls sich wiederum der zugehörige Zeichenflächenpunkt in der Nähe des Mittelpunktes eines zweiten Knoten des Graphen befindet,dieser zweite Knoten ebenfalls ausgewählt.Falls ein zweiter Knoten gewählt wurde,gibt die Funktionsmethode bei Beendigung true zurück,sonst false.Zwischen den beiden gewählten Knoten werden alle Pfade im Graphen mit dem ersten Knoten als Anfangsknoten und dem zweiten Knoten als Endknoten erzeugt.Die Pfade werden als Liste von Strings,wobei jeder String jeweils die Knotenfolge des Pfads,die Kantensumme und das Kantenprodukt enthält,in dem Referenzparameter Sliste vom Typ TStringlist gespeichert und außerdem auf dem Label Ausgabe vom Typ TLabel jeweils beim Zeichnen ausgegeben.Flaeche vom Typ TCanvas gibt die Objekt-Zeichenfläche an,auf der der die Pfade gezeichnet werden.Der Gebrauch der Funktionsmethode wird in der Methode TKnotenform.AllePfadeMouseDown(Sender:TObject;Button:TMouseButton; Shift:TShiftState; X,Y:Integer) demonstriert,die von TKnotenform.AllePfadezwischen zweiKnotenClick(Sender: TObject) aufgerufen wird (jeweils Unit UKnoten).Die ungewöhnliche Nutzung und Aufbau der Function (zweimaliger Aufruf) erweist sich als geeignetes und vorteilhaftes Mittel bei der ereignisorientierten Programmierung,wie sie von in Delphi unter Windows bereitgestellt wird.

Anwendung: Minimales Gerüst des Graphen:

Procedure TPfadgraph.Kruskal(Ausgabe:TLabel;var Sliste:TStringlist;Flaeche:TCanvas)

Die Funktionsmethode bestimmt einen minimalen Spannbaum bzw. einen Wald nach dem Algorithmus von Kruskal.Die Bewertung der Kanten wird durch die Function Bewertung vom Typ TWert

vorgegeben.Der Spannbaum bzw. der Wald wird markiert auf der (Objekt-)Zeichenfläche Flaeche vom Typ TCanvas gezeichnet und die Kantenfolge des Gerüstes als String in der Stringliste Sliste vom Typ TStringlist als Referenzparameter zurückgegeben.Im Demomodus wird der Ablauf des Algorithmus durch Markieren der gerade gewählten Kante verdeutlicht.Im Label Ausgabe werden während des Algorithmus jeweils die zur Zeit gewählten Kanten und nach Ablauf des Algorithmus das Ergebnis als Folge der Kanten-(werte) mit Kantensumme angezeigt. (TWert=function(Ob:TObject):Extended)

Ohne Menü: Tiefer Baum vereinfacht:

Procedure TPfadknoten.ErzeugeTiefeBaumPfadeeinfach (Preorder:Boolean;Flaeche:TCanvas;Ausgabe:TLabel;var SListe:TStringlist)

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen tiefen Baumpfade zu allen anderen Knoten des Graphen,falls existent.Die Pfade werden markiert auf der Zeichenfläche TCanvas gezeichnet.Der Parameter Preorder bestimmt (Preorder=true),daß der Baum gemäß der Ordnung Preorder durchlaufen wird,ansonsten wird die Ordnung Postorder benutzt.

Procedure TPfadgraph.AlletiefenBaumpfade voneinemKnotenbestimmeneinfach (X,Y:Integer;Ausgabe:TLabel;var SListe:TStringList;Flaeche:TCanvas)

Diese Methode bestimmt zunächst zu einem Zeichenflächenpunkt mit den Koordinaten X und Y einen Knoten des Graphen,dessen Mittelpunkt in der Nähe des Punktes liegt.Wenn kein solcher Knoten existiert,wird der Knoten gewählt,der als erstes Element in der Knotenliste des Graphen gespeichert ist.Anschließend werden alle möglichen tiefen Baumpfade des Graphen erzeugt,die den ausgewählten Knoten als Startknoten haben und markiert gezeichnet.Die Pfade werden als Liste von Strings,wobei jeder String jeweils die Knotenfolge des Pfads,die Kantensumme und das Kantenprodukt enthält, in dem Referenzparameter Sliste vom Typ TStringlist gespeichert und außerdem auf dem Label Ausgabe vom Typ TLabel jeweils beim Zeichnen ausgegeben.Flaeche vom Typ TCanvas gibt die Objekt-Zeichenfläche an,auf der der die Pfade gezeichnet werden.

Ohne Menü: Erzeuge Minimale Pfade

Procedure TPfadknoten.ErzeugeminimalePfade

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen minimalen Pfade bezüglich der Funktion (Bewertung der Kan-

ten bzw. Pfade) Bewertung vom Typ TWert zu allen anderen Knoten des Graphen,falls existent.Alle Pfade werden in der Pfadliste des Startknoten gespeichert.Außerdem enthält die Pfadliste aller anderen Knoten außer dem Startknoten den Pfad vom Startknoten aus zu diesem Knoten.Falls keine Pfade existieren sind die entsprechenden Pfade leer (die Pfadliste enthält keinen Eintrag).Der Algorithmus erzeugt alle Pfade zu den Knoten des Graphen und sortiert anschließend die Pfadlisten der einzelnen Knoten der Pfadlänge nach.(TWert=function(Ob:TObject):Extended)

Ohne Menü: Knoten ist Kreisknoten und Graph hat Kreise

Function TPfadknoten.KnotenistKreisknoten:Boolean

Diese Funktionsmethode gibt zurück,ob ein Knoten Teil eines Kreises im Graphen ist.

Function TPfadgraph.GraphhatKreise:Boolean

Die Funktionsmethode testet,ob der Graph Kreise hat.

Ohne Menü: Alle Pfade und minimale Pfade

Procedure TPfadknoten.ErzeugeAllePfadeundMinimalenPfad (ZKno:TPfadKnoten;var Minlist:TKantenliste)

Diese Methode erzeugt vom Knoten als Startknoten aus alle möglichen Pfade zu allen anderen Knoten des Graphen,falls existent.Alle Pfade zum Knoten ZKno werden in der Pfadliste des Knotens ZKno gespeichert.
Falls keine Pfade existieren ist die entsprechende Pfadliste leer (die Pfadliste enthält keinen Eintrag).Der minimale Pfad gemäß der Bewertung Bewertung wird in der Kantenliste Minlist vom Typ TKantenliste gespeichert.Die Methode muß mit der Kantenliste des Graphen für den Parameter Minlist aufgerufen werden.

Function TPfadgraph.AllePfadeundminimalenPfad zwischenzweiKnotenbestimmen (X,Y:Integer;Ausgabe:TLabel;var SListe:TStringList;Flaeche:TCanvas):Boolean

Wenn diese Funktionsmethode zum erstemal aufgerufen wird,wird,wenn sich ein Zeichenflächenpunkt mit den Koordinaten X und Y in der (unmittelbaren) Nähe des Mittelpunkts eines Knoten des Graphen befindet,dieser Knoten als erster Knoten ausgewählt.Falls ein erster Knoten ausgewählt wurde,gibt die Funktionsmethode als Resultat false zurück,sonst true.Bei einem zweiten Aufruf der Funktionsmethode mit den Koordinaten X und Y wird,falls sich wiederum der zugehörige Zeichenflächenpunkt in der Nähe des Mittelpunktes eines zweiten Knoten des Graphen

befindet,dieser zweite Knoten ebenfalls ausgewählt.Falls ein zweiter Knoten gewählt wurde,gibt die Funktionsmethode bei Beendigung true zurück,sonst false.Zwischen den beiden gewählten Knoten werden alle Pfade und der minimale Pfad bezüglich der Bewertung Bewertung vom Typ TWert im Graphen mit dem ersten Knoten als Anfangsknoten und dem zweiten Knoten als Endknoten erzeugt.Die Pfade werden als Liste von Strings,wobei jeder String jeweils die Knotenfolge des Pfads,die Kantensumme und das Kantenprodukt enthält, in dem Referenzparameter Sliste vom Typ TStringlist gespeichert und außerdem auf dem Label Ausgabe vom Typ TLabel jeweils beim Zeichnen ausgegeben.Flaeche vom Typ TCanvas gibt die Objekt-Zeichenfläche an,auf der der die Pfade gezeichnet werden.

Beschreibung der Unit UMath1:

Die Unit UMath1 definiert die Objekt-Datentypen und die zugehörigen Methoden zur Realisierung der mathematischen Anwendungen Netzwerkzeitplan, Hamiltonkreise,Euler-Kreis, Färbbarkeit, Euler-Linie, Endlicher Automat, Graph als Relation, Maximaler Netzfluss und Maximales Matching.Für jede Anwendung wird ein neuer Objekt-Datentyp definiert,der Nachfolger von TInhaltsgraph (Unit UInhGrph) ist.Falls erforderlich,werden auch für die Knoten und Kanten der Anwendung neue Datentypen als Nachfolger von TInhaltsknoten bzw. TInhaltskante festgelegt.Mittels der Funktionsmethode TInhaltsgraph.InhaltskopiedesGraphen wird für jede Anwendung ein neuer Graph (als Nachfolger von TInhaltsgraph,TInhaltsknoten und TInhaltskante) erzeugt,der die gleiche Struktur wie der vorgegebene Graph aufweist,jedoch durch die neuen Objekttypen über geeignete Methoden verfügt,um den entsprechenden Anwendungsalgorithmus ausführen zu können. Dieses Verfahren hat den Vorteil ,daß sich die Anwendungen nicht gegenseitig und auch nicht den ursprünglich vorgegebenen Graph beeinflussen,als auch sind zu jeder Anwendung die unbedingt nur für diese Anwendung benötigten Datenfelder und Methoden direkt in der Objekttyp-Deklaration sichtbar.Durch dieses Verfahren können die Algorithmen einfacher und durchsichtiger gestaltet werden.Außerdem kann dadurch auch jede einzelne Anwendung unter didaktischen und methodischen Gesichtspunkten unabhängig von den anderen als Muster dargestellt werden.

Die folgende Beschreibung ist deshalb sinnvollerweise nach den einzelnen Anwendungen gegliedert.

Anwendung Netzwerkzeitplan:

Diese Anwendung definiert drei neue Objektdatentypen,nämlich TNetzknoten,TNetzkante und TNetzgraph,die sich jeweils von TInhaltsknoten, TInhaltskante und TInhaltsgraph durch Vererbung ableiten.

Methoden von TNetzknoten:

Der Datentyp definiert folgende neue Datenfelder:

Anfang_:Extended
Ende_:Extended
Puffer_:Extended
Ergebnis_:string

Anfang_ nimmt die frühstmögliche Anfangszeit,und Ende_ die spätmöglichste Endzeit aller von diesem Knoten ausgehenden bzw. eingehenden Kanten (die Tätigkeiten mit einer bestimmten Zeitdauer darstellen) auf.Puffer_ bedeutet die Differenz zwischen beiden Werten und Ergebnis_ dient dazu,einen Ergebnisstring,der als Wert des Knoten bei Mausklick auf den Knoten angezeigt werden kann,zu speichern.

Darüberhinaus enthält der Datentyp eine Reihe von Propertys,um über sie auf die oben genannten deklarierten Datenfelder zuzugreifen.

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

Durch die folgenden Methoden von TNetzknoten (Proceduren und Funktionen) wird auf die Propertys lesend und schreibend zugegriffen:

Procedure TNetzknoten SetzeAnfangszeit(Z:Extended)
Function TNetzknoten WelcheAnfangszeit:Extended
Procedure TNetzknoten SetzeEndzeit(Z:Extended)
Function TNetzknoten WelcheEndzeit:Extended
Procedure TNetzknoten SetzePuffer(z:Extended)
Function TNetzknoten WelcherPuffer:Extended
Procedure TNetzknoten SetzeErgebnis(S:string);
Function TNetzknoten WelchesErgebnis:string;

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TNetzknoten.Create

Das ist der Constructor für Instanzen vom Datentyp TNetzknoten.

Function TNetzknoten.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TNetzknoten,wobei die Listenelemente den Inhalt der oben genannten Datenfelder (evtl. umgewandelt) als Strings enthalten und gibt sie zurück.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsgraph zugegriffen.

Procedure TNetzknoten.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsgraph zugegriffen.

Methoden von TNetzkante:

Der Datentyp definiert folgendes neue Datenfeld:

Ergebnis_:string

Ergebnis_ dient dazu,einen Ergebnisstring,der als Wert der Kante bei Mausklick auf die Kante angezeigt werden kann,zu speichern.

Darüberhinaus enthält der Datentyp eine Property,um über sie auf das oben genannte Feld Ergebnis_ zuzugreifen.

Property Ergebnis:string read WelchesErgebnis write SetzeErgebnis

Durch die folgenden Methoden von TNetzknoten (Proceduren und Funktionen) wird auf die Propertys lesend und schreibend zugegriffen:

Procedure TNetzkante.SetzeErgebnis(S:string)
Function TNetzkante.WelchesErgebnis:string

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TNetzkante.Create

Das ist der Constructor für Instanzen vom Datentyp TNetzkante.
Function TNetzkante.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TNetzknoten,wobei die Listenelemente den Inhalt der oben genannten Datenfelder (evtl. umgewandelt) als Strings enthalten und gibt sie zurück.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Procedure TNetzknoten.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Methoden von TNetzgraph:

Der Datentyp definiert keine neue Datenfelder.

Constructor TNetzgraph.Create

Das ist der Constructor für Instanzen vom Datentyp TNetzgraph.

Procedure TNetzgraph.BestimmeAnfangszeit(KnoStart:TNetzknoten; Flaeche:TCanvas)

Diese Methode bestimmt ausgehend von der (Anfangs-)Zeit,die im Startknoten KnoStart vom Typ TNetzknoten gespeichert ist,die frühstmögliche Anfangszeit für alle Knoten des Graphen.Im Demo-modus wird jeweils der Knoten,in dem die Anfangszeit momentan berechnet wird,unter Anzeige der Anfangszeit blau markiert auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas gezeichnet.

Procedure TNetzgraph.BestimmeEndzeit (KnoZiel:TNetzknoten;Flaeche:TCanvas)

Diese Methode bestimmt ausgehend von der (End-)Zeit,die im Ziel-knoten KnoZiel vom Typ TNetzknoten gespeichert ist,die spätestmögliche Endzeit für alle Knoten des Graphen.Im Demomodus wird jeweils der Knoten,in dem die Endzeit momentan berechnet wird, unter Anzeige der Endzeit blau markiert auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas gezeichnet.

Procedure TNetzgraph.BestimmeErgebnisse(var Sliste:TStringlist)

Diese Methode bestimmt die Pufferzeit in den Knoten des Graphen und speichert sie im Datenfeld Pufferzeit_.Außerdem werden alle kritischen Knoten und Kanten rot markiert.(Kritische Knoten und Kanten sind solche,bei denen die Pufferzeit Null ist.)Es wird eine Liste als Referenzparameter Sliste vom Typ TStringlist erzeugt,deren String-Elemente jeweils für alle Knoten den KnotenInhalt,die frühstmögliche Anfangs-und Endzeit sowie die Pufferzeit als auch für jede Kante den Kanteninhalt,den frühstmöglichen und spätmöglichen Anfang,die frühstmögliche und spätmöglichste Endzeit sowie die Pufferzeit enthalten.Diese Da-ten werden auch im Feld Ergebnis jedes Knotens bzw. jeder Kante gespeichert,und können so z.B. in einem Fenster bei Mausklick mit der linken Maustaste auf die Knoten bzw. Kanten angezeigt

werden.

Procedure TNetzgraph.Netz(var G:TInhaltsgraph;var Oberflaeche:TForm; Flaeche:TCanvas;Ausgabe:TLabel;Var Sliste:TStringlist)

Diese Methode bestimmt den Anfangs-und Zielknoten des Netzgraphen (mit den Eigenschaften,dass in bzw. von diesen beiden Knoten keine Kanten einlaufen bzw. ausgehen) und liest eine Anfangs-und Endzeit für diese Knoten ein.Anschließend werden für alle Knoten die frühstmögliche Anfangs- und spätmöglichste Endzeit sowie die Pufferzeit und für alle Kanten die frühstmögliche und spätmögliche Anfangszeit,die frühstmögliche und spätmöglichste Endzeit sowie die Pufferzeit berechnet,und die Ergebnise werden mit den Knoten-und Kanteninhalten zusammen als Elemente vom Typ String in dem Referenzparameter Sliste vom Typ TStringlist gespeichert.
Kritische Knoten und Kanten werden rot markiert auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas ausgegeben,und im Demo-Modus wird jeweils der Knoten,in dem die Zeit momentan berechnet wird,unter Anzeige der jeweiligen Zeit blau markiert auf dieser Objekt-Zeichenfläche Flaeche vom Datentyp TCanvas gezeichnet.Die Gesamtprojektzeit wird berechnet und als Fensterinhalt ausgegeben.Die Referenzparameter Oberflaeche und G bedeuten jeweils die Form vom Typ TForm und der in dieser Form gezeichnete Graph vom Typ TInhaltsgraph.Im Label Ausgabe werden jeweils im Demomodus Kommentare zu den einzelnen Algorithmenphasen angezeigt.Schließlich werden noch die Ergebnisse (frühstmögliche Anfangszeit,spätmöglichste Endzeit usw. /vgl. Methode BestimmeErgebnis) in dem Datenfeld Ergebnis der Knoten und Kanten als String gespeichert.
Falls der Graph Kreise enthält oder ungerichtete Kanten hat,wird die Ausführung des Algorithmus abgebrochen,und es wird eine entsprechende Meldung ausgegeben.

Anwendung Hamiltonkreise:

Diese Anwendung definiert den Objektdatentyp THamiltongraph,der sich von TInhaltsgraph durch Vererbung ableitet.

Methoden von THamiltongraph:

Der Datentyp definiert keine neue Datenfelder.

Constructor THamiltongraph.Create

Das ist der Constructor für Instanzen vom Datentyp

THamiltongraph.

Procedure THamiltongraph.Hamilton(Stufe:Integer;Kno,Zielknoten: TInhaltsknotn;Var Kliste:TKantenliste;Flaeche:TCanvas)

Diese Methode sucht in einem (n-stufigen,n=Anzahl der Knoten) Backtracking-Verfahren nach einem Hamilton-Kreis,wobei sich die Variable Stufe (=n) (ausgehend von Stufe gleich 1) bei jedem Aufruf der Methode um 1 erhöht und sich die Procedure dabei rekursiv selbst aufruft.Es werden jeweils systematisch vom momentanen Knoten Kno vom Typ TInhaltsknoten ausgehend jeweils alle von diesem Knoten durch Pfade ereichbaren Knoten besucht.Backtrackingbedingung ist,daß die Stufenzahl kleiner als die Anzahl der Knoten des Graphen ist,und der erreichbare Knoten noch nicht besucht wurde.Abbruchbedingung ist,dass der Zielknoten (gleich dem Startknoten wegen Kreis) erreicht wurde,und Stufe gleich der Knotenanzahl des Graphen ist (falls Stufe größer zwei ist/wichtig für ungerichtetete Kanten). Zielknoten ist der Knoten,der gleich dem ursprünglichen Startknoten Kno des Verfahrens ist (geschlossener Pfad).In dem Referenzparameter Kliste,werden jeweils die besuchten Kanteninhalte als Kantenliste gespeichert.Flaeche vom Typ TCanvas ist die Objekt-Zeichenfläche,auf der der jeweilige gefundene Hamiltonkreis rot markiert gezeichnet wird.

Procedure THamiltongraph.Hamiltonkreise (Flaeche:TCanvas;Ausgabe:TLabel;var Sliste:TStringlist)

Diese Methode ruft die Methode Hamilton im Backtracking-Verfahren rekursiv auf und bestimmt auf diese Weise alle Hamiltonkreise des Graphen.Die Hamiltonkreise werden als Folge von Kanten mit Kantensumme als Strings in einer Stringliste Sliste vom Typ TStringlist gespeichert und als Referenzparameter zurückgegeben.Auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas werden alle Hamiltonkreise nacheinander rot markiert gezeichnet.Im Label Ausgabe vom Typ TLabel werden die einzelnen Hamiltonkreise als Folge von Kanteninhalten mit der Kantensumme ausgegeben.Zum Schluß wird der Hamiltonkreis mit der kleinsten Kantensumme als Lösung des Traveling-Salesman-Problems ausgegeben.Als Bewertung wird dabei jeweils die durch die Funktion Bewertung der Unit UInhgrph vorgegebene Bewertung gewählt.

Anwendungen geschlossene und offene Eulerlinie:

Diese Anwendung definiert den Objektdatentyp TEulergraph,der sich von TInhaltsgraph durch Vererbung ableitet.

Methoden von TEulergraph:

Der Datentyp definiert keine neue Datenfelder.

Constructor TEulergraph.Create

Das ist der Constructor für Instanzen vom Datentyp TEulergraph.

Procedure TEulergraph.Euler(Stufe:Integer;
Kno,Zielknoten:TInhaltsknoten;var Kliste:TKantenliste;
Flaeche:TCanvas;EineLoesung:Boolean;var Gefunden:Boolean;
Ausgabe:TLabel)

Diese Methode sucht in einem (n-stufigen,n=Anzahl der Knoten) Backtracking-Verfahren nach einem Euler-Kreis oder nach einer Eulerlinie,wobei sich die Variable Stufe (=n) (ausgehend von Stufe gleich 1) bei jedem Aufruf der Methode um 1 erhöht und sich die Procedure dabei rekursiv selbst aufruft.Es werden jeweils systematisch vom momentanen Knoten Kno vom Typ TInhaltsknoten ausgehend alle von diesem Knoten durch Kanten ereichbaren Knoten besucht,wobei nur solche Kanten benutzt werden,die noch nicht besucht wurden.Backtrackingbedingung ist,dass die die Stufenzahl kleiner als die Anzahl der Kanten des Graphen ist und die nächste erreichbare Kante noch nicht besucht wurde.Abbruchbedingung ist,dass eine noch nicht besuchte Kante gewählt werden kann,der Zielknoten ereicht wird und Stufe gleich der Kantenanzahl des Graphen ist.
Zielknoten ist der Knoten,der gleich dem ursprünglichen Startknoten Kno des Verfahrens ist (geschlossener Pfad) bzw. bei der Anwendung Eulerlinie gleich dem zweiten auszuwählenden Knoten,in dem der Eulerlinienpfad enden soll.In dem Referenzparameter Kliste,werden jeweils die besuchten Kanteninhalte als Kantenliste gespeichert.Flaeche vom Typ TCanvas ist die Objekt-Zeichenfläche,auf der der jeweilig gefundene Eulerlinie rot markiert gezeichnet wird.Falls der Referenzparameter EineLoesung true ist,bricht das Verfahren nach der Bestimmung eines Eulerlinie ab.Der Referenzparameter Gefunden testet,ob eine Lösung gefunden wurde.In dem Label Ausgabe vom Typ TLabel werden die Pfade jeweils als Folge von Kanteninhalten ausgegeben.

Procedure TEulergraph.Eulerkreise(Flaeche:TCanvas;
Ausgabe:TLabel;var Sliste:TStringlist;
Anfangsknoten,Endknoten:TInhaltsknoten)

Diese Methode ruft die Methode Euler im Backtracking-Verfahren rekursiv auf und bestimmt auf diese Weise alle Eulerlinien des Graphen.Die Eulerlinien werden als Folge von Kanten mit der Kantensumme als Strings in einer Stringliste vom Typ TStringlist gespeichert und als Referenzparameter zurückgegeben.Auf der Ob-

jekt-Zeichenfläche Flaeche vom Typ TCanvas werden alle Eulerlinien nacheinander rot markiert gezeichnet.Im Label Ausgabe vom Typ TLabel werden die einzelnen Eulerlinien als Folge von Kanteninhalten mit der Kantensumme ausgegeben.Anfangs-und Endknoten sind die Knoten zwischen denen die Eulerlinie gesucht werden soll (d.h. wegen der geschlossenen Linie dieselben Knoten).

Anwendung Färbbarkeit:

Diese Anwendung definiert zwei neue Objektdatentypen,nämlich TFarbknoten und TFarbgraph,die sich jeweils von TInhaltsknoten und TInhaltsgraph durch Vererbung ableiten.

Methoden von TFarbknoten:

Der Datentyp definiert folgende neue Datenfelder:

```
Knotenfarbe_:Integer;
Ergebnis_:string;
```

Den Farben,mit denen die Knoten des Graphs gefärbt werden sollen,werden (Integer-)Zahlen zugeordnet.Diese Zahlen werden in dem Datenfeld Knotenfarbe_ gespeichert.In dem Feld Ergebnis_ wird das Ergebnis (Inhalt des Knotens und momentane Farbzahl) als String gespeichert.

Darüberhinaus enthält der Datentyp eine Reihe von Propertys,um über sie auf die oben genannten deklarierten Datenfelder zu-zugreifen.

```
Property Knotenfarbe:Integer read WelcheFarbzahl write
SetzeFarbzahl;
Property Ergebnis:string read WelchesErgebnis write
SetzeErgebnis;
```

Durch die folgenden Methoden von TFarbknoten (Proceduren und Funktionen) wird auf die Propertys lesend und schreibend zuge-griffen:

```
Procedure TFarbknoten.SetzeFarbzahl(Fa:Integer)
Function TFarbknoten.WelcheFarbzahl:Integer
Procedure TFarbknoten.SetzeErgebnis(S:string)
Function TFarbknoten.WelchesErgebnis:string;
```

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TFarbknoten.Create

Das ist der Constructor für Instanzen vom Datentyp TFarbKnoten.

Function TFarbknoten.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TFarbknoten,wobei die Listenelemente den Inhalt der oben genannten Datenfelder (evtl. umgewandelt) als Strings enthalten und gibt sie zurück. Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Procedure TFarbknoten.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Methoden von TFarbgraph:

Der Datentyp definiert keine neue Datenfelder.

Constructor TFarbgraph.Create

Das ist der Constructor für Instanzen vom Datentyp TFarbgraph.

Procedure TFarbgraph.SetzebeiallenKnotenAnfangsfarbe

Diese Methode setzt bei allen Knoten des Graphen die Farbzahl auf 0.Dieser Zahl ist keine Farbe zugeodnet.Der Wert dient als Ausgangswert für die schrittweise Erhöhung der Farbzahl um 1.

Function TFarbgraph.Knotenistzufaerben(Index:Integer; AnzahlFarben:Integer):Boolean

Diese Funktionsmethode testet,ob der (momentane) Knoten,der dem Wert Index in der Knotenliste des Graphen zugeordnet ist,mit einer der Farben,deren Farbzahlen von der momentan in Farbzahl gespeicherten Farbzahl vergrößert um 1 bis zu AnzahlderFarben reichen,zu färben ist und weist dem Knoten, wenn er zu färben ist,die kleinste der noch möglichen Farbzahlen zu.Dieses wird nach dem Kriterium entschieden,ob die Farbe aller Nachbarknoten (die mit dem momentanen Knoten durch Kanten verbunden sind) ungleich der Farbe ist,mit der der momentane Knoten gefärbt werden soll.Wenn der Knoten nicht mehr zu färben ist,ist der Rückgabwert der Funktionsmethode false sonst true.

Procedure TFarbgraph.Farbverteilung (Index:Integer;AnzahlFarben:Integer varGefunden:Boolean;EineLoesung:Boolean;Flaeche:TCanvas;Ausgabe:TLabel;var Ausgabeliste:TStringlist)

Diese Methode sucht in einem (n-stufigen,n=Anzahl der Knoten) rekursiven Backtracking-Verfahren nach allen Farbverteilungen des Graphen gemäß der vorgegebenen Anzahl der Farben AnzahlFarben vom Typ Integer.Dabei ist die Stufenzahl der Knotenindex Index vom Typ Integer der Knotenliste des Graphen.Backtrackingbedingung ist,dass noch nicht alle Knoten der Knotenliste untersucht wurden und der nächste Knoten noch zu färben ist.Abruchbedingung ist,dass der nächste Knoten nicht mehr zu färben ist oder dass alle Knoten untersucht wurden. Der Parameter EineLoesung vom Typ Boolean bestimmt (EineLoesung=true),dass nur eine Lösung für die Farbverteilung gesucht werden soll.Der Referenzparameter Gefunden vom Typ gibt dann an,dass diese Verteilung gefunden wurde und bricht den Backtrackingalgorithmus ab.Die Farbverteilungen werden in dem Referenzparameter Ausgabeliste vom Typ TStringlist unter Aufruf der Methode ErzeugeErgebnis mit Knoteninhalt und Farbzahl als Elemente vom Typ String gespeichert.Wenn eine neue Farbverteilung für die Knoten gefunden worden ist,werden die gemäß dieser Farbverteilung gefärbten Knoten auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas unter Ausgabe der Farbzahl gezeichnet,und die Farbverteilung wird jeweils im Label Ausgabe vom Typ TLabel ausgegeben.

Procedure TFarbgraph.ErzeugeErgebnis

Diese Methode erzeugt im Feld Ergebnis aller Knoten des Graphen das Ergebnis als String,das aus dem Knoteninhalt und der zugehörigen Farbzahl besteht.Die Methode sollte erst aufgerufen werden,nachdem eine Farbverteilung für alle Knoten bestimmt worden ist.

Procedure TFarbgraph.FaerbeGraph
(Flaeche:TCanvas;Ausgabe:TLabel;var Sliste:TStringlist)

Diese Methode ruft im Bachtracking-Verfahren sich selbst rekursiv auf und bestimmt auf diese Weise alle Farbverteilungen des Graphen.Zuvor wird abgefragt,ob nur eine Lösung gesucht werden soll.In diesem Fall wird nur eine Lösung bestimmt.Die Farbverteilungen werden als Folge von Farbzahlen mit dem zugehörigen Knoteninhalt als Strings in einer Stringliste Sliste vom Typ TStringlist gespeichert und als Referenzparameter zurückgegeben.Auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas werden alle Farbverteilungen als in der entsprechenden Farbe gefärbte Knoten unter Ausgabe der Farbzahl gezeichnet.Im Label Ausgabe vom Typ TLabel werden die einzelnen Farbverteilungen als Folge von Farbzahlen mit dem zugehörigen Knoteninhalt ausgegeben.

Anwendung Endlicher Automat:

Diese Anwendung definiert zwei neue Objektdatentypen,nämlich TAutomatenknoten und TAutomatengraph,die sich jeweils von TInhaltsknoten und TInhaltsgraph durch Vererbung ableiten.

Methoden von TAutomatenknoten:

Der Datentyp definiert folgendes neue Datenfeld:

Knotenart_:TKnotenart

Das Feld bestimmt,ob der Knoten Startzustand,Endzustand,keines von beiden oder Start-und Endzustand ist. (1 ,2 , 0 oder 3 mit TKnotenart = 0..3)

Der Datentyp definiert folgende Property,um auf das Datenfeld zuzugreifen:

Property KnotenArt:TKnotenart read WelcheKnotenart write SetzeKnotenart

Durch die folgenden Methoden von TAutomatenknoten (Proceduren und Funktionen) wird auf die Property lesend und schreibend zugegriffen:

Procedure TAutomatenknoten.TSetzeKnotenart(Ka:TKnotenart)
Function TAutomatenknoten.WelcheKnotenart:TKnotenart

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TAutomatenknoten.Create

Das ist der Constructor für Instanzen vom Datentyp TAutomatenKnoten.

Function TAutomatenknoten.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TAutomatenknoten,wobei ein Listenelement den Inhalt des oben genannten Datenfeldes ArtdesKnoten_ (umgewandelt) als String enthält und gibt sie zurück. Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Procedure TAutomatenknoten.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps.Auf die Einträge der Wertliste wird mittels der

Propertys Wert und Position von TInhaltsknoten zugegriffen.

Methoden von TAutomatengraph:

Der Datentyp definiert folgendes neue Datenfeld:

AktuellerKnoten_:TAutomatenknoten
In diesem Datenfeld wird jeweils der aktuelle Zustand (Knoten) des endlichen Automaten gespeichert.

Der Datentyp definiert folgende Property,um auf das Datenfeld zuzugreifen:

Property MomentanerKnoten:TAutomatenknoten read WelchermomentaneKnoten
write SetzeMomentanenKnoten;

Durch die folgenden Methoden von TAutomatengraph (Proceduren und Funktionen) wird auf die Property lesend und schreibend zugegriffen:

Procedure SetzeaktuellenKnoten(Kno:TAutomatenknoten)
Function WelcheraktuelleKnoten:TAutomatenknoten

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TAutomatengraph.Create

Das ist der Constructor für Instanzen vom Datentyp TAutomatengraph.

Function TAutomatengraph.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TAutomatengraph mit dem Listenelement Momentanerknoten als Element (Speicherung des Index der Knotenliste als String) und gibt sie zurück.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsgraph zugegriffen.
Diese Methode ist evtl. entbehrlich,weil der MomentaneKnoten nicht unbedingt gespeichert werden muß.

Procedure TAutomatengraph.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps. Auf die Einträge der Wertliste wird mittels der

Propertys Wert und Position von TInhaltsgraph zugegriffen. Diese Methode ist evtl. entbehrlich,weil der MomentaneKnoten nicht unbedingt gespeichert werden muß.

Procedure TAutomatengraph.SetzeAnfangswertknotenart

Diese Methode setzt bei allen Knoten des Graphens das Datenfeld ArtdesKnoten auf den Wert Null.

Function TAutomatengraph.BestimmeAnfangsknoten:TAutomatenknoten

Diese Funktionsmethode gibt den ersten Knoten der Knotenliste des Graphen zurück,bei dem das Datenfeld ArtdesKnoten 1 ist.Falls kein solcher Knoten existiert,wird nil zurückgegeben.

Anwendung Graph als Relation:

Diese Anwendung definiert zwei neue Objektdatentypen,nämlich TRealationsknoten und TRelationsgraph,die sich jeweils von Tinhaltsknoten und Tinhaltsgraph durch Vererbung ableiten.

Methoden von TRelationsknoten:

Der Datentyp definiert folgende neue Datenfelder:

Ordnung_:Integer
Ergebnis_:string

Ordnung_ speichert eine Integerzahl,die die Ordnung des Knotens beschreibt.Ergebnis_ speichert neben dem Knoteninhalt die Ordnungszahl des Knoten als String.

Darüberhinaus enthält der Datentyp zwei Propertys,um über sie auf die oben genannten deklarierten Datenfelder zuzugreifen.

Property Ordnung:Integer read WelcheOrdnung write setzeOrdnung
Property Ergebnis:string read WelchesErgebnis write SetzeErgebnis

Durch die folgenden Methoden von TRelationsknoten (Proceduren und Funktionen) wird auf die Property lesend und schreibend zugegriffen:

Procedure TRelationsknoten.SetzeOrdnung(O:Integer)
Function TRelationsknoten.WelcheOrdnung:Integer
Procedure TRelationsknoten.SetzeErgebnis(S:string)
Function TRelationsknoten.WelchesErgebnis:string

Durch die Benutzung von Propertys wird auf die entsprechenden

Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TRelationsknoten.Create

Das ist der Constructor für Instanzen vom Datentyp TRelationsKnoten.

Function TRelationsknoten.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TRelationsknoten,wobei die Listenelemente den Inhalt der oben genannten Datenfelder (umgewandelt) als String enthalten und gibt sie zurück. Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Procedure TRelationsknoten.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.
Methoden von TRelationsgraph:

Der Datentyp definiert keine neue Datenfelder.

Constructor TRelationsgraph.Create

Das ist der Constructor für Instanzen vom Datentyp TRelationsgraph.

Procedure TRelationsgraph.SetzebeiallenKnotenAnfangsOrdnung

Diese Methode setzt bei allen Knoten das Datenfeld Ordnung_ auf den (Anfangs-)Wert Null.

Procedure TRelationsgraph.Schlingenerzeugen(var Sliste:Tstringlist)

Diese Methode erzeugt bei allen Knoten des Graphen,die keine Schlinge haben,eine Schlinge. (Dadurch wird die reflexive Relation erzeugt.)
Eine Schlinge ist eine Kante,die denselben Knoten als Anfangs- und Endknoten hat.In der Stringliste Sliste wird als Aussage gespeichert,ob der ursprüngliche Relation (Graph) reflexiv war oder nicht.

