УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №7.1

по предмету «Основы алгоритмизации и программирования»

Вариант 3

Выполнил:

Бражалович А.И.

Гр. 351004

Проверил:

Данилова Г. В.

Минск 2024

**Задание:**

Графе задан списками инцидентности. Разработать программу, реализующую нахождение кратчайших расстояний между источником и всеми городами (алгоритм Форда-Беллмана). Граф визуализировать.

**Код программы Delphi:**

MenuUnit;

Type

TInputFormMode = (Add, Delete, Find);

Var

InputFormMode: TInputFormMode;

Implementation

{$R \*.dfm}

Uses

MenuUnit;

Procedure TEditForm.CreateParams(Var Params: TCreateParams);

Begin

Inherited;

Params.ExStyle := Params.ExStyle Or WS\_EX\_APPWINDOW;

End;

Function TEditForm.FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

Result := False;

End;

Procedure TEditForm.FormShow(Sender: TObject);

Begin

Case EditFormMode Of

Vertices:

Caption := 'Редактирование вершин';

Edges:

Caption := 'Редактирование ребер';

End;

End;

Procedure TEditForm.FormKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

Procedure TEditForm.AddButtonClick(Sender: TObject);

Begin

Visible := False;

InputFormMode := Add;

Case EditFormMode Of

Vertices:

If Graph.Count < MAX\_VERTEX Then

InputVertexForm.ShowModal

Else

Application.MessageBox

(PWideChar('Количество вершин не может превышать ' +

IntToStr(MAX\_VERTEX)), 'Ошибка', MB\_OK + MB\_ICONERROR);

Edges:

InputEdgeForm.ShowModal;

End;

Visible := True;

End;

Procedure TEditForm.DeleteButtonClick(Sender: TObject);

Begin

Visible := False;

InputFormMode := Delete;

Case EditFormMode Of

Vertices:

InputVertexForm.ShowModal;

Edges:

InputEdgeForm.ShowModal;

End;

Visible := True;

End;

Procedure TEditForm.BackButtonClick(Sender: TObject);

Begin

Close;

End;

End.

Unit EditUnit;

Type

TInputFormMode = (Add, Delete, Find);

Var

InputFormMode: TInputFormMode;

Implementation

{$R \*.dfm}

Uses

MenuUnit;

Procedure TEditForm.CreateParams(Var Params: TCreateParams);

Begin

Inherited;

Params.ExStyle := Params.ExStyle Or WS\_EX\_APPWINDOW;

End;

Function TEditForm.FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

Result := False;

End;

Procedure TEditForm.FormShow(Sender: TObject);

Begin

Case EditFormMode Of

Vertices:

Caption := 'Редактирование вершин';

Edges:

Caption := 'Редактирование ребер';

End;

End;

Procedure TEditForm.FormKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

Procedure TEditForm.AddButtonClick(Sender: TObject);

Begin

Visible := False;

InputFormMode := Add;

Case EditFormMode Of

Vertices:

If Graph.Count < MAX\_VERTEX Then

InputVertexForm.ShowModal

Else

Application.MessageBox

(PWideChar('Количество вершин не может превышать ' +

IntToStr(MAX\_VERTEX)), 'Ошибка', MB\_OK + MB\_ICONERROR);

Edges:

InputEdgeForm.ShowModal;

End;

Visible := True;

End;

Procedure TEditForm.DeleteButtonClick(Sender: TObject);

Begin

Visible := False;

InputFormMode := Delete;

Case EditFormMode Of

Vertices:

InputVertexForm.ShowModal;

Edges:

InputEdgeForm.ShowModal;

End;

Visible := True;

End;

Procedure TEditForm.BackButtonClick(Sender: TObject);

Begin

Close;

End;

End.

Unit VertexListUnit;

Type

TValue = Integer;

PVertex = ^TVertex;

TVertex = Record

Value: TValue;

Edges: TEdgeList;

Next: PVertex;

End;

TValueArray = Array Of Integer;

TVertArray = Array Of TVertex;

TWayArr = Array Of Integer;

TVertexList = Record

Head: PVertex;

Count: Integer;

Procedure Create();

Procedure Add(Const Value: TValue);

Procedure DeleteEdges(Const Value: TValue);

Procedure Delete(Const Value: TValue);

Function Find(Const Value: TValue): PVertex;

Procedure Clear();

Function GetValueArray(): TValueArray;

Function GetVertexArray(): TVertArray;

Function FindWay(StartValue: Integer): TWayArr;

End;

Const

INF = 2000000000;

MAX\_VERTEX = 32;

VertexSize = 7;

Procedure DrawVerteces(PaintBox: TPaintBox; Graph: TVertexList;

Const CenterX, CenterY, Distance: Integer; Const RotateAngle: Real);

Implementation

Procedure DrawEdge(PaintBox: TPaintBox; Const X1, Y1, X2, Y2: Integer);

Var

TrueVertexSize: Integer;