Procedure TRelationsgraph.ErzeugeOrdnung(Flaeche:Tcanvas)

Diese Methode erzeugt eine Ordnungrelation auf einem gerichteten Graphen, der keine Kreise enthalten darf,nach dem TopSort-Algo-

rithmus (Erzeugen der Ordnungsrelation).Die Ordnungszahl wird in dem Datenfeld Ordnung_ gespeichert.Auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas werden die Knoten unter Ausgabe von Knoteninhalt und Ordnungszahl gezeichnet.Im Demomodus wird die Arbeitsweise des Algorithmus demonstriert,indem entsprechende Pfade blau markiert auf der Zeichenfläche Flaeche gezeichnet werden.

Procedure TRelationsgraph.Warshall(var Sliste:Tstringlist)

Diese Methode erzeugt nach dem Algorithmus von Warshall alle transitiven Kanten in einem gerichteten Graph,und fügt sie in den Graph ein (Erzeugen der transitiven Relation).In der Stringliste Sliste wird als Aussage gespeichert,ob die ursprüngliche Relation (der Graph) transitiv war oder nicht.

Procedure TRelationsgraph.ErzeugeErgebnis(var Sliste:TStringlist)

Diese Methode erzeugt in allen Knoten des Graphen das Ergebnis im Datenfeld Ergebnis_.Das Ergebnis ist ein String,der aus dem Knoteninhalt und der Ordnungszahl des Knotens besteht.Anschließend werden alle Ergebnisstrings der Knoten als Elemente in der Liste Sliste vom Typ TStringlist gespeichert und als Referenzparameter zurückgegeben.

Function TRelationsgraph.Relationistsymmetrisch:Boolean

Diese Funktionsmethode gibt als Resultat zurück,ob der Graph eine symmetrische Relation darstellt.

Anwendung Maximaler Netzfluss:

Diese Anwendung definiert drei neue Objektdatentypen,nämlich TMaxflussknoten,TMaxflusskante und TMaxflussgraph,die sich jeweils von TInhaltsknoten,TInhaltskante und TInhaltsgraph durch Vererbung ableiten.

Methoden von TMaxflussKnoten:

Der Datentyp definiert folgende neue Datenfelder:

Distanz_:Extended
Ergebnis_:string

Distanz_ speichert eine Realzahl (Datentyp:Extended),die den maximalen Zuwachs des Flusses durch diesen Knoten (entlang eines Pfades vom Quellen- zum Senkenknoten) beschreibt.Ergebnis_ speichert neben dem Knoteninhalt den maximalen Zuwachs des Flusses durch diesen Knoten als String.

Darüberhinaus enthält der Datentyp zwei Propertys,um über sie auf die oben genannten deklarierten Datenfelder zuzugreifen.

Property Distanz:Extended read WelcheDistanz write SetzeDistanz
Property Ergebnis:string read WelchesErgebnis write SetzeErgebnis;

Durch die folgenden Methoden von TMaxflussknoten (Proceduren und Funktionen) wird auf die Property lesend und schreibend zugegriffen:

Procedure TMaxflussknoten.SetzeDistanz(Di:Extended)
Function TMaxflusknoten.WelcheDistanz:Extended
Procedure TMaxflusskнten.SetzeErgebnis(S:string)
Function TMaxflussknoten.WelchesErgebnis:string

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TMaxflussknoten.Create

Das ist der Constructor für Instanzen vom Datentyp TMaxflussKnoten.

Function TMaxflussknoten.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TMaxflussknoten,wobei die letzten Listenelemente den Inhalt der oben genannten Datenfelder (umgewandelt) als Strings enthalten und gibt sie zurück.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Procedure TMaxflussknoten.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Procedure TMaxflussknoten.ErzeugeErgebnis

Diese Methode erzeugt für jeden Knoten des Graphen das Ergebnis.Ergebnis speichert neben dem Knoteninhalt den Zuwachs des Flusses im Knoten als String.

Methoden von TMaxflusskante:

Der Datentyp definiert folgende neue Datenfelder:

Fluss_:Extended
Ergebnis_:string

Fluss_ speichert eine Realzahl,die den Fluss durch die Kante beschreibt.Ergebnis_ speichert neben dem Kanteninhalt den Fluss durch die Kante als String.

Darüberhinaus enthält der Datentyp zwei Propertys,um über sie auf die oben genannten deklarierten Datenfelder zuzugreifen.

Property Fluss:Extended read WelcherFluss write setzeFluss
Property Ergebnis:string read WelchesErgebnis write SetzeErgebnis

Durch die folgenden Methoden von TMaxflusskante (Proceduren und Funktionen) wird auf die Property lesend und schreibend zugegriffen:

Procedure TMaxflusskante.SetzeFluss(Fl:Extended)
Function TMaxflusskante.WelcherFluss:Extended
Procedure TMaxflusskante.SetzeErgebnis(S:string)
Function TMaxflusskante.WelchesErgebnis:string

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TMaxflusskante.Create

Das ist der Constructor für Instanzen vom Datentyp TMaxflusskante.

Function TMaxflusskante.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TMaxflusskante,wobei die letzten Listenelemente den Inhalt der oben genannten Datenfelder (umgewandelt) als Strings enthalten und gibt sie zurück.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltskante zugegriffen

Procedure TMaxflusskante.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltskante zugegriffen

Procedure TMaxflusskante.ErzeugeErgebnis

Diese Methode erzeugt für jeden Knoten des Graphen das Ergebnis.Ergebnis speichert neben dem Kanteninhalt den Fluss durch die Kante als String.

Methoden von TMaxflussgraph:

Der Datentyp definiert folgendes neue Datenfeld:

Distanz_:Extended

Distanz_ speichert eine Realzahl (Extendedzahl),die den Fluss durch den Quellenknoten bzw. durch den Ziel- oder Senkenknoten beschreibt.

Der Datentyp definiert folgende Property:

Property Distanz:Extended read WelcheDistanz write SetzeDistanz

Durch die folgenden Methoden von TMaxflussgraph (Proceduren und Funktionen) wird auf die Property lesend und schreibend zugegriffen:

Procedure SetzeDistanz(Di:Extended)
Function WelcheDistanz:Extended
Property Distanz:Extended read WelcheDistanz write SetzeDistanz

Durch die Benutzung der Property wird auf das entsprechende Datenfeld nur mit Hilfe von Methoden zugegriffen.

Constructor TMaxflussgraph.Create

Das ist der Constructor für Instanzen vom Datentyp TMaxflussgraph.

Function TMaxflussgraph.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TMaxflussgraph mit dem Listenelement Distanz als Element (Speicherung der Extended-Zahl als String) und gibt sie zurück. Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsgraph zugegriffen.Diese Methode ist evtl. entbehrlich,weil Distanz_ nicht unbedingt gespei
chert werden muß.

Procedure TMaxflussgraph.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern

des Datentyps.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsgraph zugegriffen. Diese Methode ist evtl. entbehrlich,weil Distanz_ nicht unbedingt gespeichert werden muß.

Procedure TMaxflussgraph.LoescheFluss

Diese Methode setzt den Fluss (Datenfeld Fluss_) in allen Kanten des Graphen auf 0.

Procedure TMaxflussgraph.SetzeKnotenDistanz

Diese Methode setzt das Datenfeld Distanz_ bei allen Knoten des Graphen auf 1E32,was unendlich bedeuten soll.

Procedure TMaxflussgraph.Fluss(Kno,Endknoten:TKnoten;var Gefunden Boolean;var Gesamtfluss:Extended;Flaeche:TCanvas;Oberflaeche:TForm)

Diese Methode bestimmt den maximalen Fluss in den Kanten des Graphen sowie durch Quellen-und Senkknoten (Zielknoten) nach dem Algorithmus von Ford-Fulkerson.Die Methode ruft sich selber rekursiv auf.
Der Referenzparameter Gesammtfluss enthält den Gesamtfluss durch Quellen-und Senkknoten (Zielknoten).Der Referenzparameter Gefunden wird auf true gesetzt,wenn der maximale Fluss gefunden worden ist und dient zum Abbruch der Rekursion.Die Parameter Kno und Endknoten vom Typ TKnoten bedeuten Quellen-und Senkenknoten (Anfangs- und Endknoten (bzw.Zielknoten)) beim erstmaligen Aufruf der Methode.Ansonsten bedeutet Kno der momentan berechnete Knoten.Auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas werden die Knoten und Kanten mit Anzeige der Ergebnisse (Fluss durch Knoten und Kanten sowie die Knoten- und Kanteninhalte) gezeichnet.Im Demomodus wird der Ablauf des Algorithmus von Ford-Fulkerson durch Zeichen der Pfade auf dieser Fläche demonstriert.Der Graph darf nur gerichtete Kanten und nur einen Quellen-und Senkenknoten (Knoten,der keine eingehenden bzw. keine ausgehenden Kanten hat) besitzen.Außerdem darf der Graph keine Kreise enthalten.

Procedure TMaxflussGraph.StartFluss(Flaeche:TCanvas;var Gesamtfluss:Extended;Oberflaeche:TForm)

Diese Methode bestimmt zunächst den Quellen- und Senkenknoten (Start- und Zielknoten) des Graphen und zeichnet diese Knoten blau- und grün-markiert auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas.Anschließend wird die Methode Fluss mit diesen beiden Knoten als Parameter für Start- und Endknoten aufgrufen,und damit nach dem Algorithmus von Ford-Fulkerson der maximale Gesamtfluss sowie der Fluss in Kanten und Knoten bestimmt.
Der Gesamtfluss wird durch den Refrenzparameter Gesamtfluss vom

Typ Extended zurückgegeben.Die Kanten und Knoten werden unter Ausgabe von Knoteninhalt und Kanteninhalt sowie dem Fluss durch Knoten und Kanten auf der Objekt-Fläche Flaeche vom Typ TCanvas gezeichnet,und im Demomodus wird die Arbeitsweise des Algorithmus durch die Zeichnung der Suchpfade auf dieser Fläche demonstriert.Der Graph darf nur gerichtete Kanten und nur einen Quellen-und Senkenknoten (Knoten,der keine eingehenden bzw. keine ausgehenden Kanten hat) besitzen.Außerdem darf der Graph keine Kreise enthalten.

Procedure TMaxflussgraph.BestimmeErgebnis(var Sliste:TStringlist)

Diese Methode speichert die Anfangs-und Endknotenwerte einer Kante (getrennt durch Bindestrich),die Schranke der Kante (der Kanteninhalt) sowie den Fluss durch die Kante jeweils für jede Kante des Graphen als Stringelement der Liste Sliste vom Typ TStringlist und gibt diese Liste als Referenzparameter zurück.

Anwendung Maximales Matching

Diese Anwendung definiert zwei Objektdatentypen,nämlich TMatchknoten und TMatchgraph,die sich jeweils von TInhaltsknoten und TInhaltsgraph durch Vererbung ableiten.

Methoden von TMatchKnoten:

Der Datentyp definiert folgende neue Datenfelder:

```
Matchkante_:Integer
Vorigekante_:Integer
```

Matchkante_ speichert eine Integerzahl,die den Verweis auf eine Kante mit dem Knoten als Anfangs- oder Endknoten,die zu dem gesuchten Matching gehört,darstellt.VorigeKante_ speichert eine Integerzahl,die zur vor dem Knoten besuchten Kante gehört mit dem Knoten als Anfangs-oder Endknoten.Dabei bedeutet die Integerzahl den Index in der Kantenliste des Graphen,der zu der Kante gehört.Dieser Zahlenverweis wird von den Property-Methoden in einen Zeigerverweis umgewandelt.Ein Zahlenverweis hat den Vorteil,dass er in der Wertliste als String umgewandelt abgespeichert werden kann.

Darüberhinaus enthält der Datentyp eine Reihe von Propertys,um über sie auf die oben genannten Datenfelder zuzugreifen.

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

Auf Matchkante und VorigeKante wird über den Index in der Kantenliste des Graphen,welcher den Propertys Matchkantenindex und Vorigekantenindex entspricht,zugegriffen

Durch die folgenden Methoden von TMatchknoten (Proceduren und Funktionen) wird auf die Propertys lesend und schreibend zugegriffen:

Procedure TMatchknoten.SetzeMatchkante(Ka:TInhaltskante);
Function TMatchknoten.WelcheMatchkante:TInhaltskante;
Procedure TMatchknoten.SetzeVorigeKante(Ka:TInhaltskante);
Function TMatchknoten.WelcheVorigeKante:TInhaltskante;
Procedure TMatchknoten.SetzeMindex(M:Integer);
Function TMatchknoten.WelcherMindex:Integer;
Procedure TMatchknoten.SetzeVindex(V:Integer)
Function TMatchknoten.WelcherVindex:Integer

Durch die Benutzung der Property wird auf das entsprechende Datenfeld nur mit Hilfe von Methoden zugegriffen.

Constructor TMatchknoten.Create

Das ist der Constructor für Instanzen vom Datentyp TMatchKnoten.

Function TMatchknoten.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TMatchknoten,wobei die letzten Listenelemente den Inhalt der oben genannten Datenfelder (umgewandelt) als Strings enthalten und gibt sie zurück.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Procedure TMatchknoten.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Function Knotenistexponiert:Boolean

Diese Funktionsmethode testet,ob der Konten exponiert ist.D.h. keine Kante,die zum Matching gehört,hat diesen Knoten als Anfangs- oder Endknoten.

Methoden von TMatchgraph:

Der Datentyp definiert keine neue Datenfelder.

Constructor TMatchgraph.Create

Das ist der Constructor für Instanzen vom Datentyp TMatchgraph.

Procedure TMatchgraph.InitialisierealleKnoten

Diese Methode setzt bei allen Knoten des Graphen MatchKante und VorigeKante auf nil.

Procedure TMatchgraph.AlleKnotenSetzeVorigeKanteaufNil

Diese Methode setzt bei allen Knoten des Graphen VorigeKante auf nil.

Procedure TMatchgraph.ErzeugeAnfangsMatching

Diese Methode erzeugt im Graph ein (nicht unbedingt maximales) Anfangs-Matching,indem sie bei jedem Knoten der Knotenliste in aufsteigender Reihenfolge des Graphen prüft,welche aus- und eingehende Kanten isolierte Knoten als Anfangs- oder Endknoten haben,und dann geeignete Kanten zum Matching hinzufügt.

Function TMatchgraph.AnzahlexponierteKnoten:Integer

Diese Funktionsmethode gibt die Anzahl der exponierten Knoten des Graphen zurück.Exponierte Knoten sind solche Knoten,die nicht Anfangs-oder Endknoten einer Matchkante sind.

Procedure TMatchgraph.VergroessereMatching(G:TInhaltsgraph)

Diese Methode wird von der Methode BestimmeMaximalesMatching aufgerufen.Wenn von einem exponierten Knoten aus ein erweiternder Weg gefunden wurde,“färbt“ diese Methode die Kanten um: Kanten in diesem Weg,die bisher zum Matching gehörten,werden wieder aus dem Matching entfernt (Markierung L als Kanteninhalt wird wieder durch den ursprünglichen Kanteninhalt ersetzt),Kanten in diesem Weg,die nicht zum Matching gehörten ,werden zum Matching hinzugefügt (Markierung A wird zu M).So wird das Matching um eine Kante vergrößert.Als Parameter G ist ein Graph vom Typ TInhaltsgraph zu übergeben,in dem noch die ursprünglichen Kanteninhalte gespeichert sind (nötig für die Ersetzung von L durch den ursprünglichen Kanteninhalt).So wird das Kopieren des Graphen und ein Zwischenspeichern vermieden.

Procedure TMatchgraph.BestimmeMaximalesMatching(Flaeche:TCanvas;

Ausgabe:TLabel;G:TInhaltsgraph)

Diese Methode bestimmt ein maximales Matching auf dem vorgegebenen Graphen,indem sie solange noch exponierte Knoten des Graphen vorhanden sind,von diesen aus erweiternde Wege sucht.Wenn ein erweiternder Weg gefunden ist,wird die Methode VergroessereMatching aufgerufen,und durch „Umfärben“ der Kanten ein um eine Kante vergrößertes Matching erzeugt.
Wenn keine erweiternden Wege mehr gefunden werden können,ist das Matching maximal.Anwärter auf eine Kante,die dem vergrößerten Matching angehören könnte,werden beim Suchen des erweiternden Weg mit A gekennzeichnet, Kanten des bisherigen Matchings mit L.Wenn das Suchen des erweiternden Weges nicht erfolgreich war,werden die Markierungen wieder rückgängig gemacht.Sonst wird in der Methode VergroessereMatching „umgefärbt“ (vgl. Methode Vergroessere Matching).Auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas werden die Kanten mit den entsprechenden Markierungen (M,A und L) als Kanteninhalte gezeichnet,und im Demomodus wird das Suchen des ereiternden Weges demonstriert.Ausgabe vom Typ TLabel zeigt im Demo-Modus Kommentare zu den Algorithmusschritten an.Als Parameter G ist ein Graph vom Typ TInhaltsgraph zu übergeben,in dem noch die ursprünglichen Kanteninhalte gespeichert sind (nötig für das Ersetzen von L durch den ursprünglichen Kanteninhalt).So wird das Kopieren des ursprünglichen Graphen und ein Zwischenspeichern vermieden.

Procedure TMatchgraph.ErzeugeListe(var Sliste:Tstringlist)

Diese Methode erzeugt eine Liste Sliste vom Typ TStringlist,in der als Stringelemente jeweils alle Kanten,die zum Matching gehören (mit der Markierung M als Kanteninhalt) gespeichert sind.Die Liste wird als Referenzparameter zurückgegeben.

Beschreibung der Unit UMath2:

Die Unit UMath2 definiert die Objekt-Datentypen und die zugehörigen Methoden zur Realisierung der mathematischen Anwendungen Gleichungssystem, Absorbierende Marovkette, Statische Markovkette, Graph reduzieren, Minimale Kosten, Transportproblem, Optimales Matching und Chinesischer Briefträger.Für jede Anwendung wird ein neuer Objekt-Datentyp definiert,der Nachfolger von TInhaltsgraph (Unit UInhgrph) oder Nachfolger eines in dieser Unit definierten Nachfolgers von TInhaltsgraph ist.Falls erforderlich werden auch für Knoten und Kanten der Anwendung neue Datentypen als Nachfolger von TInhaltsknoten bzw. TInhaltskante festgelegt.Mittels der Funktionsmethode TInhaltsgraph.InhaltskopiedesGraphen wird für jede Anwendung ein neuer Graph (als Nachfolger von TInhaltsgraph,TInhaltsknoten und Tinhaltskante) erzeugt,der die gleiche Struktur (d.h. Knoten-Kanten-Relation) wie der vorgegebene Graph aufweist,jedoch durch

die neuen Objekttypen über geeignete Methoden verfügt,um den entsprechenden Anwendungsalgorithmus ausführen zu können.
Dieses Verfahren hat den Vorteil,dass sich die Anwendungen nicht gegenseitig und auch nicht den ursprünglich vorgegeben Graph beeinflussen,als auch sind in der Objekttypdeklaration zu jeder Anwendung die für nur für diese Anwendung benötigten Datenfelder und Methoden direkt sichtbar (Ausnahme TGleichungssystemKnoten definiert schon das Ergebnisfeld für TMarkovreduziereKnoten.).Dadurch können die Algorithmen einfacher und durchsichtiger gestaltet werden.Außerdem kann dadurch auch jede einzelne Anwendung unter didaktischen und methodischen Gesichtspunkten unabhängig von den anderen als Musterbeispiel dargestellt werden.
Die folgende Beschreibung ist sinnvollerweise nach den einzelnen Anwendungen gegliedert.Die Anwendungen absorbierende und statische Markovketten sowie Graph reduzieren einerseits als auch Minimale Kosten, Transportproblem, Optimales Matching, Chinesischer Briefträger andererseits greifen jeweils auf dieselben Objekt-Datenstrukturen zurück.

Anwendung Gleichungssystem

Diese Anwendung definiert zwei Objektdatentypen, nämlich TGleichungssystemknoten und TGleichungssystemgraph,die sich jeweils von TInhaltsknoten und TInhaltsgraph durch Vererbung ableiten.

Methoden von TGleichungssystemKnoten:

Der Datentyp definiert folgende neue Datenfelder:

```
Nummer_:Integer;
Ergebnis_:string;
```

Die Knoten des Graphen werden zu Beginn durchnummeriert.Das Datenfeld Nummer_ speichert die Nummer des Knoten,die der Index der Knotenliste des Graphen vergrößert um 1 ist.Ergebnis_ ist ein Datenfeld,das Ergebnisse in Form eines Strings speichert.Es wird erst bei der Anwendung Graph reduzieren von dem Datentyp TMarkovReduziereKnoten benutzt,der sich von TGleichungssystemknoten durch Vererbung ableitet.

Darüberhinaus enthält der Datentyp zwei Propertys,um über sie auf die oben genannten deklarierten Datenfelder zuzugreifen.

Property Nummer:Integer read WelcheKnotennummer write SetzeKnotennummer
Property Ergebnis:string read WelchesErgebnis write SetzeErgebnis

Durch die folgenden Methoden von TGleichungssystemknoten (Proceduren und Funktionen) wird auf die Propertys lesend und schreibend zugegriffen:

Procedure TGleichungssystem.SetzeKnotennummer(Nu:Integer)
Function TGleichungssytem.WelcheKnotennummer:Integer
Function TGleichungssystem.WelchesErgebnis:string
Procedure TGleichungssystem.SetzeErgebnis(S:string)

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TGleichungssystemKnoten.Create

Das ist der Constructor für Instanzen vom Datentyp TGleichungssystemKnoten.

Function TGleichungssystemKnoten.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TGleichungssystemknoten,wobei die letzten Listenelemente den Inhalt der oben genannten Datenfelder (umgewandelt) als String enthalten und gibt sie zurück. Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Procedure TGleichungssystemKnoten.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Methoden von TGleichungssystemgraph:

Der Datentyp definiert keine neue Datenfelder.

Constructor TGleichungssystemgraph.Create

Das ist der Constructor für Instanzen vom Datentyp Tgleichungssystemgraph.

Procedure TGleichungssystemgraph.NumeriereKnoten

Diese Methode speichert in dem Datenfeld Knotennummer den Index des Knotens in der Graphknotenliste vermehrt um 1 und nummeriert dadurch die Knoten durch.

Procedure TGleichungssystemgraph.ErgaenzeSchlingen

Diese Methode ergänzt eine Schlinge mit dem Kanteninhalt 0 bei allen Knoten des Graphen,die keine Schlingen besitzen.Schlingen sind solche Kanten,die als Anfangs-und Endknoten denselben Knoten besitzen.

Procedure TGleichungssystemgraph.ErgaenzeKanten

Diese Methode ergänzt eine gerichtete Kante mit dem Kanteninhalt 0 von einem beliebigen Knoten A zu einem beliebigen Knoten B des Graphen,falls noch keine gerichtete Kante von A nach B vorhanden ist.

Procedure TGleichungssystemgraph.EliminiereKnoten(var Kno:TKnoten; Flaeche:TCanvas;varSliste:TStringlist; Ausgabe:TLabel;var Loesung:Extended; var Ende:Boolean;var G:TInhaltsgraph;var Oberflaeche:TForm)

Der Graph wird als Darstellung eines linearen Gleichungssystems aufgefaßt.Außgehend von der Nummerierung der Knoten wird der Inhalt der gerichteten Kante vom i.ten zum j.ten Knoten als Koeeffizient $a_{i,j}$ des Gleichungssystems aufgefaßt.Der Inhalt von Schlingen (d.h. Kanten,deren Anfangs-und Endknoten gleich sind) sind dann gerade die Koffizienten $a_{i,i.}$Die Inhalte der Knoten werden als die Koeffizienten b_i rechts vom Gleichheitszeiten aufgefaßt.Die Knoten repräsentieren durch die Zuordnung der Koeffizienten $a_{i,i}$ als Schlingeninhalte außerdem die Lösungsvariablen x_i.Diese Methode löscht den Knoten,der durch den Referenzparameter Kno vom Typ TKnoten vorgegeben ist und der zu der Lösungsvariablen x_i gehört,aus dem Graphen und ändert alle Knoteninhalte und Kanteninhalte so ab,dass das dem neuen Graph zugeordnete Gleichungssystem aus dem ursprünglichen Gleichungssystem durch Äquivalenzumformung hervorgeht (d.h. für die verbleibenden Variablen ohne x_i dieselbe Lösungsmenge hat).
Der neue Graph wird auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas gezeichnet.Die Lösung für die Variable x_k wird ermittelt,wenn der Graph nur noch aus einem Knoten (der x_k zugeordnet ist) besteht,was durch den Referenzparameter Ende vom Typ Boolean angegeben wird (Ende=true),und wird als Referenzparameter Loesung vom Typ Extended zurückgegeben.Die Lösung wird als Stringelement der Liste Sliste vom Typ TStringlist hinzugefügt,die ebenfalls als Referenzparameter zurückgegeben wird.In dem Label Ausgabe werden Kommentare im Demomodus zu den Algorithmus-Schritten angezeigt. Oberflaeche vom Typ TForm ist die Form,innerhalb derer der Ursprungsgraph G vom Typ TInhaltsgraph oder ein von ihm durch Vererbung abgeleiteter Graph gezeichnet ist.

Anwendung Graph reduzieren

Diese Anwendung definiert zwei Objektdatentypen, nämlich

TMarkovreduziereKnoten und TMarkovreduziereGraph,die sich jeweils von TGleichungssystemKnoten und TGleichungssystemGraph durch Vererbung ableiten.

Methoden von TMarkovreduziereKnoten:

Der Datentyp definiert keine neuen Datenfelder.

Function TMarkovreduziereKnoten.Randknoten:Boolean

Diese Funktionsmethode testet,ob der Knoten ein Randknoten ist.Ein Randknoten ist entweder ein Knoten, der keine von ihm ausgehenden Kanten hat,oder nur eine Schlinge,deren Kanteninhalt 1 ist.Eine Schlinge ist eine Kante mit demselben Anfangs-und Endknoten.

Function TMarkovreduziereknoten.Fehler:Boolean

Diese Funktionsmethode testet bei einem Knoten,der nicht Randknoten ist,ob die Summe der von ihm ausgehenden Kanteninhalte zwischen 0.999 und 1.001 liegt (d.h. dass die Wahrscheinlichkeitssumme 1 ist).Wenn dies der der Fall ist,wird false zurückgegeben,sonst true.Bei Randknoten wird false zurückgegeben.

Function TMarkovreduziereknoten.Auswahlknoten:Boolean

Die Funktionsmethode testet,ob der Knoten ein Auswahlknoten ist.Ein Auswahlknoten ist ein Randknoten mit einer Schlinge,deren Kanteninhalt 1 oder gleich ‘q’ (Quelle) ist.Schlingen sind Kanten,deren Anfangs-und Endknoten gleich sind.

Methoden von TMarkovreduzieregraph:

Der Datentyp definiert keine neue Datenfelder.

Function TMarkovreduzieregraph.Fehler:Boolean

Diese Funktionsmethode testet,ob bei allen Knoten des Graphen außer bei den Randknoten die Summe der Inhalte der ausgehenden Kanten jeweils zwischen 0.999 und 1.001 liegt (d.h.,dass die Wahrscheinlichkeitssumme 1 ist).

Procedure TMarkovreduzieregraph.
SetzeKnotenNullundAuswahlknotenWert

Diese Methode setzt den Inhalt (Wert) aller Knoten außer bei den den Auswahlknoten auf 0 und liest den Inhalt (Wert) für die

Auswahlknoten durch die Anzeige eines Eingabefensters ein.Wenn der numerische Wert nicht im zulässigen Bereich liegt,wird eine Fehlermeldung ausgegeben,und es wird zu einer neuen Eingabe aufgefordert.Die Methode wird bei Flussgraphen,die keine Markov-Graphen sind,benötigt.

Procedure TMarkovreduzieregraph.
SetzeKnotenNullundAuswahlknoteneins

Diese Methode setzt bei allen Knoten außer bei den Auswahlknoten den Knoteninhalt (Wert) auf 0 und den Inhalt (Wert) der Auswahlknoten auf 1.Ein Auswahlknoten ist ein Randknoten mit einer Schlinge,deren Kanteninhalt 1 ist.Schlingen sind Kanten,deren Anfangs-und Endknoten gleich ist.Ein Randknoten ist entweder ein Knoten der keine von ihm ausgehenden Kanten hat,oder nur eine Schlinge,deren Kanteninhalt 1 oder ‘q’ ist.

Procedure TMarkovreduzieregraph.FindeundreduziereSchlinge

Diese Methode entfernt für jeden Knoten des Graphen alle Schlingen (falls vorhanden) und ersetzt die Werte aller von diesem Knoten ausgehenden Kanten durch den Quotienten aus dem bisherigen Wert der Kante dividiert durch die Differenz 1 vermindert um die Summe der Werte der entfernten Schlingen.So wird ein äquivalenter Markov-Graph nach Mason ohne Schlingen erzeugt.Falls ein Knoten keine Schlingen enthält,bleiben alle Kantenwerte unverändert.Schlingen sind Kanten,deren Anfangs-und Endknoten gleich ist.

Procedure TMarkovreduzieregraph.SetzeSchlingenaufNull

Diese Methode setzt bei allen Kanten,die Schlingen sind und deren Wert gleich 1 ist,den Wert auf 0. Schlingen sind Kanten,deren Anfangs-und Endknoten gleich sind.

Procedure TMarkovreduziereGraph.ErgaenzeSchlingenWerteins

Diese Methode fügt für alle Knoten des Graphen,die noch keine Schlinge haben,eine Schlinge mit dem Inhalt (Wert) 1 ein.Schlingen sind Kanten,deren Anfangs-und Endknoten gleich sind.

Procedure TMarkovreduzieregraph.SpeichereSchlingenundKantenum

Diese Methode ersetzt bei allen Kanten des Graphen,die Schlingen sind,den Inhalt (Wert) durch die Differenz 1 minus alter Schlingenwert,außer für den Fall,dass der Graph nur noch einen Knoten hat.(Dann bleibt der Wert gleich.)Für alle Kanten des Graphen,die keine Schlingen sind,wird das Vorzeichen des Kanteninhalts (Wert der Kante) vertauscht.

Procedure TMarkovreduzieregraph.SucheundreduziereParallelkanten

Diese Methode sucht bei allen Knoten des Graphen für alle ausgehenden Kanten (außer den Schlingen) nach Parallelkanten (d.h. für Kanten,die gemeinsame Endknoten haben) und speichert die Summe der Werte aller Inhalte der Parallelkanten als Inhalt (Wert) einer Kante.Die Inhalte (Werte) der übrigen Kanten werden auf Null gesetzt.(Schlingen sind Kanten,deren Anfangs-und Endknoten gleich sind.)So wird nach Mason ein äquivalenter Markov-Graph ohne Parallkanten erzeugt,wenn die Kanten mit dem Inhalt 0 noch aus dem Graph gelöscht werden.

Procedure TMarkovreduziereGraph.LoescheKantenmitWertNull

Diese Methode löscht alle Kanten mit dem Inhalt (Wert) 0 aus dem Graph.

Function TMarkovreduzieregraph.AnzahlRandknoten:Integer

Diese Funktionsmethode bestimmt die Anzahl der Randknoten im aktuellen Graph.

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)

Die Wahrscheinlichkeitsrelationen in einem Markov-Graphen kann als Fluss durch die Knoten und Kanten aufgefaßt werden.Die Flussbeziehungen können nach Mason durch lineare Gleichungen beschrieben werden,wobei die zu suchenden Lösungngsvariablen für die Wahrscheinlichkeit stehen,von diesem Knoten (Zustand) aus,den Endzustand (Randknoten) zu erreichen.Auf diese Weise wird die Ermittlung der Wahrscheinlichkeiten auf die Lösung eines linearen Gleichungssystems zurückgeführt.Daher ruft diese Methode die Vorgängermethode EliminiereKnoten von TGleichungssystemgraph auf.
Diese Methode löscht den Knoten,der durch den Referenzparameter Kno vom Typ TKnoten vorgegeben ist aus dem Graph und erzeugt einen zum ursprünglichen Graphen bezüglich der (Übergangs-)Wahrscheinlichkeiten in den übrigen Knoten äquivalenten Graph ohne den Knoten Kno.Auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas wird der neue Graph gezeichnet.Wenn nur noch ein Knoten vorhanden wird,wird der Parameter Ende auf den Wert true gesetzt und die Übergangswahrscheinlichkeit in den Endzustand als Lösung im Referenzparameter Loesung vom Typ Extended zurückgegeben.Die Lösungen werden jeweils als Stringelemente zu der Liste Sliste vom Typ TStringlist hinzugefügt und ebenfalls als Referenzparameter Loesung zurückgegeben.Die Label Ausgabe1 und Ausgabe2 geben (Ausgabe2 nur im Demomodus) Kommentare zu den Algorithmus-

schritten an.Oberflaeche vom Typ TForm ist die Form,innerhalb derer der Ursprungsgraph G vom Typ TInhaltsgraph oder ein von ihm durch Vererbung abgeleiteter Graph gezeichnet werden.

Procedure TMarkovreduziereGraph.Graphinit(Markov:Boolean; Flaeche:TCanvas;Var Oberflaeche:TForm;var G:TInhaltsgraph;Ausgabe2:Tlabel)

Diese Methode reduziert im vorgegebenen Markovgraph alle Schlingen und Parallelkanten,ergänzt bei Bedarf Kanten und formt den Markovgraph so um,dass er als Flussgraph aufgefaßt werden kann und somit die Methode LöscheKnotenGraphreduzieren anwendbar ist.Der neue Graph wird auf der Objekt-Zeichenfläche vom Typ TCanvas gezeichnet.Im Label Ausgabe2 werden im Demo-Modus Kommentare zu den einzelnen Algorithmusschritten auszugeben.Der Parameter Markov vom Typ Boolean beschreibt,ob es sich um einen Markov-Graph oder Flussgraph (ohne notwendige Wahrscheinlichkeitskantensumme 1) handeln soll (Markov=false).Oberflaeche vom Typ TForm ist die Form,innerhalb derer der Ursprungsgraph G vom Typ TInhaltsgraph oder ein von ihm durch Vererbung abgeleiteter Graph gezeichnet werden.

Anwendungen absorbierende und statische Markovketten

Diese Anwendungen definieren drei Objektdatentypen,nämlich TMarkovknoten,TMarkovkante und TMarkovgraph,die sich jeweils von TInhaltsknoten,TInhaltskante und TInhaltsgraph durch Vererbung ableiten.

Methoden von TMarkovKnoten:

Der Datentyp definiert folgende neue Datenfelder:

```
Wahrscheinlichkeit_:Extended;
MittlereSchrittzahl_:Extended;
Anzahl_:Extended;
Minimum_:Extended;
VorigeAnzahl_:Extended;
Delta_:Extended;
Ergebnis_:string;
Anzeige_:string;
```

Wahrscheinlichkeit_ speichert die Wahrscheinlichkeit,von diesem Knoten (Zustand) aus,einen absorbierenden Randknoten (Endzustand) zu erreichen.MittlereSchrittzahl_ speichert die mittlere Schrittzahl,in der von diesem Knoten (Zustand) aus,der absorbierende Randknoten (Endzustand) erreicht wird.Anzahl_ speichert die Anzahl der (Spiel-)Steine,die auf diesen Knoten momentan entfallen.Minimum_ ist die minimale Anzahl von Steinen,die sich

in dem Knoten (Zustand) angesammelt haben müssen,damit gezogen werden kann.VorigeAnzahl_ speichert die vorige Anzahl von Steinen,d.h. die Anzahl vor dem letzten Zug in diesem Knoten.Delta_ speichert die Anzahl der Steine,die bei einem Zug längs der verschiedenen Kanten zu den Nachbarknoten gezogen werden in dem jeweiligen Nachbarknoten.Ergebnis_ speichert als String den Knoteninhalt (Wert) sowie die Wahrscheinlichkeit,von diesem Knoten einen (absorbierenden) Randzustand zu erreichen als auch die mittlere Schrittzahl für diesen Vorgang.Anzeige_ speichert die Wahrscheinlichkeit als String mit einer Stellenzahl,wie sie als Wert in den Knoten (beim Zeichnen) angezeigt werden soll.

Darüberhinaus enthält der Datentyp eine Reihe von Propertys,um über sie auf die oben genannten deklarierten Datenfelder zuzugreifen.

Property Wahrscheinlichkeit:Extended read WelcheWahrscheinlichkeit write SetzeWahrscheinlichkeit;
Property MittlereSchrittzahl:Extended read WelcheMittlereSchrittzahl write SetzeMittlereSchrittzahl;
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;

Durch die folgenden Methoden von TGleichungssystemknoten (Proceduren und FunKtionen) wird auf die Propertys lesend und schreibend zugegriffen:

Procedure TMarkovKnoten.SetzeWahrscheinlichkeit(Wa:Extended)
Function TMarkovKnoten.WelcheWahrscheinlichkeit:Extended
Procedure TMarkovKnoten.SetzeMittlereSchrittzahl(Schr:Extended)
Function TMarkovKnoten.WelcheMittlereSchrittzahl:Extended
Procedure TMarkovknoten.SetzeAnzahl(Anz:Extended)
Function TMarkovknoten.WelcheAnzahl:Extended
Procedure TMarkovKnoten.SetzeMinimum(Mi:Extended)
Function TMarkovknoten.WelchesMinimum:Extended
Procedure TMarkovKnoten.SetzevorigeAnzahl(Vaz:Extended)
Function TMarkovknoten.WelchevorigeAnzahl:Extended
Procedure TMarkovKnoten.SetzeDelta(De:Extended)
Function TMarkovKnoten.WelchesDelta:Extended
Function TMarkovKnoten.WelchesErgebnis:string
Procedure TMarkovKnoten.SetzeErgebnis(S:string)
Function TMarkovknoten.WelcheAnzeige:string

Procedure TMarkovKnoten.SetzeAnzeige(S:string)

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TMarkovknoten.Create

Das ist der Constructor für Instanzen vom Datentyp TMarkovKnoten.

Function TMarkovknoten.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TMarkovknoten,wobei die Listenelemente den Inhalt der oben genannten Datenfelder (umgewandelt) als String enthalten und gibt sie zurück.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Procedure TMarkovknoten.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsknoten zugegriffen.

Procedure TMarkovknoten.ErhoeheAnzahlumEins(var Steine:Extended)

Diese Methode erhöht die Anzahl der Steine,die sich diesem Knoten (Zustand) befinden,um 1.Zu dem Wert,der in dem Referenzparameter Steine gespeichert ist,wird ebenfalls 1 hinzuaddiert.So kann z.B. die Gesamtzahl der Steine angepaßt werden.

Function TMarkovKnoten.Randknoten:Boolean

Diese Funktionsmethode testet,ob der Knoten ein Randknoten (absorbierender Zustand) ist.Ein Randknoten ist ein Knoten,der entweder keine ausgehenden Kanten oder nur eine Schlinge mit dem Wert (Kanteninhalt) 1 hat.Eine Schlinge ist eine Kante mit gleichem Anfangs-und Endknoten.

Function TMarkovknoten.Fehler:Boolean

Diese Funktionsmethode testet,ob die Summe aller Kanteninhalte der ausgehenden Kanten (Werte) zwischen 0.999 und 1.001 liegt (d.h. ob die Wahrscheinlichkeitssumme 1 ist) und gibt dann false zurück.Ansonsten ist der Wert true.

Function TMarkovknoten.Auswahlknoten:Boolean

Diese Funktionsmethode testet,ob der Knoten ein Auswahlknoten ist.Ein Auswahlknoten ist ein Randknoten (absorbierender Randzustand) mit einer Schlinge,die den Kanteninhalt (Wert) 1 hat.Eine Schlinge ist eine Kante mit gleichem Anfangs-und Endknoten.

Function TMarkovknoten.
KnotenungleichRandistueberkritisch:Boolean

Wenn der Knoten kein Randknoten ist,testet diese Funktionsmethode,ob die Anzahl der Steine,die zu diesem Knoten gehören,über dem (Steine-)Minimum liegt,ab dem man ziehen kann und muß.Wenn der Knoten Randknoten ist,wird false zurückgegeben.

Procedure TMarkovknoten.LadeKnotenkritisch

Diese Methode speichert in allen Knoten soviele Steine (in dem Feld Anzahl_),wie das Minimum minus 1 vorgibt.Dann ist der Knoten mit einem Stein weniger belegt,als es für einen Zug notwendig ist.

Methoden von TMarkovkante:

Der Datentyp definiert keine neuen Datenfelder.

Function TMarkovkante.KanteistSchlingemitWerteins:Boolean

Diese Funktionsmethode testet,ob die Kante eine Schlingemit dem Kanteninhalt (Wert) 1 ist.Eine Schlinge ist eine Kante mit gleichen Anfangs-und Endknoten.

Function TMarkovkante.Fehler:Boolean

Diese Funktionsmethode testet,ob der als Zahl aufgefaßte Kanteninhalt (Wert konvertiert in eine Zahl) zwischen 0 und 1 liegt,d.h. dass die Kantenwahrscheinlichkeit zwischen 0 und 1 liegt.In diesem Fall wird false zurückgegeben,sonst true.

Methoden von TMarkovgraph:

Der Datentyp definiert keine neue Datenfelder.

Constructor TMarkovgraph.Create

Das ist der Constructor für Instanzen vom Typ TMarkovgraph.

Function TMarkovgraph.Fehler:Boolean

Diese Funktionsmethode testet,ob die Kanteninhalte (Werte konvertiert in Zahlen,d.h. Wahrscheinlichkeiten) aller Kanten des Graphen zwischen 0 und 1 liegen und ob die Summe der Kanteninhalte (Werte konvertiert in Zahlen,d.h. Wahrscheinlichkeiten)

aller von einem Knoten ausgehenden Kanten zwischen 0.999 und 1.001 liegen (d.h. gleich 1 ist).In diesem Fall wird false,sonst true zurückgegeben.

Procedure TMarkovgraph.AnzahlgleichNull

Diese Methode setzt bei allen Knoten das Feld Anzahl_ auf den Wert 0.

Procedure TMarkovgraph.LadeKnotenkritischohneStartundRandknoten (Startknoten:TMarkovknoten)

Diese Methode lädt alle Knoten des Graphen (d.h. das Feld Steine_) außer den Randknoten und dem Knoten,der durch Startknoten vom Typ TMarkovknoten vorgegeben ist,mit der kritischen Anzahl der Steine.Die kritische Anzahl Steine ist die um 1 verminderte Zahl von Steinen,die gerade nötig sind,damit ein Zug von diesem Knoten aus möglich ist.Die kritische Anzahl vergrößert um 1 ist im Feld Minimum_ des Knotens gespeichert.

Function TMarkovgraph.AlleKnotenohneRandsindkritisch:Boolean

Diese Funktionsmethode testet,ob ob alle Knoten des Graphen (d.h. das Feld SteineAnzahl) außer den Randknoten kritisch geladen sind.Dann enthält das Feld Anzahl_ den Wert von Minimum_ minus 1.Kritisch geladen ist ein Knoten,wenn beim Hinzufügen eines Steines (Vergrößern von Anzahl_ um 1) gerade ein Zug möglich ist.

Function TMarkovgraph.
AlleKnotenausserRandsindkritischoderunterkritisch:Boolean

Diese Funktionsmethode testet,ob ob alle Knoten des Graphen (d.h. das Feld Anzahl_) außer den Randknoten kritisch oder weniger geladen sind.Dann enthält das Feld Anzahl_ den Wert von Minimum_ minus 1 oder weniger.Kritisch geladen ist ein Knoten,wenn beim Hinzufügen eines Steines (Vergrößern von Anzahl_ um 1) gerade ein Zug möglich ist.

Procedure TMarkovgraph.LadeStartknotenkritischnach(Startknoten: TMarkovknoten;Var Steine:Extended)

Diese Methode lädt den Startknoten vom Typ TMarkovknoten kritisch nach,d.h. das Feld Anzahl_ wird mit dem Wert von Minimum_ minus 1 geladen. Kritisch geladen ist ein Knoten,wenn beim Hinzufügen eines Steines (Vergrößern von Anzahl_ um 1) gerade ein Zug möglich ist.
Der Referenzparameter Steine wird um die Anzahl der Steine vergrößert,um die Anzahl_ vergrößert wurde.

Function TMarkovgraph.EnthaeltAuswahlknoten:Boolean

Diese Funktionsmethode testet,ob der Graph einen Auswahlknoten enthält.Ein Auswahlknoten ist ein Randknoten,der eine Schlinge mit dem Kanteninhalt (Wert) 1 enthält.Eine Schlinge ist eine Kante mit demselben Anfangs-und Endknoten.Ein Randknoten hat entweder keine ausgehenden Kanten oder eine Schlinge mit dem Kanteninhalt (Wert) 1.

Procedure TMarkovgraph.BestimmeMinimum(Genauigkeit:Integer)

Diese Methode bestimmt den Wert für Minimum_ für alle Knoten des Graphen.Wenn man die Kanteninhalte der von einem Knoten ausgehenden Kanten als Brüche auffaßt,ist der Wert das kgV der Nenner.Um die Rechengeschwindigkeit zu steigern,werden bei den Kanteninhalten nur die Anzahl der Stellen nach dem Komma berücksichtigt,die durch Genauigkeit vorgegeben sind.

Procedure TMarkovgraph.SetzeKnotenWahrscheinlichkeitgleichNull

Diese Methode setzt die Felder Wahrscheinlichkeit_ und MittlereSchrittzahl_ bei allen Knoten des Graphen auf Null.

Procedure TMarkovgraph.ZieheSteineabs(Ausgabe:TLabel;var Schrittzahl:Extended;Flaeche:TCanvas;Ob:TForm)

Diese Methode zieht bei den Knoten des Graphen,falls möglich,d.h. wenn der Knoten überkritisch und kein Randknoten ist,die den Wahrscheinlichkeiten in den ausgehenden Kanten (Kanteninhalten) entsprechenden Steine und verteilt sie auf die Zielknoten.Das Feld Steine_ wird im Quellknoten verringert und in den Zielknoten entsprechend vermehrt.Auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas wird der Graph mit der neuen Verteilung der Steine im Demomodus neu gezeichnet und im Feld Ausgabe vom Typ TLabel wird die Verteilung der Steine (im Demomodus) zusätzlich ausgegeben.Der Referenzparameter Schrittzahl wird bei dem Zug eines Steins jeweils um 1 erhöht und das Verfahren wird bei der Erreichung von Schrittzahlen größer als 1 E25 abgebrochen,um Endlosschleifen oder zu langen Rechenzeiten vorzubeugen,ebenfalls bei Steinezahlen>5E11.Ob ist die Knotenform.

Procedure TMarkovgraph.Markovabs (Kno:TMarkovknoten;Ausgabe1,Ausgabe2: TLabel;var Gesamtzahl:Extended;Var SListe:Tstringlist;Flaeche:TCanvas; Genauigkeit:Integer;Ob:TForm)

Diese Methode faßt den Graph als absorbierenden Markovgraph auf und bestimmt die Wahrscheinlichkeit und mittlere Schrittzahl,um vom Knoten Kno in die im Graph vorgegebenen Auswahlknoten (das

sind die durch die Schlinge 1 markierten absorbierenden Randknoten) zu kommen.Wahrscheinlichkeit und mittlere Schrittzahl werden in den Feldern Wahrscheinlichkeit_ und MittlereSchrittzahl_ von Kno gespeichert.Der Algorithmus arbeitet nach dem im Buch von Arthur Engel,Wahrscheinlichkeitsrechnung und Statistik beschriebenen Spielbrett-Algorithmus (Lit 15),wobei solange Steine von Knoten zu Knoten entlang der Kanten des Graphen gemäß den Kantenwahrscheinlichkeiten zu ziehen sind,bis wieder der ürsprünglich kritisch geladene Zustand aller inneren Knoten (Zustände) wiederkehrt.
In den Labeln Ausgabe1 und Ausgabe2 vom Typ TLabel werden im Demomodus Ergebnisse und die Verteilung der Steine auf den Knoten angezeigt.Im Referenzparameter Gesamtzahl vom Typ Extended wird die Gesamtzahl der nachgeladenen Steine angegegeben.In der Stringliste Sliste vom Typ TStringlist werden die Knoteninhalte,die zugehörigen Wahrscheinlichkeiten und die mittlere Schrittzahlen als Strings gespeichert und zurückgegeben.Flaeche vom Typ TCanvas ist die Objekt-Zeichenfläche,auf der der Graph (mit Anzeige der veränderten Steinezahlen in den Knoten im Demomodus) gezeichnet wird.Genauigkeit ist die Anzahl der Nachkommastellen,mit denen gerechnet wird.Ob ist die Form Knotenform.

Procedure TMarkovgraph.
Markovkette(Flaeche:TCanvas;Ausgabe1,Ausgabe2:TLabel;
Var Sliste:TStringlist;Ob:TForm)

Diese Methode faßt den Graph als Markovgraph auf und bestimmt durch Aufruf der Methode Markovabs die Wahrscheinlichkeit und mittlere Schrittzahl,um von jedem Knoten des Graphen oder von einem bestimmten Knoten in die im Graph vorgegebenen Auswahlknoten (das sind die durch die Schlinge mit dem Wert 1 markierten absorbierender Randknoten) zu kommen.Wahrscheinlichkeit und mittlere Schrittzahl werden in den Feldern Wahrscheinlichkeit_ und MittlereSchrittzahl_ der Knoten des Graphen bzw. in dem einen zu berechnenden Knoten gespeichert.Durch ein Eingabefenster wird gefragt,ob alle Knoten des Graphen oder nur ein Knoten berechnet werden soll.
Der Algorithmus arbeitet nach dem im Buch von Arthur Engel, Wahrscheinlichkeitsrechnung und Statistik beschriebenen Spielbrett-Algorithmus (Lit 15),wobei solange Steine von Knoten zu Knoten entlang der Kanten des Graphen gemäß den Kantenwahrscheinlichkeiten gezogen werden,bis wieder der ürsprünglich kritisch geladene Zustand aller inneren Knoten (Zustände) wiederkehrt.
In den Labeln Ausgabe1 und Ausgabe2 vom Typ TLabel werden im Demo-Modus Ergebnisse und die Verteilung der Steine auf den Knoten angezeigt.In der Stringliste Sliste vom Typ TStringlist werden die Knoteninhalte,die zugehörigen Wahrscheinlichkeiten

und die mittlere Schrittzahlen als Strings gespeichert und zurückgegeben.Flaeche vom Typ TCanvas ist die Objekt-Zeichenfläche,auf der der Graph (mit Anzeige der veränderten Steinezahlen in den Knoten im Demo-Modus) gezeichnet bzw. der neu berechnete Graph mit den Knotenwahrscheinlichkeiten angezeigt wird.Ob ist die Form Knotenform.

Methoden von TMarkovstatgraph:

Der Datentyp definiert keine neue Datenfelder.

Constructor TMarkovstatgraph.Create

Das ist der Constructor für Instanzen vom Typ TMarkovstatgraph.

Function TMarkovstatgraph.Fehler:Boolean

Diese Funktionsmethode testet,ob die Kanteninhalte (Werte) (Wahrscheinlichkeiten) aller Kanten des Graphen zwischen 0 und 1 liegen und ob die Summe der Kanteninhalte (Werte) (Wahrscheinlichkeiten) aller von einem Knoten ausgehenden Kanten zwischen 0.999 und 1.001 liegen (d.h. gleich 1 ist).In diesem Fall wird false,sonst true zurückgegeben.

Function TMarkovstatgraph.AnzahlRandknoten:Integer

Diese Funktionsmethode bestimmt die Anzahl der Knoten des Graphen,die keine ausgehenden Kanten haben.

Procedure TMarkovstatgraph.ZieheSteinestat(Ausgabe:Tlabel; Gesamtzahl:Extended;Flaeche:TCanvas;Ob:TForm)

Bevor diese Methode aufgerufen wird,sollte jeder Knoten (Zustand) soviele Steine (im Datenfeld Steine_) enthalten,dass von jedem Knoten aus gezogen werden kann.Diese Methode zieht bei den Knoten des Graphen die den Wahrscheinlichkeiten in den ausgehenden Kanten (Kanteninhalte) entsprechenden Steine und verteilt sie auf die Zielknoten.Das Feld Steine_ wird im Quellknoten verringert und in den Zielknoten entsprechend vermehrt.Auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas wird der Graph mit der neuen Verteilung der Steine im Demomodus neu gezeichnet,und im Feld Ausgabe vom Typ TLabel wird die Verteilung der Steine (im Demomodus) zusätzlich ausgegeben.Gesamtzahl vom Typ Extended ist die Gesamtzahl der Steine in allen Knoten des Graphen,d.h. die Summe der Felder Anzahl_ aller Knoten.Ob ist die Knotenform.

Procedure TMarkovstatgraph.LadeKnotenAnzahlmitMinimum

Diese Methode lädt das Feld Steine_ jedes Knoten des Graphen mit dem Inhalt des Datenfeldes Minimum_.Wenn man die Kanteninhalte

der von einem Knoten ausgehenden Kanten als Brüche auffaßt,ist der Wert für Minimum_ das kgV der Nenner.

Procedure TMarkovstatgraph.BestimmeMinimum(Genauigkeit:Integer)

Diese Methode bestimmt den Wert für das Feld Minimum_ für alle Knoten des Graphen.Wenn man die Kanteninhalte der von einem Knoten ausgehenden Kanten als Brüche auffaßt,ist der Wert das kgV der Nenner.Um die Rechengeschwindigkeit zu steigern,werden bei den Kanteninhalten nur die Anzahl der Stellen nach dem Komma berücksichtigt,die durch Genauigkeit vorgegeben sind.

Procedure TMarkovstatgraph.ErhoeheAnzahlumDelta

Mit dieser Methode kann man nach jedem Zug die Anzahl der Steine im Feld Steine_ für jeden Knoten speichern,indem Steine_ um die zu diesem Knoten hin gezogenen Steine (gespeichert in Delta_) erhöht wird.

Procedure TMarkovstatgraph.SpeichereVerteilung

Diese Methode speichert die vorige Steineverteilung,indem sie den Inhalt des Datenfeldes Anzahl_ in dem Feld VorigeAnzahl_ für jeden Knoten des Graphen speichert.

Procedure TMarkovstatgraph.SummiereAnzahlstat(var Gesamtzahl:Extended)

Diese Methode addiert die Anzahl der Steine aller Knoten in den Feldern Steine_ und gibt die Summe als Referenzparameter Gesamtzahl vom Typ Extended zurück.

Procedure TMarkovstatgraph.
BestimmeWahrscheinlichkeit(Gesamtzahl:Extended)

Diese Methode bestimmt die Wahrscheinlichkeit und die mittlere Schrittzahl in den Feldern Wahrscheinlichkeit_ und MittlereSchrittzahl_ aller Knoten des Graphen.Wenn die Wahrscheinlichkeit 0 ist,wird auch die mittlere Schrittzahl auf 0 gesetzt.

Procedure TMarkovstatgraph.ErzeugeAusgabe(var S:string)

Diese Methode erzeugt einen String S mit den Knoteninhalten und den zugehörigen Wahrscheinlichkeiten der Knoten des Graphen und gibt ihn als Referenzparameter zurück.Außerdem wird im Feld Anzeige_ jedes Knoten der Knoteninhalt und die zugehörige Wahrscheinlichkeit (als String) gespeichert.

Function TMarkovstatgraph.VorigeAnzahlgleichAnzahl(Gesamtzahl:Extended):Boolean

Diese Funktionsmethode testet ob das Feld Anzahl_ gleich dem Feld VorigeAnzahl_ für jeden Knoten des Graphen ist.Dabei wird bis zu einer Gesamtzahl der Steine des Graphen von 15000 exakt geprüft.Oberhalb dieses Wertes wird nur noch geprüft,ob die Differenz der beiden Werte kleiner als der Wert Gesamtzahl (der Steine) dividiert durch 5000 ist,um einen Abbruch des Algorithmus zu erzwingen.

Procedure TMarkovstatgraph.LadeKnotenAnzahlmitkleinsterZahl

Diese Methode lädt jeden Knoten des Graphen (d.h. das Feld Anzahl_) mit sovielen Steinen,dass ein Zug entlang aller ausgehenden Kanten (bezüglich deren Kanteninhalten als Wahrscheinlichkeiten) gerade mit der kleinsten ganzzahligen Anzahl von Steinen möglich ist,und alle Steine (im Feld Steine_) eines Knotens auf die Zielknoten verteilt werden können.

Procedure TMarkovstatgraph.Markovstat (Ausgabe1,Ausgabe2:TLabel;var Gesamtzahl:Extended;Var Sliste:TStringlist;Flaeche:Tcanvas,Genauigkeit:Integer;Ob:TForm)

Diese Methode faßt den Graph als stationären Markovgraph auf und bestimmt die Wahrscheinlichkeit und mittlere Schrittzahl in jedem Knoten (Zustand) des Graphen für den Fall einer stationären Verteilung.Wahrscheinlichkeit und mittlere Schrittzahl werden in den Feldern Wahrscheinlichkeit_ und MittlereSchrittzahl_ der Knoten gespeichert.Der Algorithmus arbeitet nach dem im Buch von Arthur Engel,Wahrscheinlichkeitsrechnung und Statistik beschriebenen Spielbrett-Algorithmus (Lit 15),wobei solange Steine von allen Knoten zu den Zielknoten entlang der Kanten des Graphen gemäß den Kantenwahrscheinlichkeiten gleichzeitig zu ziehen sind,bis sich die Anzahl der Steine in den Knoten nicht mehr ändert.
In den Labeln Ausgabe1 und Ausgabe2 vom Typ TLabel werden im Demomodus Ergebnisse und die Verteilung der Steine in den Knoten angezeigt.Im Referenzparameter Gesamtzahl vom Typ Extended wird die Gesamtzahl der nachgeladenen Steine angegegeben.In der Stringliste Sliste vom Typ TStringlist werden die Knoteninhalte,die zugehörigen Wahrscheinlichkeiten und die mittlere Schrittzahlen als Strings gespeichert und zurückgegeben.Flaeche vom Typ TCanvas ist die Objekt-Zeichenfläche,auf der der Graph mit Anzeige der veränderten Steinezahlen in den Knoten im Demomodus gezeichnet wird.Genauigkeit ist die Anzahl der Nachkommastellen,mit denen gerechnet wird.Ob ist die Form Knotenform.

ProcedureTMarkovstatgraph.Markovkettestat(Flaeche:TCanvas;Ausgabe1,Ausgabe:Tlabel;var Sliste:TStringlist;Ob:TForm)

Diese Methode ruft die Methode Markovstat auf. Sie faßt den Graph als stationären Markovgraph auf und bestimmt die Wahrscheinlichkeit und mittlere Schrittzahl in jedem Knoten (Zustand) des Graphen für den Fall einer stationären Verteilung.Wahrscheinlichkeit und mittlere Schrittzahl werden in den Feldern Wahrscheinlichkeit_ und MittlereSchrittzahl_ der Knoten gespeichert.Der Algorithmus arbeitet nach dem im Buch von Arthur Engel,Wahrscheinlichkeitsrechnung und Statistik beschriebenen Spielbrett-Algorithmus (Lit 15),wobei solange Steine von allen Knoten zu den Zielknoten entlang der Kanten des Graphen gemäß den Kantenwahrscheinlichkeiten gleichzeitig zu ziehen sind,bis sich die Anzahl der Steine in den Knoten nicht mehr ändert.
In den Labeln Ausgabe1 und Ausgabe2 vom Typ TLabel werden im Demomodus Ergebnisse und die Verteilung der Steine auf den Knoten angezeigt.In der Stringliste Sliste vom Typ TStringlist werden die Knoteninhalte,die zugehörigen Wahrscheinlichkeiten und die mittlere Schrittzahlen als Strings gespeichert und zurückgegeben.Flaeche vom Typ TCanvas ist die Objekt-Zeichenfläche,auf der der Graph mit Anzeige der veränderten Steinezahlen in den Knoten im Demomodus gezeichnet wird.Zu jedem Knoten des Graphs werden der Knoteninhalt,die Wahrscheinlichkeit und die mittlere Schrittzahl als String im Feld Ergebnis gespeichert.Ob ist die Form Knotenform.

Anwendungen Minimale Kosten,Transportproblem,Optimales Matching,Chinesischer Briefträger

Diese Anwendungen definieren zwei Objektdatentypen,nämlich TMinimaleKostenKante und TMinimaleKostenGraph,die sich jeweils von TInhaltskante und TInhaltsgraph durch Vererbung ableiten.

Methoden von TMinimaleKostenKante:

Der Datentyp definiert folgende neue Datenfelder:

```
Fluss_:Extended
Kosten:Extended
ModKosten_:string
KostenWert_:Extended
Schranke_:Extended
Richtung_:Boolean
PartnerKante_:Integer
Ergebnis_:string
```

Fluss_ enthält den Fluss durch die Kanten,Kosten_ sowie Kostenwert_ speichern die Kosten pro Flusseinheit beziehungsweise die

Kosten für den gesamten Kantenfluss und ModKosten_ die von den Kosten_ abgeleiteten modifizierten Kosten pro Flusseinheit der jeweiligen Kante.Schranke_ speichert die Schranke zu jeder Kante,d.h. den maximalen Wert für den Fluss durch diese Kante.Richtung_ enthält die Information,ob die Kante eine Vorwärts- (Richtung_=false) oder Rückwärtskante ist.PartnerKante_ enthält einen Verweis auf die entsprechende Partnerkante,d.h. bei einer Vorwärtskante die Rückwärtskante und umgekehrt.Der Verweis ist dabei der Index,unter dem die Kante in der Kantenliste des Graphen gespeichert ist.Dieser Verweis wird in einen Zeigerverweis umgewandelt,kann aber andererseits als Integerzahl Index leichter in der Wertliste als String gespeichert werden.Ergebnis_ schließlich nimmt Ergebnisse z.B. Fluss- oder Kostenwerte als String auf.

Darüberhinaus enthält der Datentyp eine Reihe von Propertys,um über sie auf die oben genannten deklarierten Datenfelder zuzugreifen.

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

Durch die folgenden Methoden von TMinimaleKostenkante (Proceduren und Funktionen) wird auf die Propertys lesend und schreibend zugegriffen:

Procedure TMinimaleKostenkante.SetzeFluss(Fl:Extended)
Function TMinimaleKostenkante.WelcherFluss:Extended
Procedure TMinimaleKostenkante.SetzeKosten(Ko:Extended)
Function TMinimaleKostenkante.WelcheKosten:Extended
Procedure TMinimaleKostenkante.SetzeModKosten(Mko:string)
Function TMinimaleKostenkante.WelcheModKosten:string;
Procedure TMinimaleKostenkante.SetzeKostenWert(Ako:Extended)
Function TMinimaleKostenkante.WelcherKostenWert:Extended
Procedure TMinimaleKostenkante.SetzeSchranke(Schr:Extended)
Function TMinimaleKostenkante.WelcheSchranke:Extended

Procedure TMinimaleKostenkante.SetzeRichtung(Ri:Boolean)
Function TMinimaleKostenkante.WelcheRichtung:Boolean
Procedure TMinimaleKostenkante.SetzePartnerKante
(Ka:TMinimaleKostenkante)
Function TMinimaleKostenkante.WelchePartnerkante:
TMinimalekostenkante
Procedure TMinimaleKostenkante.SetzeKantenindex(I:Integer)
Function TMinimaleKostenkante.WelcherKantenindex:Integer
Procedure TMinimaleKostenkante.SetzeErgebnis(Erg:string)
Function TMinimaleKostenkante.WelchesErgebnis:string

Auf die Partnerkante wird mittels ihres Index in der Kantenliste des Graphen zugegriffen.

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TMinimaleKostenkante.Create

Das ist der Constructor für Instanzen vom Datentyp TMinimaleKostenKante.

Function TMinimaleKostenkante.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TMinimalekostenkante,wobei die Listenelemente den Inhalt der oben genannten Datenfelder (umgewandelt) als String enthalten und gibt sie zurück. Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltskante zugegriffen.

Procedure TMinimaleKostenKante.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps. Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltskante zugegriffen.

Methoden von TMinimaleKostengraph:

Der Datentyp definiert folgende neue Datenfelder:

Gesamtfluss_:Extended
VorgegebenerFluss_:Extended
Ganzzahlig_:Boolean

Gesamtfluss_ speichert den Ergebnis-Gesamtfluss von der Quelle zur Senke, VorgegebenerFluss_ speichert den vom Benutzer vorgegebenen Gesamtfluss von der Quelle zur Senke, Ganzzahlig_ speichert, ob der Fluss im Graphen nur ganzzahlige Werte annehmen soll.

Darüberhinaus enthält der Datentyp eine Reihe von Propertys,um über sie auf die oben genannten deklarierten Datenfelder zu-zugreifen.

Property Gesamtfluss:Extended read WelcherGesamtfluss write SetzeGesamtfluss
Property VorgegebenerFluss:Extended read WelcherVorgegebeneFluss write SetzeVorgegebenenFluss
Property Ganzzahlig:Boolean read WelcherWertGanzzahlig write SetzeGanzzahlig

Durch die folgenden Methoden von TMinimaleKostenGraph (Proceduren und Functionen) wird auf die Propertys lesend und schreibend zugegriffen:

Procedure TMinimaleKostenGraph.SetzeGesamtfluss(Gfl:Extended)
Function TMinimaleKostenGraph.WelcherGesamtfluss:Extended
Procedure TMinimaleKostenGraph.SetzeVorgegebenenFluss (Vfl:Extended)
Function TMinimaleKostenGraph.WelcherVorgegebeneFluss:Extended
Procedure TMinimaleKostenGraph.SetzeGanzzahlig(Gz:Boolean)
Function TMinimaleKostenGraph.WelcherWertganzzahlig:Boolean

Durch die Benutzung von Propertys wird auf die entsprechenden Datenfelder nur mit Hilfe von Methoden zugegriffen.

Constructor TMinimaleKostenGraph.Create

Das ist der Constructor für Instanzen vom Datentyp TMinimaleKostenGraph.

Function TMinimaleKostenGraph.Wertlisteschreiben:TStringlist

Diese Funktionsmethode erzeugt die Wertliste von TMinimalekostenGraph,wobei die letzten Listenelemente den Inhalt der oben genannten Datenfelder (umgewandelt) als Strings enthalten und gibt sie zurück. Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsgraph zugegriffen.

Procedure TMinimaleKostenGraph.Wertlistelesen

Diese Methode liest die Einträge der Wertliste als Strings und speichert sie (evtl. mittels Typkonversion) in den Datenfeldern des Datentyps.Auf die Einträge der Wertliste wird mittels der Propertys Wert und Position von TInhaltsgraph zugegriffen.

Procedure TMinimalekostengraph.SpeichereSchranken

Diese Methode speichert den Wert einer Kante (bestimmt durch Kantenwertposition_) im Feld Schranke_ bei allen Kanten des Graphen.

Procedure TMinimalekostengraph.SpeichereSchrankenalsKosten

Diese Methode speichert den Wert einer Kante (bestimmt durch Kantenwertposition_) im Feld Kosten bei allen Kanten des Graphen.

Procedure TMinimaleKostenGraph.SpeichereKosten

Diese Methode speichert den Inhalt von Kosten_ in dem Feld KostenWert_ bei allen Kanten des Graphen.

Procedure TMinimaleKostenGraph.LadeKosten

Diese Methode speichert den Inhalt von KostenWert_ im Feld Kosten_ bei allen Kanten des Graphen.

Procedure TMinimaleKostengraph.InitialisiereKanten

Diese Methode setzt bei allen Kanten des Graphen den Fluss (Fluss_) ,die modifizierten Kosten (ModKosten_) sowie den KostenWert (KostenWert_) auf 0 und die Partnerkante (Partnerkante_) auf nil.

Procedure TMinimaleKostenGraph.
SetzeVorwaertsKantenRichtungaufvor

Diese Methode setzt bei allen Kanten des Graphen das Feld Richtung_ auf false und sollte aufgerufen werden,wenn noch keine Rückwärtskanten des Graphen erzeugt worden sind.

Procedure TMinimaleKostengraph.ErzeugeRueckkanten

Diese Methode erzeugt zu allen Kanten des Graphen,die noch keine Rückwärtskante besitzen eine Rückwärtskante und fügt sie in den Graph ein.Bei einer Rückwärtskante sind Anfangs-und Endknoten (gegenüber der ursprünglichen Kante) vertauscht,der Kanteninhalt Inhalt_ ist ‘R’ ,die Weite_ ist der negative Wert der Weite_ der ursprünglichen Kante,die (Kanten-)Richtung_ ist true,der (Kanten-)Fluss_ ist 0 und die (Kanten-) Kosten_ sowie die (Kanten-)Schranke_ werden von der ursprünglichen Kante übernommen.Partnerkante_ der Rückwärtskante ist die ursprüngliche Kante.

Procedure TMinimaleKostengraph.EliminiereRueckkanten

Diese Methode entfernt alle Rückkanten des Graphen,bei denen die (Kanten-) Richtung_ true ist,aus dem Graphen.

Procedure TMinimaleKostenGraph.ErzeugeModKosten

Diese Funktionsmethode erzeugt die modifizierten Kosten für jede Kante des Graphen und speichert sie im Feld ModKosten_.Die modifizierten Kosten sind für Vorwärtskanten gleich dem im Feld (Kanten-)Kosten_ gespeicherten Wert,falls der Fluss (Feld Fluss_) durch die Kante unterhalb des Wertes liegt,der im Feld (Kanten-)Schranke_ gespeichert ist und sonst unendlich,was durch den Wert 1 E32 dargestellt wird.Für Rückwärtskanten sind die modifizierten Kosten gleich dem negativen Wert des Wertes,der in (Kanten-)Kosten_ gespeichert ist,wenn der Fluss (Feld Fluss_) durch die Kante größer als 0 ist und sonst unendlich (für Fluss_ gleich 0),was durch einen Wert von 1 E32 dargestellt wird.Bei ganzahliger Rechnung müssen die Schranken unterhalb von 10000 liegen.

Function TMinimaleKostenGraph.
SucheminimalenWegundbestimmeneuenFluss
(Quelle,Senke:TInhaltsknoten;Flaeche:TCanvas):Boolean

Diese Funktionsmethode bestimmt von dem Quellknoten Quelle zum Senkenknoten Senke vom Typ TInhaltsknoten einen minimalen Pfad bezüglich der Bewertung durch die modifizierten Kosten (vgl. Methode ErzeugeModKosten).Der Fluss längs der Vorwärtskanten dieses Pfades wird unter Beachtung der vorgegebenen Schranken um den größtmöglichen Fluss angehoben.Bei Rückwärtskanten wird der Fluss um den entsprechenden Betrag vermindert unter Beachtung der Regel,dass kein Fluss negativ sein kann.Auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas wird der Pfad im Demomodus unter Anzeige der neuen Flusswerte markiert gezeichnet.Wenn ein minimaler Pfad gefunden wurde,ist der Rückgabewert der Funktionsmethode true,sonst false.

Procedure TMinimaleKostenGraph.BestimmeFlussKostenfuerKanten(Var SListe:TStringlist)

Diese Methode bestimmt die Gesamtkosten für jede (Vorwärts-) Kante des Graphen als Produkt aus Kosten pro Flusseinheit (Feld (Kanten-)Kosten_) und dem Fluss durch die Kante (Fluss_) und speichert sie im Feld KostenWert_.Die Ergebnisse aller Kanten des Graphen bestehend aus Anfangs-und Endknoten der Kante,dem Fluss_ durch die Kante,den Gesamtkosten sowie den Kosten pro Flusseinheit werden jeweils als Strings in der Liste SListe vom Typ TStringlist gespeichert und als Referenzparameter zurückgegeben.

Function TMinimaleKostenGraph.GesamtKosten:Extended

Diese Funktionsmethode bestimmt die Gesamtkosten für den vorgegebenen Fluss durch Addition der Gesamtkosten der einzelnen Kanten (Produkt aus (Kanten-) Fluss_ und (Kanten-)Kosten_) und gibt sie zurück.

Procedure TMinimaleKostengraph.BildeinverseKosten

Diese Methode bildet zur Lösung der Aufgabe,bei der statt der minimalen Kosten die maximalen Kosten gesucht sind,die inversen Kosten (pro Flusseinheit) zu jeder Kante des Graphen und speichert sie im Feld Kantenkosten.Dazu wird zunächst das Maximum aller Kantenkosten gesucht.Die inversen Kosten berechnen sich dann als Differenz dieses Maximums und den ursprünglichen Kosten pro Flusseinheit dieser Kante.

Function TMinimalekostengraph.BestimmeQuelleundSenke(var Quelle,Senke: TInhaltsknoten; Flaeche:TCanvas;Anzeigen:Boolean):Boolean

Diese Funktionsmethode bestimmt in dem vorgegebenen Graph den Quellenknoten Quelle und den Senkenknoten Senke vom Typ TInhaltsknoten und gibt sie als Referenzparameter zurück.Der Quellenknoten besitzt nur ausgehende Kanten und der Senkenknoten nur eingehende Kanten.Falls es mehrere Quellenknoten oder mehrere Senkenknoten gibt,wird eine Fehlermeldung ausgegeben.Der Rückgabewert der Funktionsmethode ist dann false,sonst true.Auf der Objekt-Zeichenfläche Flaeche vom Typ TCanvas werden die beiden Knoten blau bzw. grün markiert gezeichnet.Wenn die Variable Anzeigen vom Typ Boolean true ist,werden Quellen-und Senkenknoten auch in einem Ausgabefenster unter Anzeige des Knoteninhalts ausgegeben.

Function TMinimaleKostengraph. ErzeugeQuellenundSenkenknotensowieKanten(Var Quelle,Senke:TInhaltsknoten;var Fluss:Extended;Art:char):Boolean

Diese Funktionsmethode erzeugt für die Anwendungen Transportproblem,Optimales Matching und Chinesischer Briefträger zusätzliche Quellen- und Senkenknoten und fügt sie in den Graph ein.Die Art der Anwendung wird vorgegeben durch den Parameter Art vom Typ char.Die Buchstaben ‘t’,’o’ und ‘b’ stehen dabei für die drei Anwendungen Transportproblem,Optimales Matching und Chinesischer Briefträger.Außerdem wird von den Knoten des Graphen,die nicht Quellen-und Senkenknoten sind,jeweils eine neue Kante in den Graph eingefügt,deren Anfangs- bzw. Endknoten der Quellen- bzw. der Senkenknoten ist.Der Kantenwert dieser Kanten ist ‘U’,die Weite_ 0 und die Kosten_ ebenfalls 0.Je

nach Anwendung wird das Feld (Kanten-)Schranke_ unterschiedlich bestimmt.Bei der Anwendung Transportproblem (Art='t') werden die (Kanten-)Schranke_n mittels Eingabefenster von der Tastatur zu jeder dieser Kanten eingelesen (bei Eingaben außerhalb des zulässigen numerischen Bereichs wird die Eingabe wiederholt).Bei der Anwendung Optimales Matching (Art='o') sind die Schranke_n 1 und bei der Anwendung Chinesischer Briefträger gleich dem Absolutwert des Knotengrades.In dem Referenzparameter Fluss wird jeweils die Summe der Schrankenwerte aller neu erzeugten Kanten (gleich Fluss_ durch diese Kanten) für die einzelnen Anwendungen für den Quellenknoten gezählt.
Falls die Summe dieser Schranke_n für den Quellenknoten ungleich der Summe beim Senkenknoten sein sollte,wird diese Ungleichheit durch eine Meldung angezeigt.Dann ist der Rückgabewert der Function false andernfals true.

Procedure TMinimalekostengraph.
BestimmeQuelleSenkesowievorgegebenenFluss
(var Quelle,Senke:TInhaltsknoten;Ausgabe1:TLabel;
Flaeche:TCanvas;Art:char)

Diese Methode ruft die Methode ErzeugeQuellenundSenkenknotensowieKanten auf (siehe dort).Für die Parameter im Methodenkopf gilt ebenfalls das dort Gesagte.Der Parameter Ausgabe1 vom Typ TLabel dient dazu,einen Kommentar zu den ausgeführten Aktionen anzuzeigen.Der Rückgabewert der Funktionsmethode ErzeugeQuellenundSenkenknotensowieKanten wird ausgewertet,und im Falle von false wird eine entsprechende Meldung ausgegeben (Fluss durch Senkenknoten ungleich durch Quellenknoten).

Procedure TMinimaleKostengraph.SetzeSchrankenMax

Diese Methode setzt das durch Position und Wert bestimmte Feld bei allen Kanten des Graphen auf den Wert 1 E32 (unendlich).

Procedure TMinimaleKostengraph.LoescheQuellenundSenkenknoten
(Quelle,Senke: TInhaltsknoten)

Diese Methode löscht den Quellenknoten Quelle und den Senkenknoten Senke vom Typ TInhaltsknoten aus dem Graph.

Procedure TMinimaleKostengraph.ErzeugeunproduktiveKanten

Diese Methode wird von der Anwendung Chinesischer Briefträger benutzt und fügt soviele zusätzliche gerichtete Kanten parallel (Abstand der Weite_ jeweils 20) zu schon im Graph befindlichen Kanten ein,wie der gerundete Wert des Feldes (Kanten-)Fluss_ angibt (für (Kanten-)Fluss_ größer oder gleich 1).Der Wert (Kanteninhalt) der neuen Kanten sowie der ursprünglichen Kanten

ist der Wert (bestimmt durch Kantenwertposition_) von Kantenkosten,welcher beim Briefträgerproblem der Bedeutung der Weglänge (ursprünglicher Inhalt von (Kanten-)Inhalt_) entspricht.

Procedure TMinimaleKostengraph.LoescheNichtMatchKanten

Diese Methode löscht alle Kanten aus dem Graph,deren Fluss (Feld (Kanten-)Fluss_) ungleich 1 ist.

Procedure TMinimalekostengraph.MinimaleKosten (Flaeche:TCanvas;var G:TInhaltsgraph; Var Oberflaeche:TForm;Ausgabe1:TLabel;Maximal:Boolean; var Quelle,Senke:TInhaltsknoten;Art:char;var Sliste:TStringlist)

Diese Methode sucht zunächst im vorgegebenen gerichteten Graph nach einem Quellenknoten Quelle (das ist ein Knoten,der nur ausgehende Kanten besitzt) und einem Senkenknoten Senke (das ist ein Knoten,der nur eingehende Kanten besitzt).Die Knoten werden als Referenzparameter vom Typ TInhaltsknoten zurückgegeben.Vor Aufruf dieser Methode muß daher sichergestellt sein,dass jeweils (nur) ein Knoten mit diesen Eigenschaften im Graph existiert,sonst wird die Ausführung mit einer entsprechenden Fehlermeldung abgebrochen.
Zu jeder (gerichteten) Kante des Graphen wird eine gerichtete Rückwärtskante (mit vertauschtem Anfangs-und Endknoten) in den Graph eingefügt.
Die Methode arbeitet dann nach dem Algorithmus von Busacker und Gowden zur Bestimmung eines Flusses mit minimaler Kosten und sucht unter Bestimmung von modifizierten Kosten zu dem momentanen Fluss (Anfangsfluss Null) und den vorgegebenen Kosten solange nach minimalen (modifizierten) Kostenpfaden vom Quellknoten zum Senkenknoten,bis die modifizierten Kosten aller möglichen Pfade unendlich groß sind.Wenn jeweils ein minimaler Kostenpfad gefunden ist,wird der Fluss längs der Kanten dieses Pfades unter Beachtung der Schranken maximal erhöht und anschließend werden wieder erneut die neuen modifizierte Kantenkosten gebildet.
Die modifizierten Kosten sind für Vorwärtskanten gleich den im Feld (Kanten-)Kosten_ gespeicherten Wert,falls der Fluss (Feld (Kanten-)Fluss_) durch die Kante unterhalb des Wertes liegt,der im Feld (Kante-)Schranke_n gespeichert ist und sonst unendlich,was durch den Wert 1 E32 dargestellt wird.Für Rückwärtskanten sind die modifizierten Kosten gleich dem negativen Wert des Wertes,der in Kantenkosten gespeichert ist,wenn der Fluss (Feld (Kanten-)Fluss_) durch die Kante größer als 0 ist und sonst unendlich (für Fluss_ gleich 0),was durch einen Wert von 1 E32 dargestellt wird.Bei ganzahliger Rechnung müssen die Schranken unterhalb von 10000 liegen.
Nachdem der Fluss mit minimalen Kosten gefunden worden ist,wird der ursprüngliche Graph wiederhergestellt und die Rückwärts-

kanten werden eliminiert.
Der Parameter Maximal vom Typ Boolean bestimmt (Maximal=true),ob statt eines Flusses mit minimalen Kosten ein Fluss mit maximalen Kosten gesucht werden soll.Art vom Datentyp char paßt die Ausführung der Methode an die vier Anwendungen Mininimale Kosten (Art='m'),Optimales Matchimg (Art='o'),Transportproblem (Art='t') oder Chinesischer Briefträger (Art='b') an,indem, bei den letzten drei Anwendungen,der Quellen-und der Senkenknoten,die zusätzlich erzeugt wurden,wieder gelöscht werden.Beim Briefträgerproblem werden die Gesamtkosten als unproduktive Weglängen bezeichnet.Die Ergebnisse der Berechnungen,nämlich die Kanten unter Angabe des Inhalts bzw. Namens (Wert) von Anfangs-und Endknoten,der (Kanten-) Fluss_,die Kosten des Flusses zu diesen Kanten (Gesamtkosten und pro Flusseinheit/Kosten- und Kostenwert_) und die (Kanten-)Schranke_n werden als Strings in der Liste Sliste vom Typ TStringlist gespeichert und als Referenzparameter zurückgegeben.Außerdem wird zu dieser Liste als Eintrag noch zusätzlich der Gesamtfluss und der vorgegebene Fluss hinzugefügt.
Kommentare zu den Algorithmusschritten werden im Demo-Modus im Label Ausgabe1 vom Typ TLabel ausgegeben.Auf der Objektzeichenfläche Flaeche vom Typ TCanvas werden im Demomodus die gefundenen minimalen Kostenpfade markiert dargestellt,und allgemein die Kanten unter Angabe des errechneten (minimalen Kosten-)Flusses oder unter Angabe der Kosten bzw. modifizierten Kosten oder der Schranken gezeichnet.Oberflaeche vom Typ TForm ist die Form,innerhalb derer der Ursprungsgraph G vom Typ TInhaltsgraph oder ein von ihm durch Vererbung abgeleiteter Graph gezeichnet werden.

<u>Beschreibung der Unit UKnoten:</u>

Die Unit definiert den Datentyp TKnotenform,der sich durch Vererbung von dem in Delphi vordefinierten Datentyp TForm ableitet.Die Variable Knotenform vom Typ TKnotenform wird in dieser Unit global definiert und stellt durch seine Ereignismethoden den grundlegenden Teil der Benutzeroberfläche des Programms Knotengraph dar,die durch visuelle Vererbung im Hauptformular Knotenformular vom Typ TKnotenformular (Unit UForm) wirksam wird.

TKnotengraph hat folgende Datenfelder:

```
PaintBox: TPaintBox;
Panel: TPanel;
Button: TButton;
ListBox: TListBox;
Image: TImage;
Ausgabe1: TLabel;
```

```
Ausgabe2: TLabel;
Eingabe: TEdit;
OpenDialog: TOpenDialog;
SaveDialog: TSaveDialog;
PrintDialog: TPrintDialog;
BitBtn1: TBitBtn;
BitBtn2: TBitBtn;
BitBtn3: TBitBtn;
BitBtn4: TBitBtn;
BitBtn5: TBitBtn;
BitBtn6: TBitBtn;
BitBtn7: TBitBtn;
BitBtn8: TBitBtn;
BitBtn9: TBitBtn;
BitBtn10: TBitBtn;
BitBtn11: TBitBtn;
MainMenu: TMainMenu;
Datei: TMenuItem;
NeueKnoten: TMenuItem;
Graphladen: TMenuItem;
Graphhinzufgen: TMenuItem;
Graphspeichern: TMenuItem;
Graphspeichernunter: TMenuItem;
Graphdrucken: TMenuItem;
Quit: TMenuItem;
Bild: TMenuItem;
Ergebniszeichnen: TMenuItem;
Bildzeichnen: TMenuItem;
Bildkopieren: TMenuItem;
Bildwiederherstellen: TMenuItem;
UngerichteteKanten: TMenuItem;
Knotenradius: TMenuItem;
GenauigkeitKnoten: TMenuItem;
GenauigkeitKanten: TMenuItem;
Knoten: TMenuItem;
Knotenerzeugen: TMenuItem;
Knotenloeschen: TMenuItem;
Knoteneditieren: TMenuItem;
Knotenverschieben: TMenuItem;
Kanten: TMenuItem;
Kantenerzeugen: TMenuItem;
Kantenloeschen: TMenuItem;
Kanteeditieren: TMenuItem;
Kanteverschieben: TMenuItem;
Eigenschaften: TMenuItem;
AnzahlKantenundEcken: TMenuItem;
ParallelkantenundSchlingen: TMenuItem;
Eulerlinie: TMenuItem;
Kreise: TMenuItem;
AnzahlBruecken: TMenuItem;
```

```
Knotenanzeigen: TMenuItem;
Kantenanzeigen: TMenuItem;
Ausgabe: TMenuItem;
Ausgabefenster: TMenuItem;
Abbruch: TMenuItem;
Demo: TMenuItem;
Pausenzeit: TMenuItem;
Hilfe: TMenuItem;
Inhalt: TMenuItem;
Info: TMenuItem;
private
Aktiv_:Boolean;
Graph_:TInhaltsgraph;
GraphH_:TInhaltsgraph;
GraphK_:TInhaltsgraph;
GraphZ_:TInhaltsgraph;
```

Der Datentyp TKnotenform enthält (als Datenfeld bzw. Oberflächensymbol) eine Paintbox vom Typ TPaintbox (oberer, großer Teil der Knotenform), auf deren Oberfläche der Graph dargestellt wird und einen Panel vom Typ TPanel (der untere kleinere Teil der Knotenform),der die Label Ausgabe1 und Ausgabe2 zur Darstellung von Ergebnistexten,ein Eingabefenster Eingabe vom Typ TEdit zur Eingabe von Texten sowie einen Schalter Button vom Typ TButton zur Auslösung einer Aktion beinhaltet (beide Elemente werden nur bei der Anwendung Endlicher Automat benutzt).Image vom Typ TImage nimmt den Graphen als Bild auf,um ihn in die Zwischenablage von Windows kopieren.Listbox vom Typ TListbox dient dazu,um Ergebnisse in Form von Strings oberhalb des Panels am unteren Ende der Paintbox anzuzeigen.Die Listbox kann dabei entweder aus einer Zeile bestehen (mit Scrollmöglichkeit) oder aber die halbe bzw. (fast) ganze Paintbox ausfüllen.Von der Listbox aus werden die Ergebnisse in das Gitternetz vom Typ TStringGrid der Ausgabeform übernommen.Opendialog und Savedialog vom Datentyp TOpendialog bzw. TSavedialog sind die Standard-Dialogfelder für das Laden und Speichern von Dateien (hier: Graphendateien mit der Extention gra).Printdialog vom Typ TPrintdialog ist das Standarddialogelemet zur Auswahl eines Druckers und die Aktionsschalterelemente BitBtn (1 bis 11) von Typ TBitBtn sind Aktions-Schalter,um mittels Mausklick auf diese Flächen die Algorithmen zur Erzeugung,Verwaltung bzw. der Veränderung des Graphen vereinfacht starten zu können.

Mainmenu ist das Hauptmenü vom Typ TMainmenü und enthält folgende Untermenüs vom Typ TMenuItem:
Datei ,NeueKnoten ,Graphladen ,Graphhinzufgen , Graphspeichern, Graphspeichernunter, Graphdrucken, Quit, Bild, Ergebniszeichnen, Bildzeichnen , Bildkopieren ,Bildwiederherstellen, Ungerichtete Kanten, Knotenradius, GenauigkeitKnoten, GenauigkeitKanten, Knoten ,Knotenerzeugen, Knotenloeschen ,Knoteneditieren, Knoten-

verschieben, Kanten, Kantenerzeugen ,Kantenloeschen ,Kanteeditieren ,Kanteverschieben, Eigenschaften , AnzahlKantenundEcken , ParallelkantenundSchlingen, Eulerlinien, Kreise, AnzahlBruecken , Knotenanzeigen , Kantenanzeigen, Ausgabe, Ausgabefenster, Abbruch, Demo, Pausenzeit ,Hilfe ,Inhalt und Info .

Weitere Felder sind Graph_,GraphH_ und GraphK_ vom Typ TInhaltsgraph (Unit UInhgrph).Das Feld Graph_ speichert den Graphen vom Typ TInhaltsgraph, der durch den Benutzer durch die Menüs „Einfügen“, „Editieren“, „Löschen“ und „Verschieben“ von Knoten und Kanten sowie „Neuer Knoten“,„Graph laden“.,„Graph speichern“,„Graph speichern unter“, „Graph hinzufügen“ erzeugt und verändert wird.Das Feld GraphH_ nimmt die Ergebnisgraphen (mittels Typkonversion als Vorgängerobjekt vom Typ TInhaltsgraph von neu als Nachfolgerobjekt des Graphen Graph_ definierten Graphentypen) der (mathematischen) Anwendungen und der Pfade auf, und dieser Graph GraphH_ wird beim Aufruf des Menüpunkt „Ergebnis zeichnen“ dargestellt.Das Boolesche Feld Aktiv_ mit der Property Aktiv von TKnotenform bzw. Knotenform bestimmt,ob Graph_ (Aktiv=true) oder GraphH_ gezeichnet wird.Der Graph GraphK_ vom Typ TInhaltsgraph dient zur Zwischenspeicherung des Graphen Graph_,so dass nach einer Änderung des Graphen mittels der zur Manipulation des Graphen Graph_ genannten Menüalgorithmen,der ursprüngliche Graph wiederhergestellt werden kann.Das Feld GraphZ_ dient zur Aufnahme eines der Graphen Graph_ oder GraphH_ ,um damit verschiedene Ereignismethoden flexibel einsetzen zu können.

Der Datentyp enthält folgende Propertys,um über sie auf die oben genannten deklarierten (private) Datenfelder zuzugreifen.

Property Aktiv:boolean read Istaktiv write setzeaktiv
Property Graph:TInhaltsgraph read WelcherGraph write SetzeGraphG
Property GraphH:TInhaltsgraph read WelcherGraphH write SetzeGraphH
Property GraphK:TInhaltsgraph read WelcherGraphK write SetzeGraphK
Property GraphZ:TInhaltsgraph read WelcherGraphZ write SetzeGraphZ

Durch die folgenden Methoden von TMinimaleKostenGraph (Proceduren und Funktionen) wird auf die Propertys lesend und schreibend zugegriffen:

Function TKnotenform.Istaktiv:Boolean
Procedure TKnotenform.Setzeaktiv(A:Boolean)
Function TKnotenform.WelcherGraphG:TInhaltsgraph
Procedure TKnotenform.SetzeGraphG(Gr:TInhaltsgraph)
Function TKnotenform.WelcherGraphH:TInhaltsgraph

Procedure TKnotenform.SetzeGraphH(Gr:TInhaltsgraph)
Function TKnotenform.WelcherGraphK:TInhaltsgraph
Procedure TKnotenform.SetzeGraphK(Gr:TInhaltsgraph)
Function TKnotenform.WelcherGraphZ:TInhaltsgraph
Procedure TKnotenform.SetzeGraphZ(Gr:TInhaltsgraph)
Durch die Benutzung der Propertys wird auf das entsprechende Datenfeld nur mit Hilfe von Methoden zugegriffen.

Die Methoden von TKnotenform lassen sich unterteilen in allgemeine Methoden,Ereignismethoden der Komponenten,Ersatz-Ereignismethoden und Menu-Ereignismethoden.Dieser Einteilung folgt die nachstehende Beschreibung.

Methoden von TKnotenform:

Bedeutung des Parameters Sender:

In einer Ereignisbehandlungsroutine informiert der Parameter Sender darüber, welche Komponente das Ereignis empfangen und infolgedessen die Behandlungsroutine aufgerufen hat.

Allgemeine Methoden:

Procedure TKnotenFormBildloeschen

Diese Methode löscht das Bild des Graphen in der Paintbox.

Procedure TKnotenForm.StringlistnachListbox(S:TStringlist;var L:TListbox)

Diese Methode kopiert die (String-) Einträge der Stringliste S vom Typ TStringlist in die Listbox L vom Typ TListbox.

Procedure TKnotenForm.Menuenabled(Enabled:Boolean)

Diese Methode setzt die Eigenschaft (Property) Enabled bei allen (Unter-)Menüs außer bei den Untermenüs von Ausgabe (der Wert von Enabled dieser Untermenus bleibt auf true gesetzt) sowie bei den Aktionsschalterelementen BitBtn (1 bis 11) auf den Wert des Parameters Enabled vom Typ Boolean.

Procedure TKnotenForm.AusgabeLoeschen(Onlyfirst:Boolean)

Diese Methode löscht bei Onlyfirst=true den Inhalt des ersten Ausgabelabels und bei Onlyfirst=false den Inhalt beider Ausgabelabels.

Procedure TKnotenform.Fehler

Diese Methode erzeugt ein Ausgabefenster mit dem Text Fehler

und setzt bei allen Menüs die Eigenschaft Enabled auf true.Sie wird im Exception-Teil der Ereignismethoden verwendet.

Procedure TKnotenform.WMMenuselect(var Message:TWMMenuselect)

Diese Ereignismethode wird aufgerufen,wenn ein beliebiges Menü oder Untermenü angewählt wird.Die Ereignismethoden Paintbox.OnMouseDown, Paintbox.OnDblClick, Paintbox.OnMousemove werden auf ihre Standardwerte zurückgesetzt und der Visible-Eigenschaft von Listbox wird false zugewiesen,außer wenn das Menu Ausgabefenster angewählt wurde.Die Label Ausgabe1 und Ausgabe2 wird der Leerstring zugewiesen.Das Feld Aktiv_ wird auf true gesetzt (aktiver Graph Graph).Wenn auf der Variablen GraphH ein Graph gespeichert ist,wird dessen Zustand auf false gesetzt, und der letzte Mausklickknoten wird gelöscht.Der Graph GraphH wird aus dem Speicher entfernt,und der Variablen der Wert nil zugewiesen,sobald ein beliebiges Untermenü außer den Untermenüs der Hauptmenüs Pfade und Anwendungen angewählt wird.Wenn kein Untermenü von Menü Ausgabe angewählt wurde,wird das Feld Abbruch_ der Variablen Graph auf false gesetzt.Der Graph GraphK wird aus dem Speicher entfernt,und ein neuer leerer Graph wird erzeugt,wenn eines der Untermenüs der Hauptmenüs Pfade und Anwendungen gewählt wurde(kein letzter Graph gespeichert).

Ereignismethoden der Komponenten:

.

Procedure TKnotenForm.FormActivate(Sender: TObject)

Diese Methode erzeugt die Kantenform vom Typ TKantenform und die Graphen Graph und GraphH vom Typ TInhaltsgraph,setzt das Feld Aktiv_ auf true,weist den Namen des Help-Files sowie der Listbox die richtige Breite zu.Falls das Programm Knotengraph mit einem Dateinamen als Parameter aufgerufen wird,wird dieser Graph geladen und gezeichnet.

Procedure TKnotenform.PaintBoxPaint(Sender: TObject)

Diese Ereignismethode wird aufgerufen,wenn sich Größe und Lage von der Form Knotenform ändern.Dann wird,wenn das Feld Aktiv_ true ist,der Graph Graph neu gezeichnet,anderfalls der Graph GraphH.Die Größen von Listbox und Paintbox werden neu angepasst,ebenso die Größe der Scrollleisten.

Procedure TKnotenform.PaintBoxMouseDown(Sender:TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

Diese Ereignismethode wird bei verschiedenen Anwendungen durch

eine Ersatz-Ereignismethoden ersetzt,um andere Aktionen mit Hilfe von Mausklicks ausführen zu können.Diese Methode wird aufgerufen,wenn über der Fläche der Paintbox eine Maustaste gedrückt wird.Wenn sich an der Mausklickposition der linken Maustaste ein Knoten oder eine Kante befindet,wird der Inhalt des Knoten oder der Kante mittels eines Fensters ausgegeben.Der Knoten wird dann als letzter Mausklickknoten gespeichert.Wenn die rechte Maustaste gedrückt wird,wird die Methode zum Erzeugen einer neuen Kante KanteerzeugenMouseDown aufgerufen und diese Ereignismethode dadurch ersetzt.Außerdem wird der GraphK aus dem Speicher gelöscht und ein neuer leerer GarphK erzeugt.Button vom Typ TMouseButton übergibt die linke,mittlere oder rechte gedrückte Maustaste,Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten (wird von der Methode nicht verwendet). X und Y sind die Koordinaten der Mausklickstelle der Paintbox.Der Graph Graph wird neu gezeichnet.

Procedure TKnotenform.PaintBoxDblClick(Sender:TObject)

Diese Ereignismethode wird aufgerufen,wenn mit der linken Maustaste ein Doppelklick auf die Fläche der Paintbox ausgeübt wird.Dann wird die Ereignismethode PaintboxMouseDown durch die Ereignismethode KnotenerzeugenMouseDown ersetzt und dadurch die Aktion Erzeugen eines neuen Knotens eingeleitet.Der Graph GraphK wird aus dem Speicher entfernt,und auf der Variablen GraphK wird der momentan vorhandene aktuelle Graph als Kopie neu gespeichert.

Procedure TKnotenform.PaintBoxMouseMove(Sender:TObject;
Shift:TShiftState;X,Y:Integer)

Diese Ereignismethode wird aufgerufen,wenn der Mauszeiger über die Fläche der Paintbox (mittels gedrückter Maustaste) gezogen wird.Geschieht dies mit der linken Maustaste,wird entweder ein Knoten oder eine Kante verschoben,je nachdem in der Nähe welches Objektes sich der Mauszeiger zu Beginn der Aktion befand.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.X und Y sind die Koordinaten der Klickstelle der Maus in der Paintbox.

Procedure TKnotenform.PanelClick(Sender:TObject)

Diese Ereignismethode reagiert auf einen Mausklick auf die Panelfläche und öffnet (zeigt) die Listbox,falls sie (nichleere) Einträge enthält.Die Listbox ist dann mit einer Zeile (evtl. mit zusätzlicher Scrolleiste,falls eine Zeile für die Darstellung der Daten nicht ausreicht) mit den anzuzeigenden Daten direkt oberhalb der Panelfläche am unteren Rand der Paintbox sichtbar.

Procedure TKnotenform.PanelDblClick(Sender:TObject)

Diese Ereignismethode reagiert auf einen Doppel-Mausklick auf die Panelfläche und öffnet (zeigt) die Listbox,falls diese (nichtleere) Einträge enthält.Die Listbox ist dann mit mit der Größe von ca. 1/3 der Formgröße (evtl. mit zusätzlicher Scrolleiste,falls eine Zeile für die Darstellung der Daten nicht ausreicht) mit den anzuzeigenden Daten direkt oberhalb der Panelfläche in der Paintbox sichtbar.

Procedure TKnotenform.PanelMouseDown(Sender:TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

Diese Ereignismethode reagiert auf das Drücken einer Maustaste über der Panel-Zeichenfläche.Wenn die rechte Maustaste gedrückt wurde,wird die Eigenschaft Visible der Listbox auf false (Schließen der Listbox) gesetzt.Wenn die linke Maustaste und zusätzlich die Strg-Taste gedrückt wird,wird der Inhalt des letzten mit der Maus angewählte Knoten (letzter Mausklickknoten) in einem Fenster angezeigt.Bei der Shift-Taste wird der Einzelschrittmodus gestartet und bei der Alt-Taste wird ein Einzelschritt ausgelöst.Shift und Alt zusammen beenden den Enzelschrittmodus.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.X und Y sind die Koordinaten der Mausklickstelle.

Procedure TKnotenform.Ausgabe1Click(Sender:TObject)

Diese Ereignismethode reagiert auf einen Doppel-Mausklick auf die Ausgabe1fläche und öffnet (zeigt) die Listbox,falls diese (nichtleere) Einträge enthält.Die Listbox ist dann mit einer Zeile (evtl. mit zusätzlicher Scrolleiste,falls eine Zeile für die Darstellung der Daten nicht ausreicht) mit den anzuzeigenden Daten direkt oberhalb der Panelfläche am unteren Ende der Paintbox sichtbar.

Procedure TKnotenform.Ausgabe1DblClick(Sender:TObject)

Diese Ereignismethode reagiert auf einen Doppel-Mausklick auf die Ausgabe1-Fläche und öffnet (zeigt) die Listbox,falls sie (nichtleere) Einträge enthält.Die Listbox ist dann in der Paintboxgröße (evtl. mit zusätzlicher Scrolleiste,falls eine Zeile für die Darstellung der Daten nicht ausreicht) mit den anzuzeigenden Daten direkt oberhalb der Panelfläche in der Paintbox sichtbar.

Procedure TKnotenform.Ausgabe1MouseDown(Sender:TObject;
Button: TMouseButton; Shift: TShiftState;X,Y:Integer)

Diese Ereignismethode reagiert auf das Drücken einer Maustaste über der Ausgabe1-Fläche.Wenn die rechte Maustaste gedrückt

wurde,wird die Eigenschaft Visible der Listbox auf false (Schließen der Listbox) gesetzt.Wenn die linke Maustaste und zusätzlich die Shift-Taste gedrückt wird,wird der Inhalt des letzten mit der Maus angewählte Knoten (letzter Mausklickknoten) in einem Fenster angezeigt. Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.X und Y sind die Koordinaten der Stelle des Mausklicks in der Paintbox.

Procedure TKnotenform.Ausgabe2Click(Sender:TObject)

Diese Ereignismethode reagiert auf einen Mausklick auf die Ausgabe2-Label—Fläche und öffnet (zeigt) die Listbox,falls sie (nichtleere) Einträge enthält.Die Listbox ist dann mit einer Zeile (evtl. mit zusätzlicher Scrolleiste,falls eine Zeile für die Darstellung der Daten nicht ausreicht) mit den anzuzeigenden Daten direkt oberhalb der Panelfläche am unteren Ende der Paintbox sichtbar.

Procedure TKnotenform.Ausgabe2DblClick(Sender:TObject)

Diese Ereignismethode reagiert auf einen Doppel-Mausklick (linke Maustaste) auf die Ausgabe2-Label-Fläche und öffnet (zeigt) die Listbox,falls diese (nichtleere) Einträge enthält.Die Listbox ist dann mit mit der Größe von ca. 1/2 Formgröße (evtl. mit zusätzlicher Scrolleiste,falls eine Zeile für die Darstellung der Daten nicht ausreicht) mit den anzuzeigenden Daten direkt oberhalb der Panelfläche in der Paintbox sichtbar.

Procedure TKnotenform.Ausgabe2MouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState;X,Y:Integer)

Diese Ereignismethode reagiert auf das Drücken einer Maustaste über der Ausgabe2-LAabel-Fläche.Wenn die rechte Maustaste gedrückt wurde,wird die Eigenschaft Visible der Listbox auf false (Schließen der Listbox) gesetzt.Wenn die linke Maustaste und zusätzlich die Strg-Taste gedrückt wird,wird der Inhalt des letzten mit der Maus angewählte Knoten (letzter Mausklickknoten) in einem Fenster angezeigt.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.X und Y sind die Koordinaten der Mausklickstelle.

Procedure TKnotenform.FormCloseQuery(Sender:TObject;var CanClose:Boolean)

Diese Ereignismethode reagiert auf das Schließen und Beenden von Knotenform.Falls Änderungen des Graphen nicht gespeichert sind,wird das Speichern des Graphen vorgeschlagen und kann angewählt werden.Außerdem erfolgt eine Sicherheitsabfrage,ob das Programm wirklich beendet werden sollte.Das Feld Abbruch_ von GraphH wird auf false gesetzt.Die Variable Canclose bestimmt,ob

das Programm wirklich geschlossen werden darf.Die Graphen Graph, GraphH und GraphK werden aus dem Speicher entfernt.Anschließend wird Knotenform selber aus dem Speicher entfernt.

Ersatz-Ereignismethoden:

Die im folgenden genanten Knoten und Kanten sind im Graph Knotenform.Graph enthalten und das Wort Graph bezieht sich auf den Graph Knotenform.Graph.

Procedure TKnotenform.KnotenerzeugenMouseDown(Sender: TObject; Button: TMouseButton;Shift:TShiftState;X,Y:Integer)

Diese Ereignismethode kann (mittels der Anweisung Paintbox.On-MouseDown:= KnotenerzeugenMouseDown) die Ereignismethode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt auf der Paintboxfläche mit den Koordinaten X und Y wird dann ein neuer Knoten erzeugt und in den Graphen Graph eingefügt.Der Knoten-inhalt wird durch ein Eingabefenster abgefragt und eingegeben.Der erweiterte Graph wird neu gezeichnet,und es wird durch die Methode selber mittels der Anweisung Paintbox.OnMouseDown:= PaintboxMouseDown auf die Standard-ereignismethode der Paintbox zurückgeschaltet.Der Parameter But-ton übergibt die gedrückte Maustaste.Shift übermittelt den Sta-tus der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenform.KnotenloeschenMouseDown(Sender: TObject; Button: TMouseButton;Shift:TShiftState;X,Y:Integer)

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:= KnotenloeschenMouseDown) die Ereignis-methode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt auf der Paintboxfläche mit den Koordinaten X und Y wird dann,falls sich dort ein Knoten des Graphen befindet,dieser Kno-ten (einschließlich aller Kanten,deren Anfangs-oder Endknoten dieser Knoten ist) aus dem Graph entfernt.Andernfalls wird eine entsprechende Fehlermitteilung ausgegeben.Der reduzierte Graph wird neu gezeichnet, und es wird durch die Methode selber mit-tels Paintbox.OnMouseDown:= PaintboxMouseDown auf die Standard-ereignismethode der Paintbox zurückgeschaltet. Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maus-tasten.

Procedure TKnotenform.KnoteneditierenMouseDown(Sender: TObject; Button: TMouseButton;Shift:TShiftState;X,Y:Integer)

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:= KnoteneditierenMouseDown) die Ereignis-methode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt

auf der Panelfläche mit den Koordinaten X und Y kann dann,falls sich dort ein Knoten des Graphen befindet,der Inhalt dieses Knotens in einem Editierfenster abgeändert werden (oder beibehalten werden).Andernfalls (falls sich beim Punkt mit den Koordinaten X und Y kein Knoten befindet) wird eine entsprechende Fehlermitteilung ausgegeben.Der veränderte Graph wird neu gezeichnet,und es wird durch die Methode selber mittels Paintbox.OnMouseDown:= PaintboxMouseDown auf die Standardereignismethode der Paintbox zurückgeschaltet.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenform.KnotenverschiebenaufMouseup (Sender: TObject;Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseup:= KnotenverschiebenaufMouseup) die Ereignismethode PaintboxMouseup ersetzen.Beim Loslassen der Maustaste oberhalb der Paintboxfläche wird dann,falls sich dort ein Knoten des Graphen befindet,das Datenfeld Besucht_ dieses Knotens auf false gesetzt.Es wird dann durch die Methode selbst mittels Paintbox.OnMouseup:=nil die Reaktion auf das Ereignis Mausetaste loslassen wieder abgeschaltet,und es wird ebenfalls durch die Methode selbst mittels Paintbox.OnMouseDown:=PaintboxMouseDown auf die Standardereignismethode der Paintbox zurückgeschaltet (durch Aufruf von KnotenverschiebenabMousedown).Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenform.KnotenverschiebenabMouseDown(Sender: TObject;Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMousedown:= KnotenverschiebenabMousedown) die Ereignismethode PaintboxMousedown ersetzen.Beim Drücken der Maustaste auf den Punkt mit den Koordinaten X und Y der Paintboxfläche wird dann,falls sich dort ein Knoten des Graphen befindet,das Datenfeld Besucht_ dieses Knotens auf true gesetzt (und bei allen anderen Knoten des Graphen auf false).Der Inhalt des ausgewählten Knoten wird im Label Ausgabe1 angezeigt.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.Nach Abschluss dieser Methode sollte ein Aufruf Paintbox.OnMousedown:=KnotenverschiebenabMousedown erfolgen.

Procedure TKnotenform.KanteerzeugenMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:= KanteerzeugenMouseDown) die Ereignismethode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt auf der Paintboxfläche mit den Koordinaten X und Y wird dann,falls sich dort ein Knoten befindet,dieser Knoten als erster Knoten für die einzufügende Kante ausgewählt.Die Kanteneigenschaften (Inhalt,Weite,Richtung,Art des Inhalts (Real,Integer,String) werden durch ein Eingabefenster (Kanteform) abgefragt und eingegeben.Durch einen erneuten Mausklick auf einen Punkt mit den Koordinaten X und Y wird ein zweiter Knoten ausgewählt,der sich ebenfalls in der Nähe dieses Punktes der Paintboxfläche befindet.Mit dem ersten und zweiten Knoten als Anfangs- und Endknoten (oder umgekehrte Reihenfolge) wird eine Kante mit den vorgegebenen Eigenschaften erzeugt und in den Graph eingefügt.Falls sich kein Knoten in der Nähe der beiden Punkte mit den Koordinaten X und Y befindet,wird eine entsprechende Mitteilung ausgegeben und keine Kante eingefügt,ebenso,falls im Eingabefenster Abbruch gewählt wurde.Der evtl. veränderte Graph wird neu gezeichnet,und es wird (nach Einfügen einer neuen Kante oder bei Abbruch oder Fehlermeldung) durch die Methode selber mittels Paintbox.OnMouseDown:=PaintboxMouseDown auf die Standardereignismethode der Paintbox zurückgeschaltet.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenform.KanteloeschenMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:= KanteloeschenMouseDown) die Ereignismethode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt auf der Paintboxfläche mit den Koordinaten X und Y wird dann,falls sich dort eine Kante des Graphen befindet,diese Kante aus dem Graph entfernt.Andernfalls wird eine entsprechende Fehlermitteilung ausgegeben.Der reduzierte Graph wird neu gezeichnet,und es wird durch die Methode selber mittels Paintbox.OnMouseDown:=PaintboxMouseDown auf die Standardereignismethode der Paintbox zurückgeschaltet.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenform.KanteeditierenMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:= KanteeditierenMouseDown) die Ereignismethode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt auf der Panelfläche mit den Koordinaten X und Y können

dann,falls sich dort eine Kante des Graphen befindet,die Eigenschaften dieser Kante (Inhalt,Weite,Richtung,Art des Inhalts (Real,Integer,String) in einem Editierfenster (Kantenform) abgeändert werden (oder beibehalten werden,z.B. falls Abbruch gewählt wurde).Andernfalls (falls sich beim Punkt mit den Koordinaten X und Y keine Kante befindet) wird eine entsprechende Fehlermitteilung ausgegeben.Der veränderte Graph wird neu gezeichnet,und es wird durch die Methode selber mittels Paintbox.OnMouseDown:= PaintboxMouseDown auf die Standardereignismethode der Paintbox zurückgeschaltet.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenform.KanteverschiebenaufMouseUp(Sender: TObject;Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseup:= KanteverschiebenaufMouseup)die Ereignismethode PaintboxMouseup ersetzen.Beim Loslassen der Maustaste oberhalb der Paintboxfläche wird dann das Datenfeld Besucht_ aller Kanten des Graphen auf false gesetzt.Es wird dann durch die Methode selber mittels Paintbox.OnMouseup:=nil die Reaktion auf das Ereignis Mausetaste loslassen wieder abgeschaltet,und es wird durch die Methode selber mittels Paintbox.OnMouseDown:=PaintboxMouseDown auf die Standardereignismethode der Paintbox zurückgeschaltet (durch Aufruf von KnotenverschiebenabMousedown).Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenform.KanteverschiebenabMouseDown(Sender: TObject;Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMousedown:= KanteverschiebenabMousedown) die Ereignismethode PaintboxMousedown ersetzen.Beim Drücken der Maustaste auf den Punkt mit den Koordinaten X und Y oberhalb der Panelfläche wird dann,falls sich dort eine Kante des Graphen befindet,das Datenfeld Besucht_ dieser Kante auf true gesetzt (und bei allen anderen Kanten des Graphen auf false).Der Inhalt der ausgewählten Kante wird im Label Ausgabe1 angezeigt.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.Nach Abschluss dieser Methode sollte ein Aufruf Paintbox.OnMousedown:= KanteverschiebenabMousedown erfolgen.

Menü-Ereignismethoden:

Menü Datei:

Procedure TKnotenform.NeueKnotenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Neue Knoten mit der Maus angewählt wird.Wenn Änderungen im momentanen Graph Graph noch nicht gespeichert wurden,wird zunächst abgefragt,ob der Graph gespeichert werden soll und,wenn gewünscht,wird ein entsprechendes Dateiauswahlfenster zum Speichern angezeigt.Dann werden alle Knoten und Kanten im vorhandenen Graph Graph gelöscht und ein leerer Graph erzeugt.Das Bild des Graphen wird gelöscht,der letzte Mausklickknoten (der zuletzt mit der Maus angewählte Knoten) wird auf nil gesetzt,der Graph GraphH wird aus dem Speicher gelöscht,und die Variable GraphH auf nil sowie die Kanten- und Knotengenauigkeit auf 3 gesetzt (Anzeige der Inhalte mit 3 Stellen nach dem Komma).

Procedure TKnotenform.GraphladenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Graph laden mit der Maus angewählt wird.Wenn Änderungen im momentanen Graph Graph noch nicht gespeichert wurden,wird zunächst abgefragt,ob der Graph gespeichert werden soll und,wenn gewünscht,wird ein entsprechendes Dateiauswahlfenster zum Speichern angezeigt.Alle Knoten und Kanten im vorhandenen Graph Graph werden gelöscht,und ein leerer Graph wird erzeugt.Das Bild des Graphen wird gelöscht,der letzte Mausklickknoten (der zuletzt mit der Maus angewählte Knoten) wird auf nil gesetzt,der Graph GraphH wird aus dem Speicher gelöscht,und die Variable GraphH wird auf nil gesetzt.Ein Dateiauswahlfenster wird geöffnet,und nach Auswahl einer Datei wird dieser Graph geladen und neu gezeichnet.Bei Auswahl von Abbrechen bleibt der vorhandene Graph erhalten und wird neu gezeichnet.

Procedure TKnotenform.GraphhinzufgenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Graph hinzufügen mit der Maus angewählt wird.Wenn Änderungen im momentanen Graph Graph noch nicht gespeichert wurden,wird zunächst abgefragt,ob der Graph gespeichert werden soll und,wenn gewünscht ,wird ein entsprechendes Dateiauswahlfenster zum Speichern angezeigt.Alle Knoten und Kanten im vorhandenen Graph Graph werden gelöscht,und es wird ein leerer Graph erzeugt.Das Bild des Graphen wird gelöscht,der letzte MausklickKnoten (der zuletzt mit der Maus angewählte Knoten) wird auf nil gesetzt,der Graph GraphH wird aus dem Speicher gelöscht und die Variable GraphH auf nil gesetzt.Ein Dateiauswahlfenster wird geöffnet,und nach Auswahl einer geeigneten Datei wird deren Graph zu dem bestehenden Graph hinzugefügt und neu gezeichnet.Bei Auswahl von Abbrechen bleibt der vorhandene Graph erhalten und wird neu gezeichnet.

Procedure TKnotenform.GraphspeichernClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Graph speichern mit der Maus angewählt wird.Wenn schon ein Dateiname zum Graphen gehört (d.h.. der Graph schon vorher unter diesem Namen geladen wurde),wird der momentane Graph unter diesem Namen gespeichert.Ansonsten wird die Methode GraphspeichernunternClick aufgerufen.

Procedure TKnotenform.GraphspeichernunterClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Graph speichern unter mit der Maus angewählt wird.Ein Dateiauswahlfenster wird angezeigt,und entweder kann ein vorhander Dateiname ausgewählt werden oder aber ein neuer Name angegeben werden.Die Standardextension für die zu speichernde Datei ist .gra.Bei Auswahl von OK wird der Graph abgespeichert,bei Auswahl von Abbrechen nicht abgespeichert.

Procedure TKnotenform.GraphdruckenClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Graph drukken mit der Maus angewählt wird.Das Bild des Graphen,das auf der Paintboxfläche gezeichnet ist,wird vergrößert auf dem unter Windows eingestellten Drucker ausgedruckt.Zuvor kann ein Drucker in einem Druckerauswahlfenster gewählt werden.

Procedure TKnotenform.QuitClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Quit mit der Maus angewählt wird.Zunächst wird die Methode FormCloseQuery ausgeführt.Falls es diese Methode zuläßt,wird Knotenform geschlossen und das Programm beendet.

Menü Bild:

Procedure TKnotenform.ErgebniszeichnenClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Ergebnis zeichnen mit der Maus angewählt wird.Wenn GraphH nicht nil ist (GraphH nimmt den Ergebnisgraph auf),wird das vorhandene Bild des Graphen auf der Paintboxfläche gelöscht und der Graph GraphH wird gezeichnet.Das Feld Aktiv_ wird auf false gesetzt.

Procedure TKnotenform.BildzeichnenClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Bild zeichnen mit der Maus angewählt wird.Das vorhandene Bild des Graphen (evtl. des Graphen GraphH) wird gelöscht,und der Graph Graph

gezeichnet.Paintbox.OnMouseDown wird auf die Ereignismethode PaintboxMouseDown gesetzt und das Feld Aktiv_ auf true.
Procedure TKnotenform.BildkopierenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Bild kopieren mit der Maus angewählt wird.Nach einer entsprechenden Benutzerauswahl in einem Abfragefenster (erscheint nur,wenn GraphH ungleich nil ist) wird das Bild des Graphen Graph oder des Graphen GraphH (nur für GraphH ungleich nil) vergrößert (mittels der Komponente Image vom Typ TImage) in die Zwischenablage von Windows kopiert.

Procedure TKnotenform.BildwiederherstellenClick(Sender:TObject)

Diese Methode macht die letzten Änderungen der Algorithmen der Menüs von Knoten und Kanten sowie Neue Knoten,Graph laden und Graph hinzufügen,Ungerichtete Kanten,Knotenradius wieder rückgängig und stellt den vorigen Graph,der in GraphK gespeichert ist,(für GraphK ungleich nil und nicht leer) wieder her.

Procedure TKnotenform.UngerichteteKantenClick(Sender:TObject)

Diese Methode erzeugt aus dem vorliegenden Graphen einen Graphen mit ungerichteten Kanten,indem jede gerichtete Kante durch eine ungerichtete Kante ersetzt wird.

Procedure TKnotenform.KnotenradiusClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Knotenradius mit der Maus angewählt wird.Ein Eingabefenster erscheint,in dem ein neuer Wert für den Radius eingegeben wird,mit dem die Knoten der Graphen Graph und GraphH gezeichnet werden.Der Radius kann zwischen den Werten 10 bis 100 gewählt werden.Ansonsten wird der Wert nicht geändert.Vorgabewert ist 15.Anschließend kann die Liniedicke,mit der Knoten und Kanten gezeichnet werden,in den drei Stufen 1 bis 3 mittels eines Eingabefensters eingestellt werden.Diese Vorgaben wirken sich auch auf schon gezeichnete Knoten und Kanten aus.

Procedure TKnotenform.GenauigkeitKnotenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Genauigkeit Knoten mit der Maus angewählt wird.Ein Eingabefenster erscheint,in dem ein neuer Wert für die Genauigkeit eingegeben werden kann,mit der der Wert eines Knoten beim Zeichnen der Graphen Graph und GraphH dargestellt wird.Nicht negative Werte (die nicht größer als 7 sein dürfen) für die Stellenzahl bedeuten die anzuzeigende Stellenzahl hinter dem Komma,falls der Wert des Knotens als Zahl aufgefaßt werden kann (ansonsten wird der Wert wie bei der Eingabe vorgegeben dargestellt),negative Werte

bedeuten (als Absolutbetrag aufgefaßt) die Gesamtzahl der Zeichen,mit der als (Knoten-)Wert gespeicherte Strings ausgegeben werden.Gezählt wird von links.Weiter rechts stehende Zeichen werden abgeschnitten.Der voreingestellte Wert für die Genauigkeit der Knoten ist 3.Falls x eingegeben wird,wird der Wert eines Knoten so ausgegeben,wie er als String eingegeben wurde.

Procedure TKnotenform.GenauigkeitKantenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Genauigkeit Kanten mit der Maus angewählt wird.Ein Eingabefenster erscheint,in dem ein neuer Wert für die Genauigkeit eingegeben werden kann,mit der der Wert einer Kante beim Zeichnen der Graphen Graph und GraphH dargestellt wird.Nicht negative Werte (die nicht größer als 7 sein dürfen) für die Stellenzahl bedeuten die anzuzeigende Stellenzahl hinter dem Komma,falls der Wert der Kante als Zahl aufgefaßt werden kann (ansonsten wird der Wert wie bei der Eingabe vorgegeben dargestellt),negative Werte bedeuten (als Absolutbetrag aufgefaßt) die Gesamtzahl der Zeichen,mit der die als (Kanten-)Wert gespeicherten Strings ausgegeben werden.Gezählt wird von links.Weiter rechts stehende Zeichnen werden abgeschnitten.Der voreingestellte Wert für die Genauigkeit der Kanten ist 3.Falls x eingegeben wird,wird Wert einer Kante so ausgegeben,wie er als String eingegeben wurde.

Menü Knoten:

Procedure TKnotenform.KnotenerzeugenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Knoten erzeugen mit der Maus angewählt wird.Bei einem Mausdoppelklick auf einen Punkt der Paintbox-Zeichenfläche wird ein Eingabefenster geöffnet,in dem der Inhalt des Knotens eingegeben werden kann.Anschließend wird ein neuer Knoten mit dem vorgegeben Inhalt erzeugt,in den Graphen Graph eingefügt und an der Stelle des Mausklicks mit seinem Mittelpunkt gezeichnet.

Procedure TKnotenform.KnotenloeschenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Knoten löschen mit der Maus angewählt wird.Bei einem Mausklick auf einen Knoten des Graphen wird dieser Knoten samt allen Kanten,die den zu löschenden Knoten als Anfangs- oder Endknoten haben,aus dem Graph gelöscht und der reduzierte Graph neu gezeichnet.

Procedure TKnotenform.KnoteneditierenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Knoten editieren mit der Maus angewählt wird.Bei einem Mausklick auf einen Knoten wird dieser Knoten angewählt,und es öffnet sich ein

Eingabefenster,das den bisherigen Knoteninhalt enthält.Dieser Inhalt kann nun editiert werden.Beim Mausklick auf OK wird der neue Knoteninhalt übernommen,bei Abbruch bleibt es beim bisherigen Knoteninhalt.

Procedure TKnotenform.KnotenverschiebenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Knoten verschieben mit der Maus angewählt wird.Bei Drücken (und Gedrückt lassen) der linken Maustaste über einem Knoten wird dieser Knoten angewählt und kann mit der Maus an einen anderen Ort verschoben werden.Die Koordinatenwerte X und Y des Knotens ändern sich entsprechend.Der Vorgang wird durch das Loslassen der Maustaste beendet.

Menü Kante:

Procedure TKnotenform.KantenerzeugenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Kanten erzeugen mit der Maus angewählt wird.Mit der linken Maustaste werden durch Mausklick zunächst nacheinander zwei Knoten ausgewählt,zwischen denen die neue Kante eingefügt werden soll.Nach Auswahl des zweiten Knotens erscheint ein Eingabefenster (Kantenform),in das die Eigenschaften der neuen Kante (Inhalt,Weite,Richtung,Art des Inhalts,d.h. Real,Integer,String) eingegeben werden können.Beim Mausklick auf OK,wird die neue Kante erzeugt und eingefügt,bei Abbruch bleibt der bisherige Graph unverändert.

Procedure TKnotenform.KantenloeschenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Kanten löschen mit der Maus angewählt wird.Mit der linken Maustaste wird durch Mausklick die zu löschende Kante ausgewählt.Danach wird die Kante aus dem Graph gelöscht.

Procedure TKnotenform.KanteeditierenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Kante editieren mit der Maus angewählt wird.Mit der linken Maustaste wird durch Mausklick die zu editierende Kante ausgewählt.Danach erscheint ein Eingabefenster (Kantenform),in das die Eigenschaften der neuen Kante (Inhalt,Weite,Richtung,Art des Inhalts,d.h. Real,Integer,String) eingegeben werden können.Beim Mausklicken auf OK,wird die Kante verändert,und der Graph neu gezeichnet.Bei Abbruch bleibt der bisherige Graph unverändert.

Procedure TKnotenform.KanteverschiebenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Kante verschieben mit der Maus angewählt wird.Bei einem Drücken (und Gedrückt lassen) der linken Maustaste über einer Kante,wird diese Kante angewählt,und die Weite der Kante kann mit der Maus durch Ziehen senkrecht zur Verbindungsstrecke von Anfangs- und Endknoten verändert werden.Die in der Kante gespeicherte Kantenweite ändert sich entsprechend.Der Vorgang wird durch das Loslassen der Maustaste beendet.

Menü Eigenschaften:

Procedure TKnotenform.AnzahlKantenundEckenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Anzahl Kanten und Ecken mit der Maus angewählt wird.Es erscheint ein Ausgabefenster mit Informationen über die Zahl der Kanten und Knoten,die Zahl der Gebiete im planaren Graphen und die Anzahl der Kanten vom Typ Real,Integer und String.

Procedure TKnotenform.ParallelkantenundSchlingenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Parallelkanten und Schlingen mit der Maus angewählt wird.Es erscheint ein Ausgabefenster mit Informationen über die Zahl der Schlingen (Kanten mit gleichem Anfangs- und Endknoten) und der Zahl der der parallelen bzw. antiparallelen Kanten im Graphen für jeweils den Fall,dass alle Kantenrichtungen berücksichtigt oder nicht berücksichtigt werden sollen (d.h. der Graph als gerichtet oder ungerichtet angesehen wird).

Procedure TKnotenform.EulerlinieClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Eulerlinie mit der Maus angewählt wird.Es erscheint ein Ausgabefenster mit Informationen darüber,ob der Graph eine geschlossene oder offene Eulerlinie besitzt.Im letzteren Fall werden der Anfangsknoten und Endknoten des Pfades automatisch bestimmt und angezeigt.

Procedure TKnotenform:KreiseClick(Sender: TObject)

Diese Methode wird ausgeführt,wenn das Menü Kreise mit der Maus angewählt wird.Es erscheint ein Ausgabefenster mit Informationen darüber,ob der Graph einen Kreis besitzt oder nicht.Für den ungerichteten Graph wird die Kantenzahl eines Kreises mit minimaler und maximaler Kantenzahl angegeben.Außerdem wird bei Vorliegen entsprechender Voraussetzungen angezeigt,ob der

schlichte Graph maximal planar oder bestimmt nicht planar ist (nach einem notwendigen Kriterium).Schließlich werden alle Kreise mit fester Länge,die in einem Eingabefenster gewählt werden kann,im Graphen,dessen Kanten als ungerichtet aufgefaßt werden,bestimmt.Die Kreise werden als Knotenfolge im Ausgabefenster und im Label Ausgabe1 ausgegeben und im Demomodus markiert im Graphen angezeigt.Die Anzahl dieser Kreise wird angezeigt.

Procedure TKnotenform.AnzahlBrueckenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Anzahl Brükken mit der Maus angewählt wird.Es erscheint ein Ausgabefenster mit Informationen über die Anzahl der Brückenkanten im vorliegenden Graphen.Eine Kante ist Brücke,wenn bei ihrer Entfernung aus dem Graph der Graph in mehr Gebiete als zuvor zerfällt.Die Brückenkanten werden im Ausgabefenster (Menü Ausgabe) angezeigt,und können auch durch einen Mausklick auf die Panelzeichenfläche mittels der der Listbox sichtbar gemacht werden.Die Brücken werden markiert im Graphen angezeigt.

Procedure TKnotenform.KnotenanzeigenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Knoten anzeigen mit der Maus angewählt wird.Es erscheint ein Ausgabefenster mit Angabe der Zahl der Knoten im Graphen.Die Knoten mit Inhalt und Koordinaten sowie Grad werden im Ausgabefenster (Menu Ausgabe) angezeigt,und es erscheint automatisch die in einer Zeile geöffnete Listbox (falls nötig mit vertikaler Scrolleiste) mit denselben Informationen.

Procedure TKnotenform.KantenanzeigenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Kanten anzeigen mit der Maus angewählt wird.Es erscheint ein Ausgabefenster mit Angabe der Zahl der Kanten sowie der Zahl der gerichteten und der ungerichteten Kanten im Graphen.Die Kanten mit Inhalt und Anfangs-und Endknotenbezeichnung sowie mit Typ (Real,Integer,String) und ihrer Richtung werden im Ausgabefenster (Menu Ausgabe) angezeigt,und es erscheint automatisch die in einer Zeile geöffnete Listbox (falls nötig mit vertikaler Scrolleiste) mit denselben Informationen.

Menü Ausgabe:

Procedure TKnotenform.AusgabefensterClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Ausgabefenster mit der Maus angewählt wird.Das Ausgabefenster (Ausgabeform) wird erzeugt und geöffnet (Showmodal) und zeigt,nachdem eine der unter den Menüpunkten Pfade bzw. Anwen-

dungen anzuwählenden Algorithmen ausgeführt wurde (ebenfalls bei den Algorithmen der Untermenüs Kreise,Anzahl Brücken, Knoten anzeigen bzw. Kanten anzeigen im Menu Eigenschaften sowie bei Anwahl der Aktionsschalterelemente),Ergebnisse dieser Algorithmen an.Sonst wird „leere Ausgabe“ ausgegeben.Die Ergebnisse werden aus der Listenbox in das Gitternetz der Ausgabeform kopiert.Das Ausgebefenster wird durch Anwahl des Menüs Ende im Ausgabefenster beendet und geschlossen sowie aus dem Speicher gelöscht.

Procedure TKnotenform.AbbruchClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Abbruch mit der Maus angewählt wird.Dadurch wird die Ausführung eines gerade laufenden Algorithmus der Menüs Pfade oder Anwendungen abgebrochen.Dies ist insbesonders interessant,wenn eine Anwendung im Demomodus verzögert abläuft.Der Demo-Modus wird dann bendet.Auch der Einzelschrittmodus wird abgebrochen.Die Anwahl dieses Menüs mit der Maus wirkt als Wechselschalter zwischen Abbruch und nicht Abbruch,d.h., aus Abbruch wird nicht Abbruch und umgekehrt.Entsprechend wird das Feld Abbruch_ der Graphen Graph und GraphH gesetzt.Die Standardereignismethoden der Paintbox bezüglich Bewegen und Klicken mit der Maus werden wiederhergestellt.Der Text des Menüs (Caption) wechselt zwischen Abbruch an und Abbruch aus hin und her.

Procedure TKnotenform.DemoClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Demo mit der Maus angewählt wird.Dadurch wird ein Eingabefenster geöffnet,in das eine Verzögerungszeit (Pausenzeit:maximal 31 s) für das Ausführen der Algorithmen im Menü Pfade und Anwendungen eingegeben werden kann,und die Algorithmen werden im Demomodus ausgeführt.Dies bedeutet,dass die Funktionweise des Algorithmus durch Darstellung von zusätzlichen Informationen wie z.B. das schrittweise Zeichnen von Suchpfaden längs der einzelnen Kanten oder die Darstellung von Zwischenergebnissen in Form von Knoten- oder Kanteninhalten oder die zusätzliche Anzeige von Kommentaren in den Labeln Ausgabe1 und Ausgabe2 verlangsamt durch die eingestellte Zeitverzögerung demonstriert wird.Durch nochmaliges Anwählen dieses Menüs kann der Demomodus wieder ausgeschaltet werden.Der Text des Menüs (Caption) wechselt zwischen Demo an und Demo aus hin und her.

Procedure TKnotenform.PausenzeitClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Pausenzeit mit der Maus angewählt wird.Es erscheint ein Eingabefenster,in das die neue Pausenzeit (Verzögerungszeit) für den Demomodus eingegeben werden kann.Maximaler Wert ist 31s.

Menü Hilfe:

Procedure TKnotenform.InhaltClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Inhalt mit der Maus angewählt wird.Dadurch öffnet sich die (Windows-) Hilfe zum Programm Knotengraph.Die Hilfe kann auch kontextsensitiv zu den Menueinträgen durch die F1-Taste aufgerufen werden.

Procedure TKnotenform.InfoClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Info mit der Maus angewählt wird.Dadurch wird ein Ausgabefenster mit Informationen über den Programmauthor aufgerufen.

Aktionsschalterelemente BtBtn:

Procedure TKnotenform.InitBitBtnmenu

Diese Methode setzt die nötigen Initialisierungen und muß als erste Methode in den Ereignismethoden der Aktionschalterelemente ausgeführt werden.

Procedure TKnotenform.BitBtn1(bis 11)Click(Sender: TObject)

Diese Methoden sind die Ereignismethoden der Aktionsschalterelemente BitBtn1 bis BitBtn11.Nach der Methode IniBtnmenu wird in diesen Methoden jeweils die Menümethode NeueKnotenClick,GraphladenClick, GraphspeichernunterClick, GraphdruckenClick, KnotenloeschenClick, KnoteneditierenClick, KnotenverschiebenClick,KantenloeschenClick, KanteeditierenClick, KanteverschiebenClick bzw. BildzeichnenClick aufgerufen,so dass durch Mausklick auf den entsprechenden Schalter vereinfacht auf diese entsprechenden Menüs zugegriffen werden kann.

Beschreibung der Unit UForm:

Die Unit definiert den Datentyp TKnotenformular,der sich durch Vererbung von dem Objekt TKnotenform ableitet.Die Variable Knotenformular vom Typ TKnotenformular wird in dieser Unit global definiert,und stellt durch seine Ereignismethoden durch visuelle Vererbung von der Form Knotenform die Benutzeroberfläche des Programms Knotengraph dar.

TKnotenformular hat folgende Datenfelder:

```
Pfade: TMenuItem;
AllePfade: TMenuItem;
AlleKreise: TMenuItem;
MinimalePfade: TMenuItem;
AnzahlZielknoten: TMenuItem;
```

```
TieferBaum: TMenuItem;
WeiterBaum: TMenuItem;
AbstandvonzweiKnoten1: TMenuItem;
AllePfadezwischenzweiKnoten: TMenuItem;
MinimalesGerstdesGraphen: TMenuItem;
Anwendungen: TMenuItem;
Netzwerkzeitplan: TMenuItem;
Hamiltonkreise: TMenuItem;
Eulerkreis: TMenuItem;
Frbbarkeit: TMenuItem;
EulerLinie: TMenuItem;
EndlicherAutomat: TMenuItem;
GraphalsRelation: TMenuItem;
MaximalerNetzfluss: TMenuItem;
Gleichungssystem: TMenuItem;
Markovketteabs: TMenuItem;
Markovkettestat: TMenuItem;
Graphreduzieren: TMenuItem;
MinimaleKosten: TMenuItem;
Transportproblem: TMenuItem;
OptimalesMatching: TMenuItem;
ChinesischerBrieftrger: TMenuItem;
```

Zum Hauptmenü Mainmenu vom Typ TMainmenü von TKnotenform werden durch visuelle Vererbung folgende Untermenüs vom Typ TMenuItem hinzugefügt:
Pfade, AllePfade, AlleKreise, MinimalePfade, AnzahlZielknoten, TieferBaum, WeiterBaum, AbstandvonzweiKnoten, AllePfadezwischenzweiKnoten, MinimalesGeruestdesGraphen, Anwendungen, Netzwerkzeit, Hamiltonkreise, Eulerkreise, Faerbbarkeit, Eulerlinie, EndlicherAutomat, GraphalsRelation, MaximalerNetzfluss, MaximalesMatching, Gleichungssystem, Markovketteabs, Markovkettestat, Graphreduzieren, MinimaleKosten, Transportproblem, OptimalesMatching und ChinesischerBrieftraeger.

Außerdem wird ein Graph Automatenneugraph vom Typ TAutomatenneugraph mittels

```
TAutomatenneugraph=class(TAutomatengraph)
   constructor Create;override;
 end;
```

als globale Variable definiert.Er wird in der Anwendung Endlicher Automat benutzt und dient zur Demonstration,wie von einem Benuzer nachträglich ein Graphtyp mit neuen Knoten-oder Kantentypen deklariert werden kann,der nicht nur über alle Eigenschaften von TAutomatengraph und damit TInhaltsgraph verfügt,sondern auf den noch zusätzlich auf Grund der visuellen Vererbung alle Menü-Methoden und Komponenten der Form Knotenform angewendet werden können.

Die globale Boolesche Variable Automatenneugraphaktiv dient zur Unterscheidung,welcher Graph (Knotenform.Graph vom Typ TInhaltsgraph oder Automatenneugraph vom Typ Tautomatenneugraph) gerade benuzt wird. (Verwendung nur in der Anwendung endlicher Automat)

Constructor TAutomateneugraph.Create

Dies ist der vituelle Constructor von TAutomatenneugraph.Er ruft die Vorgängermethode auf,legt den Typ der in ihm enthaltenden Knoten-und Kantentypen fest und setzt das Feld TKnotenform.Graph auf sich selber.

Ersatz-Ereignismethoden von TKnotenformular:

Procedure TKnotenformular.AllePfadeMouseDown(Sender:TObject; Button: TMouseButton;Shift:TShiftState;X,Y:Integer)

Bezug:Graph TKnotenformular.Graph

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:= AllePfadeMouseDown) die Ereignismethode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt auf der Paintboxfläche mit den Koordinaten X und Y wird dann,falls sich dort ein Knoten befindet,dieser Knoten als erster Knoten ausgewählt.Durch einen erneuten Mausklick auf einen Punkt mit den Koordinaten X und Y wird dann ein zweiter Knoten bestimmt,der sich wiederum in der Nähe dieses Punktes befindet.Mit dem ersten und zweiten Knoten als Anfangs- und Endknoten werden alle möglichen Pfade zwischen den Knoten erzeugt und auf der Paintboxfläche markiert angezeigt als auch unter Verwendung der Bewertung Bewertung (Unit UInhgrph) im Label Ausgabe1 mit Kantensumme und Kantenprodukt als Folge von Kanteninhalten angezeigt.Die Ergebnisse werden ebenfalls im Ausgabefenster ausgegeben.Falls sich keine Knoten in der Nähe der Punkte mit den Koordinaten X und Y befinden,wird eine entsprechende Mitteilung ausgegeben,und es werden keine Pfade erzeugt.Der Graph wird neu gezeichnet,und es wird nach Ausgabe aller Pfade durch die Methode selber mittels Paintbox.OnMouseDown:=PaintboxMouseDown auf die Standardereignismethode der Paintbox zurückgeschaltet.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenformular.AbstandvonzweiKnotenMouseDown(Sender: TObject;Button:TMouseButton;Shift:TShiftState;X,Y:Integer)

Bezug:Graph TKnotenformular.Graph

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:= AbstandvonzweiKnotenMouseDown) die Ereignismethode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt auf der Paintboxfläche mit den Koordinaten X und Y wird dann,falls sich dort ein Knoten befindet,dieser Knoten als erster Knoten ausgewählt.Durch einen erneuten Mausklick auf einen Punkt mit den Koordinaten X und Y wird dann ein zweiter Knoten bestimmt,der sich wiederum in der Nähe dieses Punktes befindet.Mit dem ersten und zweiten Knoten als Anfangs- und Endknoten wird der minimale Pfad bezüglich der Bewertung Bewertung zwischen den Knoten erzeugt und auf der Paneloberfläche markiert angezeigt als auch unter der Bewertung Bewertung (Unit UInhgrph) im Label Ausgabe1 mit Kantensumme und Kantenprodukt als Folge von Kanteninhalten angezeigt.Die Ergebnisse werden ebenfalls im Ausgabefenster ausgegeben. Falls sich keine Knoten in der Nähe der Punkte mit den Koordinaten X und Y befindet,wird eine entsprechende Mitteilung ausgegeben und kein Pfad erzeugt.Der Graph wird neu gezeichnet,und es wird nach Ausgabe des minimalen Pfades durch die Methode selber mittels Paintbox.OnMouseDown:=PaintboxMouseDown auf die Standardereignismethode der Paintbox zurückgeschaltet.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenformular.PanelDownMouse(Sender:TObject; Button:TMouseButton;Shift:TShiftState;X,Y:Integer)

Bezug:Graph TKnotenformular.GraphH

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:= PanelDownMouse) die Ereignismethode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt auf der Paintboxfläche mit den Koordinaten X und Y wird dann,falls sich dort ein Knoten befindet,der Inhalt dieses Knotens Inhalt_ des Graphen GraphH in einem Ausgabefenster mit den Mittelpunktskoordinaten dieses Knotens ausgegeben.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenformular.NetzMouseDown(Sender:TObject; Button:TMouseButton;Shift:TShiftState;X,Y:Integer)

Bezug: Graph TKnotenformular.GraphH als TNetzgraph

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:= NetzDownMouse) die Ereignismethode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt auf der Paintboxoberfläche mit den Koordinaten X und Y wird dann,falls sich dort ein Knoten befindet,der Inhalt des Datenfeldes Ergebnis_ dieses Knotens des Graphen GraphH in einem Ausgabefenster

mit den Mittelpunktskoordinaten dieses Knotens ausgegeben und,falls sich dort eine Kante befindet,der Inhalt des Datenfeldes Ergebnis_ dieser Kante (mit Anfangs- und EndknotenInhalt_) des Graphen GraphH in einem Ausgabefenster ausgegeben.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenformular.Automaten1Mousedown(Sender:TObject; Button:TMouseButton;Shift:TShiftState;X,Y:Integer)

Bezug:Graph TKnotenformular.GraphH als TAutomatengraph

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:=Automaten1DownMouse) die Ereignismethode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt auf der Paintboxfläche mit den Koordinaten X und Y wird dann,falls sich dort ein Knoten befindet,dieser Knoten als Anfangszustandsknoten des Graphen ausgewählt und das Datenfeld Knotenart_ auf 1 gesetzt.Der Knoten wird grün markiert,und der Graph GraphH neu gezeichnet.Der Inhalt_ dieses Knotens wird als Anfangszustand in einem Ausgabefenster ausgegeben.Der Zustand_ von GraphH wird auf true gesetzt.Falls sich am Punkt mit den Koordinaten mit den Koordinaten X und Y kein Knoten befindet,wird die Mitteilung „Knoten mit der Maus auswählen“ in einem Ausgabefenster angezeigt.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenformular.Automaten2Mousedown(Sender:TObject; Button:TMouseButton;Shift:TShiftState;X,Y:Integer)

Bezug:Graph TKnotenformular.GraphH als TAutomatengraph

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:= Automaten2DownMouse) die Ereignismethode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt auf der Paintboxfläche mit den Koordinaten X und Y wird dann,falls sich dort ein Knoten befindet,der nicht als Anfangszustand ausgewählt wurde (sonst erfolgt entsprechende Fehlermeldung),dieser Knoten als Endzustandsknoten des Graphen ausgewählt und das Datenfeld Knotenart_ auf 2 gesetzt.Der Knoten wird blau markiert und der Graph GraphH neu gezeichnet.Der Inhalt_ dieses Knotens wird als Anfangszustand in einem Ausgabefenster ausgegeben.Der Zustand_ von GraphH wird auf true gesetzt.Falls sich am Punkt mit den Koordinaten mit den Koordinaten X und Y kein Knoten befindet,wird die Mitteilung „Knoten mit der Maus auswählen“ in einem Ausgabefenster angezeigt. Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenformular.GleichungssystemMousedown(Sender: TObject;Button:TMouseButton;Shift:TShiftState;X,Y:Integer)

Bezug:Graph TKnotenformular.GraphH als TGleichungssystemgraph

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:= GleichungssystemMouseDown) die Ereignismethode PaintboxMouseDown ersetzen. Bei Mausklick auf den Punkt auf der Paintboxfläche mit den Koordinaten X und Y wird dann,falls sich dort ein Knoten befindet,dieser Knoten als „letzter Mausklickknoten“ des Graphen GraphH ausgewählt (letzter mit der Maus angewählter Knoten).Der Knoten wird rot markiert und der Graph GraphH neu gezeichnet.Der Zustand_ von GraphH wird auf true gesetzt (ebenfalls,wenn Abbruch_ von GraphH auf true gesetzt wurde.).Falls sich am Punkt mit den Koordinaten mit den Koordinaten X und Y kein Knoten befindet,wird die Mitteilung „Knoten mit der Maus auswählen“ in einem Ausgabefenster angezeigt.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenformular.MarkovMouseDown(Sender: TObject; Button:TMouseButton;Shift:TShiftState;X,Y:Integer)

Bezug:Graph TKnotenformular.GraphH als TMarkovgraph

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:= MarkovMouseDown) die Ereignismethode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt auf der Paintboxfläche mit den Koordinaten X und Y wird dann,falls sich dort ein Knoten befindet,dieser Knoten als letzter Mausklickknoten des Graphen GraphH ausgewählt (letzter mit der Maus angewählter Knoten).Der Datenfeld Ergebnis_ dieses Knotens wird in einem Fenster ausgegeben.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Procedure TKnotenformular.KostenMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

Bezug:Graph TKnotenformular.GraphH als TMinimaleKostengraph

Diese Ereignismethode kann (mittels der Anweisung Paintbox.OnMouseDown:= KostenMouseDown) die Ereignismethode PaintboxMouseDown ersetzen.Bei Mausklick auf den Punkt auf der Paintboxfläche mit den Koordinaten X und Y wird dann,falls sich dort eine Kante befindet,ein Eingabefenster geöffnet,in dem die Kosten_ (pro Flusseinheit) für diese Kante (im Graph GraphH)

eingegeben werden können.Falls die Kosten für eine Kante erfolgreich eingegeben werden konnten,wird der Zustand_ des Graphen GraphH auf true gesetzt.Der Parameter Button übergibt die gedrückte Maustaste.Shift übermittelt den Status der Tasten Strg,Alt,der Umschalttasten und der Maustasten.

Menü-Ereignismethoden:

Menü Pfade:

Procedure TKnotenformular.AllePfadeClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Alle Pfade mit der Maus angewählt wird.Von dem zuletzt mit der linken Maustaste durch Klick angewählten Knoten aus bzw.,falls nicht vorhanden,vom zuerst in den Graph Graph eingefügten Knoten aus, werden alle möglichen Pfade zu den anderen Knoten des Graphen Graph bestimmt und im Ausgabefenster als auch in der Listbox als Folge von Kanteninhalten mit Pfadsumme und Pfadprodukt ausgegeben (Summe bzw. Produkt der Kanteninhalte:dabei wird eine Kante von der Art string als Eins gezählt gemäß der Bewertung Bewertung in der Unit UInhgrph).Im Demo-Modus werden die Pfade nacheinander - rot-gestrichelt hervorgehoben - als Kantenfolge auf der Paintboxzeichenfläche im Bild des Graphen markiert.Gleichzeitig wird die zugehörige Folge von Kanteninhalten und die jeweilige Pfadsumme sowie das Pfadprodukt im Label Ausgabe1 dargestellt.

Procedure TKnotenformular.AlleKreiseClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Alle Kreise mit der Maus angewählt wird.Von dem zuletzt mit der linken Maustaste durch Klick angewählten Knoten aus bzw.,falls nicht vorhanden,vom zuerst in den Graph Graph eingefügten Knoten aus, werden alle möglichen Kreise des Graphen Graph bestimmt und im Ausgabefenster als auch in der Listbox als Folge von Kanteninhalten mit Pfadsumme und Pfadprodukt (Summe bzw. Produkt der Kanteninhalte,dabei wird eine Kante von der Art string als Eins gezählt gemäß der Bewertung Bewertung in der Unit UInhgrph), ausgegeben.Im Demo-Modus werden die Pfade nacheinander - rot-gestrichelt hervorgehoben - als Kantenfolge auf der Paintboxzeichenfläche im Bild des Graphen markiert.Gleichzeitig wird die zugehörige Folge von Kanteninhalten und die jeweilige Pfadsumme sowie das Pfadprodukt im Label Ausgabe1 dargestellt.

Procedure TKnotenformular.MinimalePfadeClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Minimale Pfade mit der Maus angewählt wird.Von dem zuletzt mit der linken Maustaste durch Klick angewählten Knoten aus bzw.,falls nicht vorhanden,vom zuerst in den Graph Graph eingefügten Knoten aus,

werden alle minimalen Pfade,d.h. Pfade mit kleinster Pfadsumme zu den anderen Knoten des Graphen Graph bestimmt und im Ausgabefenster als auch in der Listbox als Folge von Kanteninhalten mit Pfadsumme und Pfadprodukt (Summe bzw. Produkt der Kanteninhalte,dabei wird eine Kante von der Art string als Eins gezählt gemäß der Bewertung Bewertung in der Unit UInhgrph). ausgegeben.Im Demo-Modus werden die Pfade nacheinander - rot-gestrichelt hervorgehoben - als Kantenfolge auf der Paintboxzeichenfläche im Bild des Graphen markiert.Gleichzeitig wird die zugehörige Folge von Kanteninhalten und die jeweilige Pfadsumme sowie das Pfadprodukt im Label Ausgabe1 dargestellt.

Procedure TKnotenformular.AnzahlZielknotenClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Anzahl Zielknoten mit der Maus angewählt wird.Von dem zuletzt mit der linken Maustaste durch Klick angewählten Knoten aus bzw.,falls nicht vorhanden,vom zuerst in den Graph Graph eingefügten Knoten aus,werden die auf allen nur möglichen Pfaden erreichbaren Knoten gezählt und ihre Anzahl in einem Ausgabefenster angezeigt.

Procedure TKnotenformular.TieferBaumClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Tiefer Baum mit der Maus angewählt wird.Von dem zuletzt mit der linken Maustaste durch Klick angewählten Knoten aus bzw.,falls nicht vorhanden,vom zuerst in den Graph Graph eingefügten Knoten aus jeweils als Wurzel,werden alle möglichen Pfade eines tiefen Baumsdurchlaufs zu den anderen Knoten des Graphen Graph bestimmt und im Ausgabefenster als auch in der Listbox als Folge von Kanteninhalten mit Pfadsumme und Pfadprodukt ausgegeben(Summe bzw. Produkt der Kanteninhalte:dabei wird eine Kante von der Art string als Eins gezählt gemäß der Bewertung Bewertung in der Unit UInhgrph).Im Demo-Modus werden die Pfade nacheinander - rot-gestrichelt hervorgehoben - als Kantenfolge auf der Paintboxzeichenfläche im Bild des Graphen markiert.Gleichzeitig wird die zugehörige Folge von Kanteninhalten und die jeweilige Pfadsumme sowie das Pfadprodukt im Label Ausgabe1 dargestellt.Es kann gewählt werden,ob er Baum in Preorder- oder in Postorder-Ordnung durchlaufen wird.

Procedure TKnotenformular.WeiterBaumClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Weiter Baum mit der Maus angewählt wird.Von dem zuletzt mit der linken Maustaste durch Klick angewählten Knoten aus bzw.,falls nicht vorhanden,vom zuerst in den Graph Graph eingefügten Knoten aus jeweils als Wurzel werden alle möglichen Pfade eines weiten Baumsuchdurchlaufs zu den anderen Knoten des Graphen Graph bestimmt und im Ausgabefenster als auch in der Listbox als Folge

von Kanteninhalten mit Pfadsumme und Pfadprodukt ausgegeben(Summe bzw. Produkt der Kanteninhalte:dabei wird eine Kante von der Art string als Eins gezählt gemäß der Bewertung Bewertung in der Unit UInhgrph).Im Demo-Modus werden die Pfade nacheinander - rot-gestrichelt hervorgehoben - als Kantenfolge auf der Paintboxzeichenfläche im Bild des Graphen markiert.Gleichzeitig wird die zugehörige Folge von Kanten-inhalten und die jeweilige Pfadsumme sowie das Pfadprodukt im Label Ausgabe1 dargestellt.

Procedure TKnotenformular.AbstandvonzweiKnotenClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Abstand von zwei Knoten mit der Maus angewählt wird.Zuerst werden durch Mausklick mit der linken Maustaste nacheinander zwei Knoten der Graphen Graph vom Benutzer ausgewählt,und danach wird der kürzeste Pfad (minimalste Pfad bezüglich der Kantensumme gemäß der Bewertung Bewertung in der Unit Uinhgrph) zwischen den beiden Knoten bestimmt,und im Ausgabefenster als auch in der Listbox als Folge von Kanteninhalten mit Pfadsumme und Pfadprodukt ausgegeben(Summe bzw. Produkt der Kanteninhalte:dabei wird eine Kante von der Art string als Eins gezählt gemäß der Bewertung Bewertung in der Unit UInhgrph).Gleichzeitig wird der Pfad - rot-gestrichelt hervorgehoben - als Kantenfolge auf der Paintboxzeichenfläche im Bild des Graphen markiert.Außerdem wird die zugehörige Folge von Kanteninhalten und die jeweilige Pfadsumme sowie das Pfadprodukt im Label Ausgabe1 dargestellt.

Procedure TKnotenformular.AllePfadezwischenzweiKnotenClick (Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Alle Pfade zwischen zwei Knoten mit der Maus angewählt wird.Zuerst werden durch Mausklick mit der linken Maustaste nacheinander zwei Knoten der Graphen Graph vom Benutzer ausgewählt,und danach werden alle Pfade zwischen den beiden Knoten bestimmt und im Ausgabefenster als auch in der Listbox als Folge von Kanteninhalten mit Pfadsumme und Pfadprodukt (Summe bzw. Produkt der Kanteninhalte:dabei wird eine Kante von der Art string als Eins gezählt gemäß der Bewertung Bewertung in der Unit UInhgrph), ausgegeben.Gleichzeitig werden die Pfade nacheinander rot-gestrichelt hervorgehoben - als Kantenfolge auf der Paintboxzeichenfläche im Bild des Graphen markiert.Außerdem wird die zugehörige Folge von Kanteninhalten und die jeweilige Pfadsumme sowie das Pfadprodukt im Label Ausgabe1 dargestellt.

Procedure TKnotenformular.MinimalesGeruestdesGraphenClick (Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Minimales Geruest des Graphen mit der Maus angewählt wird.Die Funktionsmethode bestimmt einen minimalen Spannbaum bzw. einen Wald nach dem Algorithmus von Kruskal.Die Bewertung der Kanten wird durch die Function Bewertung (Unit Uinhgrph) vorgegeben.Der Spannbaum bzw. der Wald wird rot-gestrichelt markiert auf der Zeichenfläche der Paintbox gezeichnet.Im Demo-Modus wird der Ablauf des Algorithmus durch Markieren der gerade gewählten Kante verdeutlicht.Im Label Ausgabe1 werden während des Algorithmus jeweils die zur Zeit gewählten Kanten und nach Ablauf des Algorithmus das Ergebnis als Folge der Kanten (Pfad) mit Kantensumme angezeigt.

Menü Anwendungen:

Procedure TKnotenformular.NetzwerkzeitClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Netzwerkzeitplan mit der Maus angewählt wird.Der Graph wird als Netzplan nach der CPM-Methode der Netzplantechnik interpretiert.Dabei bedeuten die Kanten Tätigkeiten,deren Zeitdauer durch den Kanteninhalt (mit der Bewertung Bewertung der Unit UInhgrph) dargestellt werden.Die zeitliche Reihenfolge der Tätigkeiten wird durch die Richtung der Kantenpfade (Kantenpfeile) auf dem gerichteten Graph vorgegeben.Die Knoten bedeuten Ereignisse.Durch die Kantenrichtungen ist jeweils ein Vor- bzw. Nachereignis definiert.
Der Algorithmus berechnet den frühmöglichsten Eintrittstermin (Startzeit),den spätmöglichsten Eintrittstermin (Endzeit) sowie die Pufferzeit eines CMP-Projektes in den einzelnen Kanten (Tätigkeiten) und Anfangs- und Endzeit sowie den Puffer in den einzelnen Knoten (Ereignisse). Außerdem wird der kritische Weg (Pufferzeit gleich 0) bestimmt und markiert.
Durch Klick mit der linken Maustaste auf die Kanten und Knoten werden die Ergebnisse angezeigt.Zudem werden die Ergebnisse im Ausgabefenster Ausgabe sowie in der Listbox dargestellt.Im Demomodus kann der Ablauf des Algorithmus verfolgt werden,indem die Reihenfolge und die Berechnung der Zeiten in den Knoten des Graphen dargestellt werden.Für den Start- und Endknoten (die vom Algorithmus selbstständig gefunden werden als Knoten mit nur aus-bzw. nur einlaufenden Kanten:Start- und Endereignis) werden zu Beginn die gewünschte Anfangs- und Endzeit des Projektes abgefragt.Bei Eingabe des Wertes 0 für die Endzeit oder eines kleineren Wertes als die frühstmögliche Endzeit der durch Anfangszeit und das Projekt vorgegebenen Endzeit,wird angenommen,dass als Endzeit die durch das Projekt vorgegebene frühstmögliche Endzeit gewählt werden soll.

Procedure TKnotenformular.HamiltonkreiseClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Hamiltonkreise mit der Maus angewählt wird.Ausgehend von dem zuletzt mit der linken Maustaste angewählten Knoten (falls vorhanden) bzw. von dem als erstes in den Graph eingefügten Knoten als Startknoten werden alle Hamiltonkreise des Graphen bestimmt,und im Demomodus wird das Suchen der Kreise gemäß dem verwendeten Backtracking-Algorithmus durch eine rot-markierte Darstellung der bisher für den momentanen Pfad gewählten Kanten veranschaulicht.Gleichzeitig werden die Pfade durch Angabe der Folge von Knoteninhalten mit Pfadsumme und Pfadprodukt im Label Ausgabe1 ausgegeben.Der Hamiltonkreis mit der kleinsten Länge (bezogen auf die Kantensumme) ist die Lösung des Traveling-Salesman-Problems und wird angegeben.Die Ergebnisse werden außerdem im Ausgabefenster Ausgabe angezeigt.Vor Ablauf des Algorithmus kann anfangs gewählt werden,ob der gegebene Graph (trotz eventuell vorhandener gerichteter Kanten) als ungerichtet aufgefaßt werden soll.

Procedure TKnotenformular.EulerkreiseClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Eulerkreis mit der Maus angewählt wird.Ausgehend von dem zuletzt mit der linken Maustaste angewählten Knoten (falls vorhanden) bzw. von dem als erstes in den Graph eingefügten Knoten als Startknoten werden alle Eulerkreise des Graphen bestimmt,und im Demomodus wird das Suchen der Kreise nach dem verwendeten Backtracking-Algorithmus durch eine rot-markierte Darstellung der bisher für den momentanen Pfad gewählten Kanten veranschaulicht.Gleichzeitig werden die Pfade durch Angabe der Folge von Knoteninhalten mit Pfadsumme und Pfadprodukt im Label Ausgabe1 ausgegeben.Die Ergebnisse werden außerdem im Ausgabefenster Ausgabe angezeigt.Vor Ablauf des Algorithmus kann anfangs gewählt werden,ob der gegebene Graph (trotz eventuell vorhandener gerichteter Kanten) als ungerichtet aufgefaßt werden soll.Außerdem kann gewählt werden,ob alle Lösungen oder nur eine Lösung gesucht werden soll.Wenn kein Eulerkreis existiert,wird dies durch eine entsprechende Ausgabe mitgeteilt.

Procedure TKnotenformular.FaerbbarkeitClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Färbbarkeit mit der Maus angewählt wird.Vorgegeben wird durch Eingabe in einem Fenster die Anzahl der Farben.Bestimmt werden eine bzw. alle Lösungen,den Graphen (d.h. die Knoten) mit der vorgegebenen Anzahl Farben zu färben,so dass jeweils zwei benachbarte (durch eine Kante verbundene) Knoten verschiedene Farben erhalten.
Die Ergebnisse werden im Ausgabefenster Ausgabe und graphisch durch entsprechend gefärbte Knotenkreise sowie durch Ausgabe der Farbzahlen (die den Farben entsprechen) in den Knoten angezeigt.Im Demomodus kann die graphische Darstellung verzögert

(durch eine einzugebende Pausenzeit) dargestellt werden.Gleichzeitig wird dadurch der Ablauf des Backtracking-Verfahrens veranschaulicht.Vor Beginn des Algorithmus kann gewählt werden,ob alle Lösungen oder nur eine Lösung bestimmt werden sollen.Wenn es keine Lösung gibt,wird dies durch eine entsprechende Ausgabe mitgeteilt.

Procedure TKnotenformular.EulerlinieClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Eulerlinie mit der Maus angewählt wird.Im Graph müssen genau zwei Knoten mit ungeradem Grad als Start- und Endknoten vorhanden sein,die vom Algorithmus automatisch erkannt werden.Zwischen diesen Start- und Endknoten werden alle Eulerlinien des Graphen bestimmt,und im Demomodus wird das Suchen der Linien nach dem verwendeten Backtracking-Algorithmus durch eine rot-markierte Darstellung der bisher für den momentanen Pfad gewählten Kanten veranschaulicht.Gleichzeitig werden die Pfade durch Angabe der Folge von Knoteninhalten mit Pfadsumme und Pfadprodukt im Label Ausgabe1 ausgegeben.Die Ergebnisse werden außerdem im Ausgabefenster Ausgabe angezeigt. Vor Ablauf des Algorithmus kann anfangs gewählt werden,ob der gegebene Graph (trotz eventuell vorhandener gerichteter Kanten) als ungerichtet aufgefaßt werden soll.Außerdem kann gewählt werden,ob alle Lösungen oder nur eine Lösung gesucht werden sollen.Wenn keine Eulerlinie existiert,wird dies durch eine entsprechende Ausgabe mitgeteilt.

Procedure TKnotenformular.endlicherAutomatClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü endlicher Automat mit der Maus angewählt wird.Die Knoten des Graphen werden als Zustandsmenge eines endlichen Automaten aufgefaßt und die Kanten mittels ihres Inhalts als Übergangsrelation.
Es kann gewählt werden,ob während des Ablaufs des Algorithmus Zwischenzustände des Graphen bzw. Automaten abgespeichert werden können oder nicht.Die abgespeicherten Zustände können bei Wahl der ersten Option dann wieder geladen und der Ablauf an der abgebrochenen Stelle fortgesetzt werden.
Die Wahl dieser ersten Option dient dazu die Wirkungsweise eines neuen Graphentypen TAutomatenneugraph zu demonstrieren,auf den durch Vererbung von TAutomatengraph und gleichzeitig durch visuelle Vererbung der Form Knotenform an Knotenformular schon alle Algorithmen der in der Knotenform vorhandenen Menüs angewendet werden könnnen.
Bei Wahl der zweiten Option wird der Graphentyp TAutomatengraph verwendet,bei dem die visuelle Vererbung der Form nicht wirksam ist.(Sie wirkt in diesem Fall nur auf den Graphentyp TInhaltsgraph.)
Bei der ersten Option kann zusätzlich gewählt werden,ob ein neuer Graph (vom Typ TAutomatenneugraph) neu erzeugt werden soll

oder ob ein entprechender Graph geladen werden soll.
Sonst wird wird der vorhandene Graph in den neuen Graphtyp TAutomatenneugraph konvertiert.Zum Start des Algorithmus muss der Button Start angeklickt werden.
Bei der ersten Option wird stattdessen mit dem Typ TAutomatengraph gearbeitet und der Algorithmus startet sofort. Ausgewählt werden muß zunächst durch Klick mit der linken Maustaste ein Anfangszustand und eine Menge von Endzuständen,wozu jeweils durch Mitteilung in einem Fenster aufgefordert wird.Nach Auswahl der Endzustände erscheint dann das Eingabefenster Eingabe.Die Ereignismethode für dieses Fenster ist die Methode EingabekeyPress.Durch fortgesetzte Eingabe von Zeichen des Eingabealphabets (Kanteninhalte) in dieses Fenster wird dann durch Veränderung des momentanen Zustands (Knoten) die Funktionsweise eines endlichen Automaten simuliert und veranschaulicht.Der momentane (Zustand) Knoten wird rot-markiert gekennzeichnet,und im Label Ausgabe2 wird die angewählte Zustandsfolge (Knotenfolge) dargestellt.Wenn der Endzustand erreicht ist,wird dies als Mitteilung in einem Fenster angezeigt und gefragt,ob der Algorithmus beendet werden soll.Im Ausgabefenster Ausgabe sowie in der Listbox wird die ausgewählte Zustandsfolge angezeigt.Durch einen Mausklick auf den Button Button Abbruch kann der Algorithmus jederzeit abgebrochen werden.

Procedure TKnotenformular.EingabeKeyPress(Sender:TObject;var Key:Char)

Diese Ereignismethode wird aufgerufen,falls die Enter-Taste während der Ausführung des Algorithmus Endlicher Automat gedrückt wird, und das Eingabefenster Eingabe den Focus hat.Dadurch wird gemäß des eingegebenen Zeichens der nächste Zustand ausgehend vom momentanen Zustand im endlichen Automaten angenommen,falls es sich bei der Eingabe um ein zulässiges Zeichen des Eingabealphabet gehandelt hat,das den momentanen Zustand in einen weiteren Zustand überführt (Inhalt bzw. Wert der vom momentanen Knoten ausgehende Kante).Sonst wird eine Fehlermeldung ausgegeben.Im Falle,dass der Graphentyp TAutomatenneugraph vewendet wird,wird dieser Graphentyp bei Erreichen des Endzustands wieder aus dem Speicher gelöscht.

Procedure TKnotenformular.ButtonClick(Sender:TObject)

Diese Ereignismethode wird aufgerufen,falls der Button Button (Caption Abbruch) mit der Maus angeklickt wird.Dadurch wird die Ausführung des Algorithmus des endlichen Automaten abgebrochen.Im Falle,dass der Graphentyp TAutomatenneugraph vewendet wird,wird dieser Graphentyp wieder aus dem Speicher gelöscht.

Procedure TKnotenformular.GraphalsRelationClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Graph als Relation mit der Maus angewählt wird.Die gerichteten Kanten des Graphen werden als Relation zwischen den Knoten(-inhalten bzw. -werten) aufgefaßt.Bestimmt wird (wahlweise) die transitive und die reflexive Hülle sowie eine Numerierung der Knoten in der Reihenfolge der durch die Relation vorgegeben Ordnung.Dazu müssen die entsprechenden Optionen in den Eingabefenster am Anfang gewählt werden.Außerdem wird abgefragt,ob der ursprüngliche Graph Graph wiederhergestellt werden soll oder ob Graph gemäß den gewünschten Optionen verändert werden soll.Die für die reflexive Hülle nötigen Zusatzschlingen werden durch grün-markierte Schlingenkanten der Knoten dargestellt,die für die transitive Hülle notwendigen Zusatzkanten durch rot-markierte Kanten.Die Nummerierung der Knoten gemäß der Ordnungsrelation wird (in aufsteigender Reihenfolge) zusätzlich zum Knoteninhalt in jedem Knoten angezeigt.Im Demomodus wird das Suchen der Ordnungsrelation durch blau markierte Darstellung der entsprechenden Suchpfade und Eintrag der entsprechenden Ordnungsnummern in den Knoten demonstriert.Im Ausgabefenster Ausgabe und in der Listbox wird die errechnete Ordnungsrelation ebenfalls angezeigt.

Procedure TKnotenformular.MaximalerNetzflussClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Maximaler Netfluss mit der Maus angewählt wird. Der Graph wird als Netz aufgefaßt,durch dessen Kanten ein Fluss geschickt werden kann.Die Kanteninhalte sind die oberen Kapazitätsgrenzen des Flusses.Die untere Grenze ist 0.
Der Algorithmus bestimmt den maximalen Fluss durch das Netz nach dem Verfahren von Ford-Fulkerson.
Die beiden Knoten ohne einlaufende Kanten bzw. ohne auslaufende Kanten sind der Quellen- und Senkenknoten.Sie dürfen nur einmal vorhanden sein und werden vom Algorithmus automatisch bestimmt und nacheinander in zwei Anzeigefenstern ausgegeben.Der Fluss durch die jeweilige Kante wird bei jeder Kante (zusätzlich zur Kantenschranke) nach Beenden des Algorithmus als Inhalt jeder Kante ausgegeben und auch im Ausgabefenster Ausgabe sowie in der Listbox dargestellt.Der maximale Netzwerkfluss wird in einem Anzeigefenster angegeben.Im Demomodus kann die Arbeitweise des Algorithmus nach Ford-Fulkerson durch Darstellung der Suchpfade und das Bestimmen des Flusszuwachses in den Pfaden anschaulich verfolgt werden.Das Symbol i bedeutet einen Wert,der unendlich groß ist (und im Rechner durch eine sehr große Zahl dargestellt wird).Im Ausgabefenster Ausgabe und in der Listbox werden die errechneten Ergebnise ebenfalls ausgegeben.

Procedure TKnotenformular.MaximalesMatchingClick(Sender: TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Maximales Matching mit der Maus angewählt wird.In dem vorgegebenen Graph wird ein maximales Matching mit oder ohne Vorgabe eines Anfangsmatchings mit Hilfe der Methode fortgesetzten Suchens eines erweiternden Wegs bestimmt.Ob ein Anfangsmatching zunächst bestimmt werden soll,wird zu Anfang mit Hilfe eines Eingabefensters abgefragt.Zur Bestimung des Anfangsmatchings werden alle Kanten des Graphen untersucht,ob bei ihnen der Anfangs- und Endknoten noch isolierte Knoten sind.Ist dies der Fall,wird die Kante ins Anfangsmatching aufgenommen.
Die Kanten des Matchings werden rot-markiert mit einem roten Buchstaben M als Inhalt/Wert dargestellt.Im Demomodus wird die Arbeitsweise des Algorithmus Suchen eines erweiternden Weges durch markierte Darstellung der entsprechenden Suchpfade (Wege) demonstriert.Die Anwärterkanten (um eine Kante des Matchings zu werden),werden dabei durch den Buchstaben A als Inhalt und blau-markiert,und die Kanten,die aus dem Matching wieder entfent werden sollen,werden durch den Buchstaben L als Inhalt und grün-markiert dargestellt.Das Umfärben der Kanten längs des erweiternden Weges kann im Demomodus ebenfalls verfolgt werden:aus L wird der ursprüngliche Kanteninhalt,aus A wird der Buchstabe M.
Wenn kein weiterer erweiternder Weg mehr gefunden werden kann,ist das maximale Matching erreicht.Dann wird die Kantenzahl des Matchings angezeigt und evtl. angegeben,ob das Matching perfekt ist (keine isollierten Knoten mehr vorhanden).Im Demomodus werden zusätzlich jeweils Kommentare zu den einzelnen Schritten des Algorithmus im Label Ausgabe1 angezeigt.
Die Ergebnisse,welche Kanten zum Matching gehören,werden auch im Ausgabefenster Ausgabe ausgegeben und in der Listbox dargestellt.

Procedure TKnotenformular.GleichungssystemClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Gleichungssystem mit der Maus angewählt wird.Der Graph wird als Darstellung eines linearen Gleichungssystems aufgefaßt.
Die Knoteninhalte werden als die inhomogenen Koeffizienten angesehen.Die Koeffizienten $a_{i,j}$ des zugehörigen homogenen Systems werden als Inhalte der gerichteten Kanten vom Knoten i zu Knoten j dargestellt.Die Schlingen der Knoten sind also gerade die Koeffizienten $a_{i,i}$.
Nach dem Algorithmus von Mason wird durch fortgesetztes Löschen von Knoten jeweils ein äquivalentes Gleichungssystem erzeugt,bis die Lösung bei der Knotenzahl 1 direkt bestimmt werden kann.Durch mehrfache Anwendung könen so alle Lösungen x_i bestimmt werden,wenn jeweils jeder Knoten des Graphen als letzter gelöscht wird.Sobald der letzte Knoten gelöscht wird,wird der ursprüngliche Graph wiederhergestellt,und es kann nach einer wei-

teren Lösung gesucht werden.
Im Demomodus wird die schrittweise Veränderung der Kanten-und Knoteninhalte angezeigt,und im Label Ausgabe1 werden die einzelnen Schritte des Algorithmus kommentiert.
Die Ergebnisse werden sowohl im Ausgabefenster Ausgabe als auch in der Listbox ausgegeben.Die berechneten Lösungen x_i erscheinen zum Schluß in den Knoten als Inhalt/Wert,die jeweils als letzte Knoten gelöscht wurden.Auch im Label Ausgabe2 werden die schon bestimmten Lösungen angezeigt.

Procedure TKnotenformular.MarkovketteabsClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Markovkette(abs) mit der Maus angewählt wird.Der Graph wird als absorbierende Markovkette aufgefaßt.Die absorbierenden Zustände (Knoten) werden durch Schlingen mit dem Wert 1 gekennzeichnet.Die Kanteninhalte (Werte) sind die Übergangswahrscheinlichkeiten.
Der Algorithmus bestimmt wahlweise die Wahrscheinlichkeit und die mittlere Schrittzahl in einem bzw. allen Zuständen (Knoten des Graphen),um von dort in einen absorbierenden Zustand zu gelangen.Das Lösungsverfahren arbeitet gemäß dem Algorithmus von Engel (Lit 15,S 212) eine Markovkette als Spielbrett aufzufassen und bestimmt durch fortgesetztes Ziehen von (ganzen) Spielsteinen von den Knoten des Graphen zu den Nachbarknoten,deren Anzahl durch die Übergangswahrscheinlichkeiten der Kanten vorgegeben ist,die Wahrscheinlichkeiten und Schrittzahlen in den einzelnen Knoten.Steine werden nur dann gezogen,wenn sich in einem Knoten soviele Steine angesammelt haben,dass es die Übergangswahrscheinlichkeiten gestatten,eine ganze Zahl von Steinen zu bewegen.Ein Knoten heißt kritisch,wenn der Zug von Steinen von diesem Knoten aus dann möglich ist,wenn die Anzahl der Steine dort um 1 erhöht wird.Das Spiel beginnt mit einem kritischen Zustand aller Knoten des Graphen.Im gewählten Anfangsknoten wird dann die Anzahl der Steine um 1 erhöht.Es endet,wenn der ursprüngliche kritische Zustand wieder erreicht wird.Wahrscheinlichkeit und mittlere Schrittzahl lassen sich aus der am Ende vorhandenen Anzahl der Steine in den Knoten und aus der Anzahl der absorbierten Steine berechnen.
Im Demomodus wird jeweils die Verteilung der Steine in den einzelnen Knoten in den Labeln Ausgabe1 und Ausgabe2 angezeigt.
Vor Beginn des Algorithmus kann gewählt werden,ob die Wahrscheinlichkeit und die mittlere Schrittzahl für nur einen Knoten (nämlich den zuletzt mit der linken Maustaste angewählten Knoten,falls existent oder ersatzweise dem zuerst in den Graph eingefügten Knoten) oder für alle Knoten des Graphen bestimmt werden sollen.Die errechneten Wahrscheinlichkeiten werden als Knoteninhalte/Werte angezeigt.Durch Klick auf einen Knoten mit der linken Maustaste öffnet sich ein Fenster,in dem Knoteninhalt,Koordinaten sowie Wahrscheinlichkeit und mittlere

Schrittzahl angezeigt werden.Außerdem werden die Ergebnisse auch im Ausgabefenster Ausgabe und in der Listbox ausgegeben.

Procedure TKnotenformular.MarkovkettestatClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Markovkette(stat) mit der Maus angewählt wird.Der Graph wird als stationäre Markovkette aufgefaßt.Die Kanteninhalte sind die Übergangswahrscheinlichkeiten.
Der Algorithmus bestimmt wahlweise die stationäre Wahrscheinlichkeit und die mittlere Schrittzahl in einem bzw. allen Zuständen (Knoten des Graphen). Das Lösungsverfahren arbeitet gemäß dem Algorithmus von Engel, eine Markovkette als Spielbrett aufzufassen(Lit 15,S 215) und bestimmt durch fortgesetztes,gleichzeitiges Ziehen von (ganzen) Spielsteinen von den Knoten des Graphen zu den Nachbarknoten,deren Anzahl durch die Übergangswahrscheinlichkeiten der Kanten vorgegeben ist,die Wahrscheinlichkeiten und Schrittzahlen in den einzelnen Knoten.Steine werden nur dann gezogen,wenn sich in einem Knoten soviele Steine angesammelt haben,dass es die Übergangswahrscheinlichkeiten gestatten,eine ganze Zahl von Steinen zu bewegen.Anfangs wird jeder Knoten mit der kleinsten Anzahl Steine geladen,so dass man ziehen kann.Das Ende des Spiels ist erreicht,wenn sich eine stationäre Verteilung von Steinen in den Knoten eingestellt hat.Wahrscheinlichkeit und mittlere Schrittzahl lassen sich dann aus der Zahl der Steine in den Knoten und der Gesamtzahl der Steine in allen Knoten bestimmen.
Im Demomodus wird jeweils die Verteilung der Steine in den einzelnen Knoten in den Labeln Ausgabe1 und Ausgabe2 angezeigt.
Vor Beginn des Algorithmus kann gewählt werden,ob die Wahrscheinlichkeit und die mittlere Schrittzahl für nur einen Knoten (nämlich für den zuletzt mit der linken Maustaste angewählten Knoten,falls existent oder ersatzweise für den zuerst in den Graph eingefügten Knoten) oder für alle Knoten des Graphen bestimmt werden sollen.Die errechneten Wahrscheinlichkeiten werden als Knoteninhalte angezeigt.Durch Klick auf einen Knoten mit der linken Maustaste öffnet sich ein Fenster,in dem Knoteninhalt,Koordinaten sowie Wahrscheinlichkeit und mittlere Schrittzahl angezeigt werden.Außerdem werden die Ergebnisse auch im Ausgabefenster und in der Listbox ausgegeben.

Procedure TKnotenformular.GraphreduzierenClick(Sender:TObject)

Diese Ereignismethode wird ausgeführt,wenn das Menü Graph reduzieren mit der Maus angewählt wird.Nach dem Algorithmus von Mason und unter Berücksichtigung der Regeln der Reduktion einer Markovkette (Schlingen und Parallelkanten) wird durch fortgesetztes Löschen von Knoten jeweils ein äquivalenter Graph erzeugt,bis die Wahrscheinlicheit bei der Existenz von nur noch einem Knoten im Graph direkt ermittelt werden kann.Dazu wird der

9)Verwendung der Entwicklungsumgebung EWK durch die Programmiersprache C++

Mittels des C++-Builders (getestet in der Version 3) von Borland/ Inprise lassen sich auf der Grundlage der (mittels Delphi erstellten) Entwicklungsumgebung EWK durch Vererbung aller dort definierten Objekte und der Formulare insbesondere des Hauptformulars sämtliche im Abschnitt C dieser Arbeit genannten Graphenalgorithmen 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.

Zur visuellen Vererbung wird zunächst mittels der Entwicklungsumgebung des C++-Builders ein neues Projekt,z.B. KProjekt angelegt,in das durch das Menu Project/Add to Project sämtliche Delphi-Units der Entwicklungsumgebung EWK (einschließlich der Formulare,deren DFM-Dateien sich in demselben Verzeichnis wie die Units befinden müssen) hinzugefügt werden.Anschließend kann mittels New/Kprojekt das Formular Knotenform als Knotenformular (Unit UForm) visuell vererbt werden und erbt damit alle Eigenschaften von Knotenform.

In dem Formular wird dann beispielsweise ein neues Menü Pfade/Alle Pfade angelegt,in dem dann der entsprechende Algorithmus mittels C++-Quelltext programmiert werden kann.Um analog der Delphi-Entwicklungsumgebung DWK vorzugehen,wird außerdem eine neue C++-Unit UPfad.cpp mit Header-File UPfad.hpp angelegt,in dem die Objekte und die entsprechenden Methoden zu der Graphenanwendung Alle Pfade (von einem Knoten aus) in C++ erstellt werden.

Die Unit wird in UForm eingebunden und anschließend compiliert sowie gelinkt.Die C++Entwicklungsumgebung generiert automatisch C++-Header-Files zu allen Delphi-Units,so dass sich ein unter C++ ablauffähiger Quellcode ergibt.So kann die Entwicklungsumgebung EWK allgemein zum Erstellen von Graphenalgorithmen (z.B der aus Kapitel C oder anderer weiterer Algorithmen) in C++ genutzt werden.

Das oben genannte 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.

Im folgenden sind alle Cpp-Units und Header-Files wiedergegeben:

Quellcode ULIST.HPP:

```
// Borland C++ Builder
// Copyright (c) 1995, 1998 by Borland International
// All rights reserved

// (DO NOT EDIT: machine generated header) 'UList.pas' rev: 3.00

#ifndef UListHPP
#define UListHPP
#include <SysUtils.hpp>
#include <Dialogs.hpp>
#include <Forms.hpp>
#include <Windows.hpp>
#include <Classes.hpp>
#include <SysInit.hpp>
#include <System.hpp>

//— user supplied ——————————————————————————

namespace Ulist
{
//— type declarations ——————————————————————
typedef void __fastcall (*TVorgang)(System::TObject* Ob);

typedef void __fastcall (*THandlung)(System::TObject* Ob1,
System::TObject* Ob2);

typedef bool __fastcall (*TBedingung)(System::TObject* Ob);

typedef Extended __fastcall (*TWert)(System::TObject* Ob);

typedef bool __fastcall (*TVergleich)(System::TObject* Ob1,
System::TObject* Ob2, TWert Wert);

class DELPHICLASS TListe;
class DELPHICLASS TElement;
class PASCALIMPLEMENTATION TListe : public Classes::TList
{
    typedef Classes::TList inherited;

public:
    __fastcall TListe(void);
    HIDESBASE void __fastcall Free(void);
    void __fastcall Freeall(void);
    TElement* __fastcall Element(int Index);
    __property TElement* Items[int Index] = {read=Element};
    void __fastcall AmEndeanfuegen(System::TObject* Ob);
    void __fastcall AmAnfanganfuegen(System::TObject* Ob);
    void __fastcall AnPositioneinfuegen(System::TObject* Ob, int
```

```
Index);
    void __fastcall LoescheanderPosition(System::TObject* &Ob,
int N);
    void __fastcall AmAnfangloeschen(System::TObject* &Ob);
    void __fastcall AmEndeloeschen(System::TObject* &Ob);
    void __fastcall LoescheElement(System::TObject* Ob);
    void __fastcall VertauscheElemente(int Index1, int Index2);
    void __fastcall VerschiebeElement(int Index1, int Index2);
    void __fastcall FuerjedesElement(TVorgang Vorgang);
    void __fastcall FuerjedesElementzurueck(TVorgang Vorgang);
    void __fastcall FueralleElemente(System::TObject* Ob,
THandlung Handlung);
    void __fastcall FueralleElementezurueck(System::TObject* Ob,
THandlung Handlung);
    void __fastcall Loeschen(void);
    void __fastcall Sortieren(TVergleich Vergleich, TWert Wert);
    int __fastcall Anzahl(void);
    Extended __fastcall WertSummederElemente(TWert Wert);
    Extended __fastcall WertproduktderElemente(TWert Wert);
    bool __fastcall Leer(void);
    int __fastcall Erstes(void);
    int __fastcall Letztes(void);
    int __fastcall Position(System::TObject* Ob);
    bool __fastcall ElementistinListe(System::TObject* Ob);
    int __fastcall ErsterichtigePosition(TBedingung Bedingung);
    int __fastcall ErstefalschePosition(TBedingung Bedingung);
    int __fastcall LetzterichtigePosition(TBedingung Bedingung);
    int __fastcall ErstepassendePosition(TVergleich Vergleich,
System::TObject* Ob, TWert Wert);
    int __fastcall ErsteunpassendePosition(TVergleich Vergleich,
System::TObject* Ob, TWert Wert);
    int __fastcall ErstebestePosition(TVergleich Vergleich, TWert
Wert);
    int __fastcall LetztebestePosition(TVergleich Vergleich,
TWert Wert);
public:
    /* TList.Destroy */ __fastcall virtual ~TListe(void) { }

};

class PASCALIMPLEMENTATION TElement : public Ulist::TListe
{
    typedef Ulist::TListe inherited;

private:
    int Wertposition_;
    Classes::TStringList* Wertliste_;
    void __fastcall SetzeWertposition(int P);
    int __fastcall WelcheWertposition(void);
    virtual void __fastcall Wertschreiben(System::AnsiString S);
```

```
    virtual System::AnsiString __fastcall Wertlesen();

public:
    __fastcall TElement(void);
    HIDESBASE void __fastcall Free(void);
    __property int Position = {read=WelcheWertposition,
write=SetzeWertposition, nodefault};
    virtual Classes::TStringList* __fastcall
Wertlisteschreiben(void) = 0;
    virtual void __fastcall Wertlistelesen(void) = 0;
    __property System::AnsiString Wert = {read=Wertlesen,
write=Wertschreiben};
public:
    /* TList.Destroy */ __fastcall virtual ~TElement(void) { }

};

//— var, const, procedure ——————————————————
extern PACKAGE int __fastcall GGT(int A, int B);
extern PACKAGE Extended __fastcall Tan(Extended X);
extern PACKAGE Extended __fastcall
StringtoReal(System::AnsiString S);
extern PACKAGE System::AnsiString __fastcall
RealtoString(Extended R);
extern PACKAGE System::AnsiString __fastcall Integertostring(int
I);
extern PACKAGE int __fastcall StringtoInteger(System::AnsiString
S);
extern PACKAGE bool __fastcall
StringistRealZahl(System::AnsiString S);
extern PACKAGE System::AnsiString __fastcall
RundeStringtoString(System::AnsiString S, int Stelle);
extern PACKAGE System::AnsiString __fastcall
RundeZahltoString(double R, int Stelle);
extern PACKAGE Extended __fastcall Minimum(Extended R1, Extended
R2);
extern PACKAGE Extended __fastcall Maximum(Extended R1, Extended
R2);
extern PACKAGE void __fastcall Pause(int N);

}   /* namespace Ulist */
#if !defined(NO_IMPLICIT_NAMESPACE_USE)
using namespace Ulist;
#endif
//— end unit ———————————————————————————
#endif  // Ulist
```

Quellcode UGRAPH.HPP:

```
// Borland C++ Builder
// Copyright (c) 1995, 1998 by Borland International
// All rights reserved

// (DO NOT EDIT: machine generated header) 'UGraph.pas' rev:
3.00

#ifndef UGraphHPP
#define UGraphHPP
#include <UList.hpp>
#include <SysUtils.hpp>
#include <Dialogs.hpp>
#include <Forms.hpp>
#include <Windows.hpp>
#include <Classes.hpp>
#include <SysInit.hpp>
#include <System.hpp>

//— user supplied ——————————————————————————

namespace Ugraph
{
//— type declarations ——————————————————————
typedef System::AnsiString __fastcall
(*TString)(System::TObject* Ob);

class DELPHICLASS TKantenliste;
class DELPHICLASS TKante;
class DELPHICLASS TGraph;
class PASCALIMPLEMENTATION TKantenliste : public Ulist::TListe
{
    typedef Ulist::TListe inherited;

public:
    __fastcall TKantenliste(void);
    HIDESBASE void __fastcall Free(void);
    HIDESBASE void __fastcall Freeall(void);
    TKante* __fastcall Kante(int Index);
    __property TKante* Items[int Index] = {read=Kante};
    TKantenliste* __fastcall Kopie(void);
    TGraph* __fastcall Graph(void);
    TGraph* __fastcall UGraph(void);
    System::AnsiString __fastcall Kantenlistealsstring();
public:
    /* TList.Destroy */ __fastcall virtual ~TKantenliste(void) {
}

};
```

```
class DELPHICLASS TKnoten;
class DELPHICLASS TKnotenliste;
class PASCALIMPLEMENTATION TKnotenliste : public Ulist::TListe
{
    typedef Ulist::TListe inherited;

public:
    __fastcall TKnotenliste(void);
    HIDESBASE void __fastcall Free(void);
    HIDESBASE void __fastcall Freeall(void);
    TKnoten* __fastcall Knoten(int Index);
    __property TKnoten* Items[int Index] = {read=Knoten};
    TKnotenliste* __fastcall Kopie(void);
    System::AnsiString __fastcall Knotenlistealsstring();
public:
    /* TList.Destroy */ __fastcall virtual ~TKnotenliste(void) {
}

};

class DELPHICLASS TPfadliste;
class DELPHICLASS TPfad;
class PASCALIMPLEMENTATION TPfadliste : public Ulist::TListe
{
    typedef Ulist::TListe inherited;

public:
    __fastcall TPfadliste(void);
    HIDESBASE void __fastcall Free(void);
    HIDESBASE void __fastcall Freeall(void);
    TPfad* __fastcall Pfad(int Index);
    __property TPfad* Items[int Index] = {read=Pfad};
    TPfadliste* __fastcall Kopie(void);
public:
    /* TList.Destroy */ __fastcall virtual ~TPfadliste(void) { }

};

class PASCALIMPLEMENTATION TKnoten : public Ugraph::TKnotenliste
{
    typedef Ugraph::TKnotenliste inherited;

private:
    TGraph* Graph_;
    TKantenliste* EingehendeKantenliste_;
    TKantenliste* AusgehendeKantenliste_;
    TPfadliste* Pfadliste_;
    int Wertposition_;
    Classes::TStringList* Wertliste_;
    bool Besucht_;
```

```
    bool Erreicht_;
    TGraph* __fastcall WelcherGraph(void);
    void __fastcall SetzeGraph(TGraph* G);
    TKantenliste* __fastcall WelcheEingehendeKantenliste(void);
    void __fastcall SetzeEingehendeKantenliste(TKantenliste* L);
    TKantenliste* __fastcall WelcheAusgehendeKantenliste(void);
    void __fastcall SetzeAusgehendeKantenliste(TKantenliste* L);
    TPfadliste* __fastcall WelchePfadliste(void);
    void __fastcall SetzePfadliste(TPfadliste* P);
    void __fastcall SetzeWertposition(int P);
    int __fastcall WelcheWertposition(void);
    Classes::TStringList* __fastcall WelcheWertliste(void);
    void __fastcall SetzeWertliste(Classes::TStringList* W);
    bool __fastcall Istbesucht(void);
    void __fastcall Setzebesucht(bool B);
    bool __fastcall Isterreicht(void);
    void __fastcall Setzeerreicht(bool E);
    void __fastcall Wertschreiben(System::AnsiString S);
    System::AnsiString __fastcall Wertlesen();

public:
    __fastcall virtual TKnoten(void);
    HIDESBASE void __fastcall Free(void);
    HIDESBASE void __fastcall Freeall(void);
    __property TGraph* Graph = {read=WelcherGraph,
write=SetzeGraph};
    __property TKantenliste* EingehendeKantenliste =
{read=WelcheEingehendeKantenliste,
write=SetzeEingehendeKantenliste
        };
    __property TKantenliste* AusgehendeKantenliste =
{read=WelcheAusgehendeKantenliste,
write=SetzeAusgehendeKantenliste
        };
    __property TPfadliste* Pfadliste = {read=WelchePfadliste,
write=SetzePfadliste};
    __property int Position = {read=WelcheWertposition,
write=SetzeWertposition, nodefault};
    __property Classes::TStringList* Wertliste =
{read=WelcheWertliste, write=SetzeWertliste};
    __property bool Besucht = {read=Istbesucht,
write=Setzebesucht, nodefault};
    __property bool Erreicht = {read=Isterreicht,
write=Setzeerreicht, nodefault};
    __property System::AnsiString Wert = {read=Wertlesen,
write=Wertschreiben};
    virtual Classes::TStringList* __fastcall
Wertlisteschreiben(void) = 0;
    virtual void __fastcall Wertlistelesen(void) = 0;
    void __fastcall FueralleausgehendenKanten(Ulist::THandlung
```

```
Handlung);
    void __fastcall FueralleeingehendenKanten(Ulist::THandlung
Handlung);
    void __fastcall FueralleKanten(Ulist::THandlung Handlung);
    void __fastcall FuerjedeausgehendeKante(Ulist::TVorgang Vor-
gang);
    void __fastcall FuerjedeeingehendeKante(Ulist::TVorgang Vor-
gang);
    void __fastcall FuerjedeKante(Ulist::TVorgang Vorgang);
    int __fastcall Kantenzahlausgehend(void);
    int __fastcall Kantenzahleingehend(void);
    int __fastcall Kantenzahlungerichtet(void);
    int __fastcall Kantenzahl(void);
    int __fastcall AnzahlSchlingen(void);
    int __fastcall Grad(bool Gerichtet);
    bool __fastcall IstimPfad(void);
    void __fastcall LoeschePfad(void);
    void __fastcall ErzeugeallePfade(void);
    void __fastcall ErzeugeallePfadeZielKnoten(TKnoten* Kno);
    bool __fastcall PfadzumZielknoten(TKnoten* Kno, TKante* Ka);
    void __fastcall ErzeugeTiefeBaumPfade(bool Preorder);
    void __fastcall ErzeugeWeiteBaumpfade(void);
    void __fastcall ErzeugeKreise(void);
    void __fastcall ErzeugeminimalePfade(Ulist::TWert Wert);
    void __fastcall ErzeugeminimalePfadenachDijkstra(Ulist::TWert
Wert);
    void __fastcall SortierePfadliste(Ulist::TWert Wert);
    int __fastcall AnzahlPfadZielknoten(void);
    TPfad* __fastcall MinimalerPfad(Ulist::TWert Wert);
    TPfad* __fastcall MaximalerPfad(Ulist::TWert Wert);
    bool __fastcall KnotenistKreisknoten(void);
public:
    /* TList.Destroy */ __fastcall virtual ~TKnoten(void) { }

};

class PASCALIMPLEMENTATION TKante : public Ugraph::TKantenliste
{
    typedef Ugraph::TKantenliste inherited;

private:
    TKnoten* Anfangsknoten_;
    TKnoten* Endknoten_;
    TKnoten* Pfadrichtung_;
    bool Gerichtet_;
    int Wertposition_;
    Classes::TStringList* Wertliste_;
    bool Besucht_;
    bool Erreicht_;
    TKnoten* __fastcall WelcherAnfangsknoten(void);
```

```
    void __fastcall SetzeAnfangsknoten(TKnoten* Kno);
    TKnoten* __fastcall WelcherEndknoten(void);
    void __fastcall SetzeEndknoten(TKnoten* Kno);
    TKnoten* __fastcall WelchePfadrichtung(void);
    void __fastcall SetzePfadrichtung(TKnoten* Kno);
    bool __fastcall Istgerichtet(void);
    void __fastcall Setzegerichtet(bool Gerichtet);
    void __fastcall SetzeWertposition(int P);
    int __fastcall WelcheWertposition(void);
    Classes::TStringList* __fastcall WelcheWertliste(void);
    void __fastcall SetzeWertliste(Classes::TStringList* W);
    bool __fastcall Istbesucht(void);
    void __fastcall Setzebesucht(bool B);
    bool __fastcall Isterreicht(void);
    void __fastcall Setzeerreicht(bool E);
    void __fastcall Wertschreiben(System::AnsiString S);
    System::AnsiString __fastcall Wertlesen();

public:
    __fastcall virtual TKante(void);
    HIDESBASE void __fastcall Free(void);
    HIDESBASE void __fastcall Freeall(void);
    __property TKnoten* Anfangsknoten =
{read=WelcherAnfangsknoten, write=SetzeAnfangsknoten};
    __property TKnoten* Endknoten = {read=WelcherEndknoten,
write=SetzeEndknoten};
    __property TKnoten* Pfadrichtung = {read=WelchePfadrichtung,
write=SetzePfadrichtung};
    __property bool Gerichtet = {read=Istgerichtet,
write=Setzegerichtet, nodefault};
    __property int Position = {read=WelcheWertposition,
write=SetzeWertposition, nodefault};
    __property Classes::TStringList* Wertliste =
{read=WelcheWertliste, write=SetzeWertliste};
    __property bool Besucht = {read=Istbesucht,
write=Setzebesucht, nodefault};
    __property bool Erreicht = {read=Isterreicht,
write=Setzeerreicht, nodefault};
    __property System::AnsiString Wert = {read=Wertlesen,
write=Wertschreiben};
    virtual Classes::TStringList* __fastcall
Wertlisteschreiben(void) = 0;
    virtual void __fastcall Wertlistelesen(void) = 0;
    TKnoten* __fastcall Zielknoten(TKnoten* Kno);
    TKnoten* __fastcall Quellknoten(TKnoten* Kno);
    bool __fastcall KanteistSchlinge(void);
    bool __fastcall KanteistKreiskante(void);
public:
    /* TList.Destroy */ __fastcall virtual ~TKante(void) { }
};
```

```
class PASCALIMPLEMENTATION TPfad : public Ugraph::TPfadliste
{
    typedef Ugraph::TPfadliste inherited;

public:
    __fastcall TPfad(void);
    HIDESBASE void __fastcall Free(void);
    TKnotenliste* __fastcall Knotenliste(void);
    TKantenliste* __fastcall Kantenliste(void);
    Extended __fastcall Pfadlaenge(void);
    Extended __fastcall PfadSumme(Ulist::TWert Wert);
    Extended __fastcall Pfadprodukt(Ulist::TWert Wert);
    System::AnsiString __fastcall Pfadstring(TString Sk);
    Classes::TStringList* __fastcall Pfadstringliste(TString Sk);
public:
    /* TList.Destroy */ __fastcall virtual ~TPfad(void) { }

};

class PASCALIMPLEMENTATION TGraph : public System::TObject
{
    typedef System::TObject inherited;

private:
    TKnotenliste* Knotenliste_;
    TKantenliste* Kantenliste_;
    int Wertposition_;
    Classes::TStringList* Wertliste_;
    bool Unterbrechung_;
    TKnotenliste* __fastcall WelcheKnotenliste(void);
    void __fastcall SetzeKnotenliste(TKnotenliste* K);
    TKantenliste* __fastcall WelcheKantenliste(void);
    void __fastcall SetzeKantenliste(TKantenliste* K);
    void __fastcall SetzeWertposition(int P);
    int __fastcall WelcheWertposition(void);
    Classes::TStringList* __fastcall WelcheWertliste(void);
    void __fastcall SetzeWertliste(Classes::TStringList* W);
    System::AnsiString __fastcall Wertlesen();
    void __fastcall Wertschreiben(System::AnsiString S);
    void __fastcall SetzeUnterbrechung(bool Ub);
    bool __fastcall WelcheUnterbrechung(void);

public:
    __fastcall virtual TGraph(void);
    HIDESBASE void __fastcall Free(void);
    void __fastcall Freeall(void);
    __property TKnotenliste* Knotenliste =
{read=WelcheKnotenliste, write=SetzeKnotenliste};
    __property TKantenliste* Kantenliste =
{read=WelcheKantenliste, write=SetzeKantenliste};
```

```
    __property int Position = {read=WelcheWertposition,
write=SetzeWertposition, nodefault};
    __property Classes::TStringList* Wertliste =
{read=WelcheWertliste, write=SetzeWertliste};
    __property bool Abbruch = {read=WelcheUnterbrechung,
write=SetzeUnterbrechung, nodefault};
    __property System::AnsiString Wert = {read=Wertlesen,
write=Wertschreiben};
    virtual Classes::TStringList* __fastcall
Wertlisteschreiben(void) = 0;
    virtual void __fastcall Wertlistelesen(void) = 0;
    void __fastcall ImGraphKnotenundKantenloeschen(void);
    bool __fastcall Leer(void);
    void __fastcall KnotenEinfuegen(TKnoten* Kno);
    void __fastcall Knotenloeschen(TKnoten* Kno);
    void __fastcall KanteEinfuegen(TKante* Ka, TKnoten* Anfangs-
knoten, TKnoten* EndKnoten, bool Gerichtet
        );
    void __fastcall EinfuegenKante(TKante* Ka);
    void __fastcall Kanteloeschen(TKante* Ka);
    void __fastcall LoescheKantenbesucht(void);
    void __fastcall LoescheKantenerreicht(void);
    void __fastcall LoescheKnotenbesucht(void);
    void __fastcall LoescheKnotenerreicht(void);
    void __fastcall FuerjedenKnoten(Ulist::TVorgang Vorgang);
    void __fastcall FuerjedeKante(Ulist::TVorgang Vorgang);
    TKnoten* __fastcall Anfangsknoten(void);
    TKante* __fastcall Anfangskante(void);
    int __fastcall AnzahlKanten(void);
    int __fastcall AnzahlSchlingen(void);
    int __fastcall AnzahlKantenmitSchlingen(void);
    int __fastcall AnzahlKnoten(void);
    int __fastcall AnzahlgerichteteKanten(void);
    int __fastcall AnzahlungerichteteKanten(void);
    int __fastcall AnzahlKomponenten(void);
    int __fastcall AnzahlparallelerKanten(void);
    int __fastcall AnzahlantiparallelerKanten(void);
    int __fastcall AnzahlparallelerKantenungerichtet(void);
    Extended __fastcall Kantensumme(Ulist::TWert Wert);
    Extended __fastcall Kantenprodukt(Ulist::TWert Wert);
    Classes::TStringList* __fastcall ListemitKnotenInhalt(TString
Sk);
    System::AnsiString __fastcall InhaltallerKnoten(TString Sk);
    Classes::TStringList* __fastcall
ListemitInhaltKantenoderKnoten(TString Sk);
    System::AnsiString __fastcall
InhaltallerKantenoderKnoten(TString Sk);
    void __fastcall Pfadlistenloeschen(void);
    void __fastcall SortiereallePfadlisten(Ulist::TWert Wert);
    TPfad* __fastcall BestimmeminimalenPfad(TKnoten* Kno1,
```

```
TKnoten* Kno2, Ulist::TWert Wert);
    bool __fastcall GraphhatKreise(void);
    bool __fastcall GraphhatgeschlosseneEulerlinie(bool Gerich-
tet);
    bool __fastcall GraphhatoffeneEulelinie(TKnoten* &Kno1,
TKnoten* &Kno2, bool Gerichtet);
    TGraph* __fastcall Kopie(void);
    bool __fastcall KanteverbindetKnotenvonnach(TKnoten* Kno1,
TKnoten* Kno2);
    TKante* __fastcall ErsteKantevonKnotenzuKnoten(TKnoten* Kno1,
TKnoten* Kno2);
    TKante* __fastcall ErsteSchlingezuKnoten(TKnoten* Kno);
    bool __fastcall Graphistpaar(void);
    bool __fastcall GraphistBinaerbaum(void);
public:
    /* TObject.Destroy */ __fastcall virtual ~TGraph(void) { }

};

//— var, const, procedure ——————————————————

}   /* namespace Ugraph */
#if !defined(NO_IMPLICIT_NAMESPACE_USE)
using namespace Ugraph;
#endif
//— end unit ———————————————————————————
#endif  // Ugraph
```

Quellcode UINHGRPH.HPP:

```
// Borland C++ Builder
// Copyright (c) 1995, 1998 by Borland International
// All rights reserved

// (DO NOT EDIT: machine generated header) 'UInhGrph.pas' rev:
3.00

#ifndef UInhGrphHPP
#define UInhGrphHPP
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Classes.hpp>
#include <Messages.hpp>
#include <Forms.hpp>
#include <Menus.hpp>
#include <Dialogs.hpp>
#include <Windows.hpp>
#include <Graphics.hpp>
#include <SysUtils.hpp>
#include <UKante.hpp>
```

```
#include <UGraph.hpp>
#include <UList.hpp>
#include <SysInit.hpp>
#include <System.hpp>

//— user supplied ——————————————————————

namespace Uinhgrph
{
//— type declarations ——————————————————
class DELPHICLASS TInhaltsknoten;
class PASCALIMPLEMENTATION TInhaltsknoten : public
Ugraph::TKnoten
{
    typedef Ugraph::TKnoten inherited;

private:
    int X_;
    int Y_;
    Graphics::TColor Farbe_;
    TPenStyle Stil_;
    char Typ_;
    int Radius_;
    System::AnsiString Inhalt_;
    int __fastcall Lesex(void);
    void __fastcall Schreibex(int X);
    int __fastcall Lesey(void);
    void __fastcall Schreibey(int Y);
    int __fastcall Welcherradius(void);
    void __fastcall Setzeradius(int R);
    Graphics::TColor __fastcall WelcheFarbe(void);
    void __fastcall SetzeFarbe(Graphics::TColor F);
    Graphics::TPenStyle __fastcall WelcherStil(void);
    void __fastcall SetzeStil(Graphics::TPenStyle T);
    char __fastcall WelcherTyp(void);
    void __fastcall SetzeTyp(char Typ);

public:
    __fastcall virtual TInhaltsknoten(void);
    HIDESBASE void __fastcall Free(void);
    HIDESBASE void __fastcall Freeall(void);
    virtual Classes::TStringList* __fastcall
Wertlisteschreiben(void);
    virtual void __fastcall Wertlistelesen(void);
    __property int X = {read=Lesex, write=Schreibex, nodefault};
    __property int Y = {read=Lesey, write=Schreibey, nodefault};
    __property int Radius = {read=Welcherradius,
write=Setzeradius, nodefault};
    __property Graphics::TColor Farbe = {read=WelcheFarbe,
write=SetzeFarbe, nodefault};
```

```
    __property Graphics::TPenStyle Stil = {read=WelcherStil,
write=SetzeStil, nodefault};
    __property char Typ = {read=WelcherTyp, write=SetzeTyp,
nodefault};
    void __fastcall ZeichneKnoten(Graphics::TCanvas* Flaeche);
    void __fastcall ZeichneDruckKnoten(Graphics::TCanvas*
Flaeche, int Faktor);
    void __fastcall Knotenzeichnen(Graphics::TCanvas* Flaeche,
bool Demo, int Pausenzeit);
    void __fastcall AnzeigePfadliste(Graphics::TCanvas* Flaeche,
Stdctrls::TLabel* Ausgabe, Classes::TStringList*
        &SListe, bool Zeichnen, bool LetzterPfad);
    Ugraph::TKantenliste* __fastcall ErzeugeminmaxKreise(bool
Minmax);
    void __fastcall ErzeugeKreisevonfesterLaenge(int Laenge);
public:
    /* TList.Destroy */ __fastcall virtual ~TInhaltsknoten(void)
{ }

};

typedef System::TMetaClass*TInhaltsknotenclass;

class DELPHICLASS TInhaltskante;
class PASCALIMPLEMENTATION TInhaltskante : public Ugraph::TKante
{
    typedef Ugraph::TKante inherited;

private:
    Graphics::TColor Farbe_;
    TPenStyle Stil_;
    int Weite_;
    char Typ_;
    System::AnsiString Inhalt_;
    char __fastcall Welchertyp(void);
    void __fastcall Setzetyp(char Typ);
    int __fastcall Welcheweite(void);
    void __fastcall SetzeWeite(int Weite);
    Graphics::TColor __fastcall WelcheFarbe(void);
    void __fastcall SetzeFarbe(Graphics::TColor F);
    Graphics::TPenStyle __fastcall WelcherStil(void);
    void __fastcall SetzeStil(Graphics::TPenStyle T);

public:
    __fastcall virtual TInhaltskante(void);
    HIDESBASE void __fastcall Free(void);
    HIDESBASE void __fastcall Freeall(void);
    virtual Classes::TStringList* __fastcall
Wertlisteschreiben(void);
    virtual void __fastcall Wertlistelesen(void);
```

```
    __property char Typ = {read=Welchertyp, write=Setzetyp,
nodefault};
    __property int Weite = {read=Welcheweite, write=SetzeWeite,
nodefault};
    __property Graphics::TColor Farbe = {read=WelcheFarbe,
write=SetzeFarbe, nodefault};
    __property Graphics::TPenStyle Stil = {read=WelcherStil,
write=SetzeStil, nodefault};
    bool __fastcall MausklickaufKante(int X, int Y);
    void __fastcall ZeichneKante(Graphics::TCanvas* Flaeche);
    void __fastcall ZeichneDruckKante(Graphics::TCanvas* Flaeche,
int Faktor);
    void __fastcall Kantezeichnen(Graphics::TCanvas* Flaeche,
bool Demo, int Pausenzeit);
public:
    /* TList.Destroy */ __fastcall virtual ~TInhaltskante(void) {
}

};

typedef System::TMetaClass*TInhaltskanteclass;

typedef System::TMetaClass*TInhaltsgraphclass;

class DELPHICLASS TInhaltsgraph;
class PASCALIMPLEMENTATION TInhaltsgraph : public Ugraph::TGraph
{
    typedef Ugraph::TGraph inherited;

private:
    int Knotenwertposition_;
    int Kantenwertposition_;
    bool Demo_;
    int Pausenzeit_;
    bool Zustand_;
    bool Stop_;
    int Knotengenauigkeit_;
    int Kantengenauigkeit_;
    int Radius_;
    int Liniendicke_;
    bool Graphistgespeichert_;
    System::TMetaClass*Inhaltsknotenclass_;
    System::TMetaClass*Inhaltskanteclass_;
    Ugraph::TKnotenliste* MomentaneKnotenliste_;
    Ugraph::TKantenliste* MomentaneKantenliste_;
    System::AnsiString Dateiname_;
    TInhaltsknoten* K1_;
    TInhaltsknoten* K2_;
    TInhaltsknoten* K3_;
    TInhaltsknoten* K4_;
```

```
    int __fastcall WelcheKnotenwertposition(void);
    void __fastcall SetzeKnotenwertposition(int P);
    int __fastcall WelcheKantenwertposition(void);
    void __fastcall SetzeKantenwertposition(int P);
    int __fastcall WelcheWartezeit(void);
    void __fastcall SetzeWartezeit(int Wz);
    bool __fastcall WelcherDemomodus(void);
    void __fastcall SetzeDemomodus(bool D);
    bool __fastcall WelcherEingabezustand(void);
    void __fastcall SetzeEingabezustand(bool Ezsd);
    bool __fastcall WelcherStopzustand(void);
    void __fastcall SetzeStopzustand(bool Stop);
    int __fastcall WelcheKnotengenauigkeit(void);
    void __fastcall SetzeKnotengenauigkeit(int G);
    int __fastcall WelcheKantengenauigkeit(void);
    void __fastcall SetzeKantengenauigkeit(int G);
    int __fastcall WelcherKnotenradius(void);
    void __fastcall SetzeKnotenradius(int R);
    int __fastcall WelcheLiniendicke(void);
    void __fastcall SetzeLiniendicke(int D);
    bool __fastcall IstGraphgespeichert(void);
    void __fastcall SetzeGraphgespeichert(bool Gesp);
    System::AnsiString __fastcall WelcherDateiname();
    void __fastcall SetzeDateiname(System::AnsiString N);
    void __fastcall SetzeletztenMausklickKnoten(TInhaltsknoten*
Kno);
    TInhaltsknoten* __fastcall
WelcherletzteMausklickKnoten(void);
    System::TMetaClass* __fastcall WelcheKnotenclass(void);
    void __fastcall SetzeKnotenclass(System::TMetaClass*
Inhaltsclass);
    System::TMetaClass* __fastcall WelcheKantenclass(void);
    void __fastcall SetzeKantenclass(System::TMetaClass*
Inhaltsclass);
    HIDESBASE Ugraph::TKnotenliste* __fastcall
WelcheKnotenliste(void);
    HIDESBASE void __fastcall
SetzeKnotenliste(Ugraph::TKnotenliste* L);
    HIDESBASE Ugraph::TKantenliste* __fastcall
WelcheKantenliste(void);
    HIDESBASE void __fastcall
SetzeKantenliste(Ugraph::TKantenliste* L);
    TInhaltsknoten* __fastcall WelcherK1(void);
    void __fastcall SetzeK1(TInhaltsknoten* Kno);
    __property TInhaltsknoten* K1 = {read=WelcherK1,
write=SetzeK1};
    TInhaltsknoten* __fastcall WelcherK2(void);
    void __fastcall SetzeK2(TInhaltsknoten* Kno);
    __property TInhaltsknoten* K2 = {read=WelcherK2,
write=SetzeK2};
```

```
    TInhaltsknoten* __fastcall WelcherK3(void);
    void __fastcall SetzeK3(TInhaltsknoten* Kno);
    __property TInhaltsknoten* K3 = {read=WelcherK3,
write=SetzeK3};
    TInhaltsknoten* __fastcall WelcherK4(void);
    void __fastcall SetzeK4(TInhaltsknoten* Kno);
    __property TInhaltsknoten* K4 = {read=WelcherK4,
write=SetzeK4};

public:
    __fastcall virtual TInhaltsgraph(void);
    HIDESBASE void __fastcall Free(void);
    HIDESBASE void __fastcall Freeall(void);
    __property Ugraph::TKnotenliste* MomentaneKnotenliste =
{read=WelcheKnotenliste, write=SetzeKnotenliste
        };
    __property Ugraph::TKantenliste* MomentaneKantenliste =
{read=WelcheKantenliste, write=SetzeKantenliste
        };
    __property System::TMetaClass* Inhaltsknotenclass =
{read=WelcheKnotenclass, write=SetzeKnotenclass
        };
    __property System::TMetaClass* Inhaltskanteclass =
{read=WelcheKantenclass, write=SetzeKantenclass}
        ;
    __property int Knotenwertposition =
{read=WelcheKnotenwertposition, write=SetzeKnotenwertposition,
        nodefault};
    __property int Kantenwertposition =
{read=WelcheKantenwertposition, write=SetzeKantenwertposition,
        nodefault};
    __property int Pausenzeit = {read=WelcheWartezeit,
write=SetzeWartezeit, nodefault};
    __property bool Demo = {read=WelcherDemomodus,
write=SetzeDemomodus, nodefault};
    __property bool Zustand = {read=WelcherEingabezustand,
write=SetzeEingabezustand, nodefault};
    __property bool Stop = {read=WelcherStopzustand,
write=SetzeStopzustand, nodefault};
    __property int Knotengenauigkeit =
{read=WelcheKnotengenauigkeit, write=SetzeKnotengenauigkeit,
nodefault
        };
    __property int Kantengenauigkeit =
{read=WelcheKantengenauigkeit, write=SetzeKantengenauigkeit,
nodefault
        };
    __property int Radius = {read=WelcherKnotenradius,
write=SetzeKnotenradius, nodefault};
    __property int Liniendicke = {read=WelcheLiniendicke,
```

```
write=SetzeLiniendicke, nodefault};
    __property bool Graphistgespeichert =
{read=IstGraphgespeichert, write=SetzeGraphgespeichert,
nodefault
        };
    __property System::AnsiString Dateiname =
{read=WelcherDateiname, write=SetzeDateiname};
    __property TInhaltsknoten* LetzterMausklickknoten =
{read=WelcherletzteMausklickKnoten,
write=SetzeletztenMausklickKnoten
        };
    virtual Classes::TStringList* __fastcall
Wertlisteschreiben(void);
    virtual void __fastcall Wertlistelesen(void);
    void __fastcall Demopause(void);
    void __fastcall FuegeKnotenein(TInhaltsknoten* Kno);
    void __fastcall FuegeKanteein(TInhaltsknoten* Kno1,
TInhaltsknoten* Kno2, bool Gerichtet, TInhaltskante*
        Ka);
    HIDESBASE void __fastcall EinfuegenKante(TInhaltskante* Ka);
    void __fastcall LoescheKante(TInhaltsknoten* Kno1,
TInhaltsknoten* Kno2);
    void __fastcall LoescheInhaltskante(TInhaltskante* Ka);
    void __fastcall ZeichneGraph(Graphics::TCanvas* Flaeche);
    void __fastcall ZeichneDruckGraph(Graphics::TCanvas* Flaeche,
int Faktor);
    void __fastcall Graphzeichnen(Graphics::TCanvas* Flaeche,
Stdctrls::TLabel* Ausgabe, Ulist::TWert Wert
        , Classes::TStringList* SListe, bool Demo, int Pausen-
zeit, int Kantengenauigkeit);
    void __fastcall FaerbeGraph(Graphics::TColor F,
Graphics::TPenStyle T);
    bool __fastcall FindezuKoordinatendenKnoten(int &A, int &B,
TInhaltsknoten* &Kno);
    bool __fastcall FindedenKnotenzuKoordinaten(int &A, int &B,
TInhaltsknoten* &Kno);
    TInhaltsknoten* __fastcall Graphknoten(TInhaltsknoten* Kno);
    virtual void __fastcall SpeichereGraph(System::AnsiString
Dateiname);
    virtual void __fastcall LadeGraph(System::AnsiString Dateina-
me);
    virtual void __fastcall EingabeKante(TInhaltskante* &Ka, bool
&Aus, bool &Abbruch);
    bool __fastcall Kantezeichnen(int X, int Y);
    bool __fastcall Inhaltskanteloeschen(int X, int Y);
    bool __fastcall Knoteninhaltzeigen(int X, int Y);
    bool __fastcall Knotenverschieben(int X, int Y);
    bool __fastcall Kanteverschieben(int X, int Y);
    virtual void __fastcall EditiereKnoten(TInhaltsknoten* &Kno,
bool &Abbruch);
```

```
    bool __fastcall Knoteneditieren(int X, int Y);
    virtual void __fastcall EditiereKante(TInhaltskante* &Ka,
bool &Aus, bool &Abbruch);
    bool __fastcall Kanteeditieren(int X, int Y);
    bool __fastcall Kanteninhaltzeigen(int X, int Y);
    virtual void __fastcall EingabeKnoten(TInhaltsknoten* &Kno,
bool &Abbruch);
    bool __fastcall Knotenzeichnen(int X, int Y);
    bool __fastcall ZweiKnotenauswaehlen(int X, int Y,
TInhaltsknoten* &Kno1, TInhaltsknoten* &Kno2, bool
        &Gefunden);
    bool __fastcall Inhaltsknotenloeschen(int X, int Y);
    TInhaltsgraph* __fastcall
InhaltsKopiedesGraphen(System::TMetaClass* Inhaltsgraphclass,
System::TMetaClass*
        Inhaltsknotenclass, System::TMetaClass*
Inhaltskanteclass, bool UngerichteterGraph);
    int __fastcall AnzahlTypKanten(char Typ);
    int __fastcall AnzahlTypKnoten(char Typ);
    int __fastcall AnzahlBruecken(Classes::TStringList* &SListe,
Stdctrls::TLabel* Ausgabe, Graphics::TCanvas*
        Flaeche);
    Classes::TStringList* __fastcall AlleKnotenbestimmen(void);
    int __fastcall AnzahlKnotenkleinsterKreis(System::AnsiString
&St, Graphics::TCanvas* Flaeche);
    int __fastcall AnzahlKnotengroesterKreis(System::AnsiString
&St, Graphics::TCanvas* Flaeche);
    int __fastcall KreisefesterLaenge(int Laenge,
Classes::TStringList* &SListe, Graphics::TCanvas*
        Flaeche, Stdctrls::TLabel* Ausgabe);
    Classes::TStringList* __fastcall AlleKantenbestimmen(void);
public:
    /* TObject.Destroy */ __fastcall virtual ~TInhaltsgraph(void)
{ }

};

//— var, const, procedure ——————————————————
extern PACKAGE Extended __fastcall Bewertung(System::TObject*
Ob);
extern PACKAGE System::AnsiString __fastcall
ErzeugeKnotenstring(System::TObject* Ob);
extern PACKAGE System::AnsiString __fastcall
ErzeugeKantenstring(System::TObject* Ob);
extern PACKAGE void __fastcall LoescheBild(TInhaltsgraph* &G,
Forms::TForm* &Oberflaeche);

}   /* namespace Uinhgrph */
#if !defined(NO_IMPLICIT_NAMESPACE_USE)
using namespace Uinhgrph;
```

```
#endif
//— end unit ————————————————————————————
#endif  // UinhGrph
```

Quellcode UKANTE.HPP:

```
// Borland C++ Builder
// Copyright (c) 1995, 1998 by Borland International
// All rights reserved

// (DO NOT EDIT: machine generated header) 'UKante.pas' rev:
3.00

#ifndef UKanteHPP
#define UKanteHPP
#include <StdCtrls.hpp>
#include <Dialogs.hpp>
#include <Forms.hpp>
#include <Controls.hpp>
#include <Graphics.hpp>
#include <Classes.hpp>
#include <Messages.hpp>
#include <Windows.hpp>
#include <SysUtils.hpp>
#include <SysInit.hpp>
#include <System.hpp>

//— user supplied ———————————————————————————

namespace Ukante
{
//— type declarations ———————————————————————
class DELPHICLASS TKantenform;
class PASCALIMPLEMENTATION TKantenform : public Forms::TForm
{
    typedef Forms::TForm inherited;

__published:
    Stdctrls::TCheckBox* Checkausgehend;
    Stdctrls::TCheckBox* Checkeingehend;
    Stdctrls::TCheckBox* CheckInteger;
    Stdctrls::TCheckBox* CheckReal;
    Stdctrls::TEdit* Inhalt;
    Stdctrls::TEdit* Weite;
    Stdctrls::TScrollBar* ScrollBar;
    Stdctrls::TButton* OK;
    Stdctrls::TButton* Abbruch;
    Stdctrls::TLabel* Textausgehend;
    Stdctrls::TLabel* TextEingehend;
```

```
    Stdctrls::TLabel* TextInteger;
    Stdctrls::TLabel* TextReal;
    Stdctrls::TLabel* TextWeite;
    Stdctrls::TLabel* TextInhalt;
    void __fastcall OKClick(System::TObject* Sender);
    void __fastcall InhaltChange(System::TObject* Sender);
    void __fastcall CheckIntegerClick(System::TObject* Sender);
    void __fastcall CheckRealClick(System::TObject* Sender);
    void __fastcall InhaltKeyPress(System::TObject* Sender, char
&Key);
    void __fastcall FormActivate(System::TObject* Sender);
    void __fastcall AbbruchClick(System::TObject* Sender);
    void __fastcall WeiteKeyPress(System::TObject* Sender, char
&Key);
    void __fastcall ScrollBarChange(System::TObject* Sender);
    void __fastcall WeiteChange(System::TObject* Sender);

private:
    bool Kanteein_;
    bool Kanteaus_;
    bool KantenInteger_;
    bool Kantenreal_;
    System::AnsiString Kanteninhalt_;
    int Kantenweite_;
    bool Kantenabbruch_;
    bool __fastcall WelchesKantenein(void);
    void __fastcall SetzeKantenein(bool Ke);
    bool __fastcall WelchesKantenaus(void);
    void __fastcall SetzeKantenaus(bool Ka);
    bool __fastcall WelchesKantenInteger(void);
    void __fastcall SetzeKantenInteger(bool Ki);
    bool __fastcall WelchesKantenreal(void);
    void __fastcall SetzeKantenreal(bool Kr);
    System::AnsiString __fastcall WelcherKanteninhalt();
    void __fastcall SetzeKanteninhalt(System::AnsiString Ki);
    int __fastcall WelcheKantenweite(void);
    void __fastcall SetzeKantenweite(int Kw);
    bool __fastcall WelcherKantenabbruch(void);
    void __fastcall SetzeKantenabbruch(bool Ab);

public:
    __fastcall TKantenform(Classes::TComponent* AOwner);
    __property bool Kanteein = {read=WelchesKantenein,
write=SetzeKantenein, nodefault};
    __property bool Kanteaus = {read=WelchesKantenaus,
write=SetzeKantenaus, nodefault};
    __property bool KantenInteger = {read=WelchesKantenInteger,
write=SetzeKantenInteger, nodefault};
    __property bool Kantenreal = {read=WelchesKantenreal,
write=SetzeKantenreal, nodefault};
```

```
    __property System::AnsiString Kanteninhalt =
{read=WelcherKanteninhalt, write=SetzeKanteninhalt};
    __property int Kantenweite = {read=WelcheKantenweite,
write=SetzeKantenweite, nodefault};
    __property bool Kantenabbruch = {read=WelcherKantenabbruch,
write=SetzeKantenabbruch, nodefault};
public:

    /* TCustomForm.CreateNew */ __fastcall
TKantenform(Classes::TComponent* AOwner, int Dummy) : Forms::
        TForm(AOwner, Dummy) { }
    /* TCustomForm.Destroy */ __fastcall virtual
~TKantenform(void) { }

public:
    /* TWinControl.CreateParented */ __fastcall TKantenform(HWND
ParentWindow) : Forms::TForm(ParentWindow
        ) { }

};

//— var, const, procedure ——————————————————
extern PACKAGE TKantenform* Kantenform;

}   /* namespace Ukante */
#if !defined(NO_IMPLICIT_NAMESPACE_USE)
using namespace Ukante;
#endif
//— end unit ———————————————————————————————
#endif  // Ukante
```

Quellcode UAUSGABE.HPP:

```
// Borland C++ Builder
// Copyright (c) 1995, 1998 by Borland International
// All rights reserved

// (DO NOT EDIT: machine generated header) 'UAusgabe.pas' rev:
3.00

#ifndef UAusgabeHPP
#define UAusgabeHPP
#include <Printers.hpp>
#include <Clipbrd.hpp>
#include <Grids.hpp>
#include <StdCtrls.hpp>
#include <Menus.hpp>
#include <Dialogs.hpp>
#include <Forms.hpp>
```

```
#include <Controls.hpp>
#include <Graphics.hpp>
#include <Classes.hpp>
#include <Messages.hpp>
#include <Windows.hpp>
#include <SysUtils.hpp>
#include <SysInit.hpp>
#include <System.hpp>

//— user supplied ——————————————————————

namespace Uausgabe
{
//— type declarations ——————————————————————
class DELPHICLASS TAusgabeform;
class PASCALIMPLEMENTATION TAusgabeform : public Forms::TForm
{
    typedef Forms::TForm inherited;

__published:
    Grids::TStringGrid* Gitternetz;
    Menus::TMainMenu* MainMenu;
    Menus::TMenuItem* Ende;
    Menus::TMenuItem* Kopieren;
    Menus::TMenuItem* Drucken;
    Dialogs::TPrintDialog* PrintDialog;
    void __fastcall EndeClick(System::TObject* Sender);
    void __fastcall FormPaint(System::TObject* Sender);
    void __fastcall KopierenClick(System::TObject* Sender);
    void __fastcall DruckenClick(System::TObject* Sender);

private:
    Stdctrls::TListBox* Listenbox_;
    Stdctrls::TListBox* __fastcall WelcheListbox(void);
    void __fastcall SetzeListbox(Stdctrls::TListBox* Lb);

public:
    __fastcall TAusgabeform(Classes::TComponent* AOwner);
    __property Stdctrls::TListBox* Listenbox =
{read=WelcheListbox, write=SetzeListbox};
public:
    /* TCustomForm.CreateNew */ __fastcall
TAusgabeform(Classes::TComponent* AOwner, int Dummy) : Forms::
        TForm(AOwner, Dummy) { }
    /* TCustomForm.Destroy */ __fastcall virtual
~TAusgabeform(void) { }

public:
    /* TWinControl.CreateParented */ __fastcall TAusgabeform(HWND
ParentWindow) : Forms::TForm(ParentWindow;
```

```
//— var, const, procedure ——————————————————
extern PACKAGE TAusgabeform* Ausgabeform;

}   /* namespace Uausgabe */
#if !defined(NO_IMPLICIT_NAMESPACE_USE)
using namespace Uausgabe;
#endif
//— end unit ———————————————————————————
#endif  // UAusgabe
```

Quellcode UKNOTEN.HPP:

```
// Borland C++ Builder
// Copyright (c) 1995, 1998 by Borland International
// All rights reserved

// (DO NOT EDIT: machine generated header) 'UKnoten.pas' rev:
3.00

#ifndef UKnotenHPP
#define UKnotenHPP
#include <Buttons.hpp>
#include <Printers.hpp>
#include <Clipbrd.hpp>
#include <Forms.hpp>
#include <Graphics.hpp>
#include <Messages.hpp>
#include <Windows.hpp>
#include <SysUtils.hpp>
#include <Classes.hpp>
#include <Menus.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <UAusgabe.hpp>
#include <UKante.hpp>
#include <UGraph.hpp>
#include <UInhGrph.hpp>
#include <UList.hpp>
#include <SysInit.hpp>
#include <System.hpp>

//— user supplied ——————————————————————————

namespace Uknoten
{
//— type declarations ———————————————————————
class DELPHICLASS TKnotenform;
```

```
class PASCALIMPLEMENTATION TKnotenform : public Forms::TForm
{
    typedef Forms::TForm inherited;

__published:
    Extctrls::TPaintBox* PaintBox;
    Extctrls::TPanel* Panel;
    Stdctrls::TButton* Button;
    Stdctrls::TListBox* ListBox;
    Extctrls::TImage* Image;
    Stdctrls::TLabel* Ausgabe1;
    Stdctrls::TLabel* Ausgabe2;
    Stdctrls::TEdit* Eingabe;
    Dialogs::TOpenDialog* OpenDialog;
    Dialogs::TSaveDialog* SaveDialog;
    Dialogs::TPrintDialog* PrintDialog;
    Buttons::TBitBtn* BitBtn1;
    Buttons::TBitBtn* BitBtn2;
    Buttons::TBitBtn* BitBtn3;
    Buttons::TBitBtn* BitBtn4;
    Buttons::TBitBtn* BitBtn5;
    Buttons::TBitBtn* BitBtn6;
    Buttons::TBitBtn* BitBtn7;
    Buttons::TBitBtn* BitBtn8;
    Buttons::TBitBtn* BitBtn9;
    Buttons::TBitBtn* BitBtn10;
    Buttons::TBitBtn* BitBtn11;
    Menus::TMainMenu* MainMenu;
    Menus::TMenuItem* Datei;
    Menus::TMenuItem* NeueKnoten;
    Menus::TMenuItem* Graphladen;
    Menus::TMenuItem* Graphhinzufgen;
    Menus::TMenuItem* Graphspeichern;
    Menus::TMenuItem* Graphspeichernunter;
    Menus::TMenuItem* Graphdrucken;
    Menus::TMenuItem* Quit;
    Menus::TMenuItem* Bild;
    Menus::TMenuItem* Ergebniszeichnen;
    Menus::TMenuItem* Bildzeichnen;
    Menus::TMenuItem* Bildkopieren;
    Menus::TMenuItem* Bildwiederherstellen;
    Menus::TMenuItem* UngerichteteKanten;
    Menus::TMenuItem* Knotenradius;
    Menus::TMenuItem* GenauigkeitKnoten;
    Menus::TMenuItem* GenauigkeitKanten;
    Menus::TMenuItem* Knoten;
    Menus::TMenuItem* Knotenerzeugen;
    Menus::TMenuItem* Knotenloeschen;
    Menus::TMenuItem* Knoteneditieren;
    Menus::TMenuItem* Knotenverschieben;
```

```
    Menus::TMenuItem* Kanten;
    Menus::TMenuItem* Kantenerzeugen;
    Menus::TMenuItem* Kantenloeschen;
    Menus::TMenuItem* Kanteeditieren;
    Menus::TMenuItem* Kanteverschieben;
    Menus::TMenuItem* Eigenschaften;
    Menus::TMenuItem* AnzahlKantenundEcken;
    Menus::TMenuItem* ParallelkantenundSchlingen;
    Menus::TMenuItem* Eulerlinie;
    Menus::TMenuItem* Kreise;
    Menus::TMenuItem* AnzahlBruecken;
    Menus::TMenuItem* Knotenanzeigen;
    Menus::TMenuItem* Kantenanzeigen;
    Menus::TMenuItem* Ausgabe;
    Menus::TMenuItem* Ausgabefenster;
    Menus::TMenuItem* Abbruch;
    Menus::TMenuItem* Demo;
    Menus::TMenuItem* Pausenzeit;
    Menus::TMenuItem* Hilfe;
    Menus::TMenuItem* Inhalt;
    Menus::TMenuItem* Info;
    void __fastcall Bildloeschen(void);
    void __fastcall StringlistnachListbox(Classes::TStringList*
S, Stdctrls::TListBox* &L);
    void __fastcall Menuenabled(bool Enabled);
    void __fastcall Ausgabeloeschen(bool Onlyfirst);
    void __fastcall Fehler(void);
    void __fastcall FormActivate(System::TObject* Sender);
    void __fastcall PaintBoxPaint(System::TObject* Sender);
    void __fastcall PaintBoxMouseDown(System::TObject* Sender,
Controls::TMouseButton Button, Classes::TShiftState
        Shift, int X, int Y);
    void __fastcall PaintBoxDblClick(System::TObject* Sender);
    void __fastcall PaintBoxMouseMove(System::TObject* Sender,
Classes::TShiftState Shift, int X, int Y
        );
    void __fastcall PanelClick(System::TObject* Sender);
    void __fastcall PanelDblClick(System::TObject* Sender);
    void __fastcall PanelMouseDown(System::TObject* Sender,
Controls::TMouseButton Button, Classes::TShiftState
        Shift, int X, int Y);
    void __fastcall Ausgabe1Click(System::TObject* Sender);
    void __fastcall Ausgabe1DblClick(System::TObject* Sender);
    void __fastcall Ausgabe1MouseDown(System::TObject* Sender,
Controls::TMouseButton Button, Classes::TShiftState
        Shift, int X, int Y);
    void __fastcall Ausgabe2Click(System::TObject* Sender);
    void __fastcall Ausgabe2DblClick(System::TObject* Sender);
    void __fastcall Ausgabe2MouseDown(System::TObject* Sender,
Controls::TMouseButton Button, Classes::TShiftState
```

```
        Shift, int X, int Y);
    void __fastcall FormCloseQuery(System::TObject* Sender, bool
&CanClose);
    void __fastcall KnotenerzeugenMouseDown(System::TObject* Sen-
der, Controls::TMouseButton Button, Classes::TShiftState
        Shift, int X, int Y);
    void __fastcall KnotenloeschenMousedown(System::TObject* Sen-
der, Controls::TMouseButton Button, Classes::TShiftState
        Shift, int X, int Y);
    void __fastcall KnoteneditierenMouseDown(System::TObject*
Sender, Controls::TMouseButton Button, Classes::TShiftState
        Shift, int X, int Y);
    void __fastcall KnotenverschiebenaufMouseUp(System::TObject*
Sender, Controls::TMouseButton Button,
        Classes::TShiftState Shift, int X, int Y);
    void __fastcall KnotenverschiebenabMousedown(System::TObject*
Sender, Controls::TMouseButton Button
        , Classes::TShiftState Shift, int X, int Y);
    void __fastcall KanteerzeugenMouseDown(System::TObject* Sen-
der, Controls::TMouseButton Button, Classes::TShiftState
        Shift, int X, int Y);
    void __fastcall KanteloeschenMouseDown(System::TObject* Sen-
der, Controls::TMouseButton Button, Classes::TShiftState
        Shift, int X, int Y);
    void __fastcall KanteeditierenMouseDown(System::TObject* Sen-
der, Controls::TMouseButton Button, Classes::TShiftState
        Shift, int X, int Y);
    void __fastcall KanteverschiebenaufMouseup(System::TObject*
Sender, Controls::TMouseButton Button,
        Classes::TShiftState Shift, int X, int Y);
    void __fastcall KanteverschiebenabMouseDown(System::TObject*
Sender, Controls::TMouseButton Button,
        Classes::TShiftState Shift, int X, int Y);
    void __fastcall PanelDownMouse(System::TObject* Sender,
Controls::TMouseButton Button, Classes::TShiftState
        Shift, int X, int Y);
    void __fastcall NeueKnotenClick(System::TObject* Sender);
    void __fastcall GraphladenClick(System::TObject* Sender);
    void __fastcall GraphhinzufgenClick(System::TObject* Sender);
    void __fastcall GraphspeichernClick(System::TObject* Sender);
    void __fastcall GraphspeichernunterClick(System::TObject*
Sender);
    void __fastcall GraphdruckenClick(System::TObject* Sender);
    void __fastcall QuitClick(System::TObject* Sender);
    void __fastcall ErgebniszeichnenClick(System::TObject* Sen-
der);
    void __fastcall BildzeichnenClick(System::TObject* Sender);
    void __fastcall BildkopierenClick(System::TObject* Sender);
    void __fastcall BildwiederherstellenClick(System::TObject*
Sender);
```

```
    void __fastcall UngerichteteKantenClick(System::TObject* Sen-
der);
    void __fastcall KnotenradiusClick(System::TObject* Sender);
    void __fastcall GenauigkeitKnotenClick(System::TObject* Sen-
der);
    void __fastcall GenauigkeitKantenClick(System::TObject* Sen-
der);
    void __fastcall KnotenerzeugenClick(System::TObject* Sender);
    void __fastcall KnotenloeschenClick(System::TObject* Sender);
    void __fastcall KnoteneditierenClick(System::TObject* Sen-
der);
    void __fastcall KnotenverschiebenClick(System::TObject* Sen-
der);
    void __fastcall KantenerzeugenClick(System::TObject* Sender);
    void __fastcall KantenloeschenClick(System::TObject* Sender);
    void __fastcall KanteeditierenClick(System::TObject* Sender);
    void __fastcall KanteverschiebenClick(System::TObject* Sen-
der);
    void __fastcall AnzahlKantenundEckenClick(System::TObject*
Sender);
    void __fastcall
ParallelkantenundSchlingenClick(System::TObject* Sender);
    void __fastcall EulerlinieClick(System::TObject* Sender);
    void __fastcall KreiseClick(System::TObject* Sender);
    void __fastcall AnzahlBrueckenClick(System::TObject* Sender);
    void __fastcall KnotenanzeigenClick(System::TObject* Sender);
    void __fastcall KantenanzeigenClick(System::TObject* Sender);
    void __fastcall AusgabefensterClick(System::TObject* Sender);
    void __fastcall AbbruchClick(System::TObject* Sender);
    void __fastcall DemoClick(System::TObject* Sender);
    void __fastcall PausenzeitClick(System::TObject* Sender);
    void __fastcall InhaltClick(System::TObject* Sender);
    void __fastcall InfoClick(System::TObject* Sender);
    void __fastcall InitBitBtnMenu(void);
    void __fastcall BitBtn1Click(System::TObject* Sender);
    void __fastcall BitBtn2Click(System::TObject* Sender);
    void __fastcall BitBtn3Click(System::TObject* Sender);
    void __fastcall BitBtn4Click(System::TObject* Sender);
    void __fastcall BitBtn5Click(System::TObject* Sender);
    void __fastcall BitBtn6Click(System::TObject* Sender);
    void __fastcall BitBtn7Click(System::TObject* Sender);
    void __fastcall BitBtn8Click(System::TObject* Sender);
    void __fastcall BitBtn9Click(System::TObject* Sender);
    void __fastcall BitBtn10Click(System::TObject* Sender);
    void __fastcall BitBtn11Click(System::TObject* Sender);

private:
    bool Aktiv_;
    Uinhgrph::TInhaltsgraph* Graph_;
    Uinhgrph::TInhaltsgraph* GraphH_;
```

```
    Uinhgrph::TInhaltsgraph* GraphK_;
    Uinhgrph::TInhaltsgraph* GraphZ_;
    bool __fastcall Istaktiv(void);
    void __fastcall Setzeaktiv(bool A);
    Uinhgrph::TInhaltsgraph* __fastcall WelcherGraph(void);
    void __fastcall SetzeGraph(Uinhgrph::TInhaltsgraph* Gr);
    Uinhgrph::TInhaltsgraph* __fastcall WelcherGraphH(void);
    void __fastcall SetzeGraphH(Uinhgrph::TInhaltsgraph* Gr);
    Uinhgrph::TInhaltsgraph* __fastcall WelcherGraphK(void);
    void __fastcall SetzeGraphK(Uinhgrph::TInhaltsgraph* Gr);
    Uinhgrph::TInhaltsgraph* __fastcall WelcherGraphZ(void);
    void __fastcall SetzeGraphZ(Uinhgrph::TInhaltsgraph* Gr);
    HIDESBASE MESSAGE void __fastcall
WMMenuselect(Messages::TWMMenuSelect &Message);

public:
    __property bool Aktiv = {read=Istaktiv, write=Setzeaktiv,
nodefault};
    __property Uinhgrph::TInhaltsgraph* Graph =
{read=WelcherGraph, write=SetzeGraph};
    __property Uinhgrph::TInhaltsgraph* GraphH =
{read=WelcherGraphH, write=SetzeGraphH};
    __property Uinhgrph::TInhaltsgraph* GraphK =
{read=WelcherGraphK, write=SetzeGraphK};
    __property Uinhgrph::TInhaltsgraph* GraphZ =
{read=WelcherGraphZ, write=SetzeGraphZ};
public:
    /* TCustomForm.Create */ __fastcall virtual
TKnotenform(Classes::TComponent* AOwner) : Forms::TForm(
        AOwner) { }
    /* TCustomForm.CreateNew */ __fastcall
TKnotenform(Classes::TComponent* AOwner, int Dummy) : Forms::
        TForm(AOwner, Dummy) { }
    /* TCustomForm.Destroy */ __fastcall virtual
~TKnotenform(void) { }

public:
    /* TWinControl.CreateParented */ __fastcall TKnotenform(HWND
ParentWindow) : Forms::TForm(ParentWindow
        ) { }

};

//— var, const, procedure ——————————————————
extern PACKAGE TKnotenform* Knotenform;

}   /* namespace Uknoten */
#if !defined(NO_IMPLICIT_NAMESPACE_USE)
using namespace Uknoten;
#endif
```

```
//— end unit ————————————————————————————
#endif  // UKnoten
```

Quellcode UPFAD.HPP:

```
#ifndef UPfadHPP
#define UPfadHPP
#include <Printers.hpp>
#include <Clipbrd.hpp>
#include <Forms.hpp>
#include <Graphics.hpp>
#include <Messages.hpp>
#include <Windows.hpp>
#include <SysUtils.hpp>
#include <Classes.hpp>
#include <Menus.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <UKante.hpp>
#include <UGraph.hpp>
#include <UInhGrph.hpp>
#include <UList.hpp>
#include <SysInit.hpp>
#include <System.hpp>

namespace UPfad
{
 class TPfadknoten : public Uinhgrph::TInhaltsknoten
 {
   void _fastcall GehezuallenNachbarknoten(TKnoten *Kno,TKante*
Ka);
   public:
   __fastcall virtual TPfadknoten(void);
   void __fastcall  Free(void);
   void __fastcall  Freeall(void);
   void __fastcall ErzeugeallePfade(void);
   __fastcall virtual ~TPfadknoten(void){};
  };

  class  TPfadgraph : public Uinhgrph::TInhaltsgraph
  {
   typedef Uinhgrph::TInhaltsgraph inherited;
   public:
     __fastcall virtual TPfadgraph(void);
     void __fastcall  Free(void);
     void __fastcall  Freeall(void);
     void __fastcall AllePfadevoneinemKnotenbestimmen
```

```
    (int X, int Y, Stdctrls::TLabel* Ausgabe,
    Classes::TStringList*
    &SListe, Graphics::TCanvas* Flaeche);
    __fastcall virtual ~TPfadgraph(void){};
   };
}

#if !defined(NO_IMPLICIT_NAMESPACE_USE)
using namespace UPfad;
#endif
#endif
```

Quellcode UPFAD.CPP:

```
#include <vcl.h>
#include „UPfad.hpp“

#pragma hdrstop
#pragma package(smart_init)

TKantenliste* MomentaneKantenliste;

__fastcall  TPfadknoten::TPfadknoten(void):TInhaltsknoten()
{
}

void __fastcall TPfadknoten::Free(void){
  Free();
}

void __fastcall TPfadknoten::Freeall(void){
  Freeall();
}

void _fastcall TPfadknoten::GehezuallenNachbarknoten(TKnoten*
Kno,TKante* Ka){
TObject* Ob;
int Index;
 Application->ProcessMessages();
  if (Graph->Abbruch) {goto Endproc;};
  if (!Ka->Zielknoten(Kno)->Besucht){
    Ka->Pfadrichtung=Ka->Zielknoten(Kno);
    MomentaneKantenliste->AmEndeanfuegen(Ka);
    Pfadliste->AmEndeanfuegen(MomentaneKantenliste->Kopie()
     ->Graph());
     if (Pfadliste->Anzahl()>10000){
       ShowMessage(„Mehr als 10000 Pfade!Abbruch!“);
       Graph->Abbruch=true;
        goto Endproc;
    }
```

```
    Ka->Zielknoten(Kno)->Besucht=true;
    if (!Ka->Zielknoten(Kno)->AusgehendeKantenliste->Leer()){
      for (Index=0;Index<=Ka->Zielknoten(Kno)
        ->AusgehendeKantenliste->Anzahl()-1;Index++){
        GehezuallenNachbarknoten(Ka->Zielknoten(Kno),Ka
        ->Zielknoten(Kno)->AusgehendeKantenliste->Kante(Index));
      }
    }
    MomentaneKantenliste->AmEndeloeschen(Ob);
    Ka->Zielknoten(Kno)->Besucht=false;
  }
  Endproc:
};

void _fastcall TPfadknoten::ErzeugeallePfade(void)
{
int Index;
  MomentaneKantenliste=new TKantenliste();
  Graph->Pfadlistenloeschen();
  Graph->LoescheKnotenbesucht();
  Besucht=true;
  if (!AusgehendeKantenliste->Leer()){
    for (Index=0;Index<=AusgehendeKantenliste->Anzahl()-1;
     Index++){
      GehezuallenNachbarknoten(this,AusgehendeKantenliste
       ->Kante(Index));
    }
  }
  MomentaneKantenliste->Free();
  MomentaneKantenliste=0;
}

__fastcall  TPfadgraph::TPfadgraph(void):TInhaltsgraph()
{
}

void __fastcall TPfadgraph::Free(void){
  Free();
}

void __fastcall TPfadgraph::Freeall(void){
  Freeall();
}

void __fastcall TPfadgraph::AllePfadevoneinemKnotenbestimmen(int
X, int Y, Stdctrls::TLabel* Ausgabe, Classes::TStringList*
&SListe, Graphics::TCanvas* Flaeche){
TPfadknoten* Kno;
  if (!Leer()){
    if (!FindezuKoordinatendenKnoten(X,Y,(TInhaltsknoten*)Kno))
```

```
{
      Kno=(TPfadknoten*)Anfangsknoten();
    }
    Kno->ErzeugeallePfade();
    if (Kno->Pfadliste->Leer()){
      ShowMessage(„Keine Pfade“);
    }
    else {
      Kno->AnzeigePfadliste(Flaeche,Ausgabe,SListe,true,true);
    }
  }
  this->Pfadlistenloeschen();
}
```

Quellcode UFORM.H:

```
#ifndef UFormH
#define UFormH

#include <vcl\vcl.h>
#include <vcl\sysutils.hpp>
#include <vcl\SysDefs.h>
#include <SysDefs.h>
#include <vcl\controls.hpp>
#include <vcl\classes.hpp>
#include <vcl\forms.hpp>
#include <vcl.h>
#include <vcl\SysDefs.h>
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include „UKNOTEN.hpp“
#include <Buttons.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
#include <Menus.hpp>
#include <UKante.hpp>
#include <UGraph.hpp>
#include <UInhGrph.hpp>
#include <UPfad.h>

class TKnotenformular : public TKnotenform
{
__published:    // IDE-managed Components
    TMenuItem *Pfade1;
    TMenuItem *AllePfade1;
    void __fastcall AllePfade1Click(TObject *Sender);
```

```
private:    // User declarations
public:     // User declarations
    __fastcall TKnotenformular(TComponent* Owner);
};
extern PACKAGE TKnotenformular *Knotenformular;
#endif
```

Quellcode UFORM.CPP:

```
#include <vcl.h>
#include <stdlib.h>
#include <vcl\sysutils.hpp>
#include <vcl\controls.hpp>
#include <vcl\classes.hpp>
#include <vcl\forms.hpp>
#include <vcl\SysDefs.h>
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <UKnoten.hpp>
#include <UPfad.hpp>
#include <UForm.h>

#pragma hdrstop
#pragma package(smart_init)
#pragma link „UKNOTEN“
#pragma resource „*.dfm“

TKnotenformular *Knotenformular;

__fastcall TKnotenformular::TKnotenformular(TComponent* Owner)
    : TKnotenform(Owner)
{
}

void __fastcall TKnotenformular::AllePfade1Click(TObject *Sen-
der)
{
Classes::TStringList* SListe;
TPfadgraph* PGraph;
TPfadknoten* PKno;
{
  try {
    if (Graph->Leer()) { abort();}
    Ausgabe1->Caption=“Berechnung läuft...“;
    Ausgabe1->Refresh();
    PaintBox->OnMouseDown=PanelDownMouse;
    PaintBox->OnDblClick=0;
```

```
    PaintBox->OnMouseMove=0;

   GraphH=Graph->InhaltsKopiedesGraphen
   (__classid(TPfadgraph),__classid(TPfadknoten),
    __classid(TInhaltskante),false);
    Aktiv=false;
    Menuenabled(false);
    Bildloeschen();
    GraphH->ZeichneGraph(PaintBox->Canvas);
    SListe=new TStringList;
    (TPfadgraph*)(GraphH).AllePfadevoneinemKnotenbestimmen(X,Y,
    Ausgabe1,SListe,
    Paintbox.Canvas);
    PKno=(TPfadknoten*)Graph->LetzterMausklickknoten;
    PGraph=(TPfadgraph*)GraphH;
    PGraph->AllePfadevoneinemKnotenbestimmen(PKno->X,PKno->Y,
    Ausgabe1,SListe,PaintBox->Canvas);
    StringlistnachListbox(SListe,ListBox);
    Menuenabled(true);
    Bildloeschen();
    GraphH->ZeichneGraph(PaintBox->Canvas);
    if (Graph->Abbruch) {
    ShowMessage(„Abbruch!“);}
    Ausgabe1->Caption=“”;
  }
  catch ( ... )
   {
    Menuenabled(true);
    AbbruchClick(Sender);
    ShowMessage(„Fehler“);
    }
   }
}
```

Quellcode KPROJEKT.CPP:

```
//---------------------------------------
#include <vcl.h>
#pragma hdrstop
USERES(„KPROJEKT.res“);
USEFORMNS(„UAUSGABE.PAS“, Uausgabe, Ausgabeform);
USEUNIT(„UGRAPH.PAS“);
USEUNIT(„UINHGRPH.PAS“);
USEFORMNS(„UKANTE.PAS“, Ukante, Kantenform);
USEFORMNS(„UKNOTEN.PAS“, Uknoten, Knotenform);
USEUNIT(„ULIST.PAS“);
USEFORM(„UForm.cpp“, Knotenformular);
USEUNIT(„UPfad.cpp“);
USEFILE(„UPFAD.HPP“);
//---------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
```

```
{
    try
    {
        Application->Initialize();
        Application->CreateForm(__classid(TKnotenformular),
       &Knotenformular);
        Application->CreateForm(__classid(TKnotenform), &Knoten
      form);
        Application->CreateForm(__classid(TAusgabeform), &Aus
      gabeform);
        Application->CreateForm(__classid(TKantenform), &Kanten
      form);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    return 0;
}
//---------------------------------------
```