Begin

TrueVertexSize := Round(PaintBox.Width \* VertexSize / 100);

With PaintBox.Canvas Do

Begin

MoveTo(X1 + TrueVertexSize Div 2, Y1 + TrueVertexSize Div 2);

LineTo(X2 + TrueVertexSize Div 2, Y2 + TrueVertexSize Div 2);

End;

End;

Procedure DrawEdges(PaintBox: TPaintBox; Const CenterX, CenterY,

Distance: Integer; Const RotateAngle: Real; Vertex1: PVertex;

IndexNode1: Integer);

Var

CurrNode: PEdge;

Vertex2: PVertex;

IndexNode2: Integer;

Begin

CurrNode := Vertex1^.Edges.Head;

While CurrNode <> Nil Do

Begin

Vertex2 := Vertex1^.Next;

IndexNode2 := IndexNode1 + 1;

While (Vertex2 <> Nil) And (Vertex2^.Value <> CurrNode^.Value) Do

Begin

Inc(IndexNode2);

Vertex2 := Vertex2^.Next;

End;

If Vertex2 <> Nil Then

DrawEdge(PaintBox,

Round(CenterX + Distance \* Sin(IndexNode1 \* RotateAngle)),

Round(CenterY - Distance \* Cos(IndexNode1 \* RotateAngle)),

Round(CenterX + Distance \* Sin(IndexNode2 \* RotateAngle)),

Round(CenterY - Distance \* Cos(IndexNode2 \* RotateAngle)));

CurrNode := CurrNode^.Next;

End;

End;

Procedure DrawVertex(PaintBox: TPaintBox; Const X, Y: Integer;

Const Text: String);

Var

TrueVertexSize: Integer;

Begin

TrueVertexSize := Round(PaintBox.Width \* VertexSize / 100);

With PaintBox.Canvas Do

Begin

Ellipse(X, Y, X + TrueVertexSize, Y + TrueVertexSize);

TextOut(X + (TrueVertexSize - TextWidth(Text)) Div 2,

Y + (TrueVertexSize - TextHeight(Text)) Div 2, Text);

End;

End;

Procedure DrawVerteces(PaintBox: TPaintBox; Graph: TVertexList;

Const CenterX, CenterY, Distance: Integer; Const RotateAngle: Real);

Var

Angle: Real;

CurrNode: PVertex;

IndexNode: Integer;

Begin

Angle := 0;

CurrNode := Graph.Head;

IndexNode := 0;

While CurrNode <> Nil Do

Begin

DrawEdges(PaintBox, CenterX, CenterY, Distance, RotateAngle, CurrNode,

IndexNode);

DrawVertex(PaintBox, Round(CenterX + Distance \* Sin(Angle)),

Round(CenterY - Distance \* Cos(Angle)), IntToStr(CurrNode^.Value));

Angle := Angle + RotateAngle;

CurrNode := CurrNode^.Next;

Inc(IndexNode);

End;

End;

Procedure TVertexList.Create();

Begin

Head := Nil;

Count := 0;

End;

Function CreateNode(Const Value: TValue): PVertex;

Var

NewNode: PVertex;

Begin

New(NewNode);

NewNode^.Value := Value;

NewNode^.Edges.Create();

NewNode^.Next := Nil;

Result := NewNode;

End;

Procedure TVertexList.Add(Const Value: TValue);

Var

NewNode, CurrNode: PVertex;

Begin

NewNode := CreateNode(Value);

If Head = Nil Then

Head := NewNode

Else If Head^.Value > Value Then

Begin

NewNode^.Next := Head;

Head := NewNode;

End

Else

Begin

CurrNode := Head;

While (CurrNode^.Next <> Nil) And (CurrNode^.Next^.Value <= Value) Do

CurrNode := CurrNode^.Next;

NewNode^.Next := CurrNode^.Next;

CurrNode^.Next := NewNode;

End;

Inc(Count);

End;

Procedure TVertexList.DeleteEdges(Const Value: TValue);

Var

CurrNode: PVertex;

Begin

CurrNode := Head;

While CurrNode <> Nil Do

Begin

If CurrNode^.Edges.Find(Value) <> Nil Then

CurrNode^.Edges.Delete(Value);

CurrNode := CurrNode^.Next;

End;

End;

Procedure TVertexList.Delete(Const Value: TValue);

Var

TempNode, CurrNode: PVertex;

Begin

If Head^.Value = Value Then

Begin

TempNode := Head;

Head := Head^.Next;

End

Else

Begin

CurrNode := Head;

While CurrNode^.Next^.Value <> Value Do

CurrNode := CurrNode^.Next;

TempNode := CurrNode^.Next;

CurrNode^.Next := CurrNode^.Next^.Next;

End;

DeleteEdges(TempNode^.Value);

TempNode^.Edges.Clear();

Dispose(TempNode);

Dec(Count);

End;

Function TVertexList.Find(Const Value: TValue): PVertex;

Var

CurrNode: PVertex;

Begin

CurrNode := Head;

While (CurrNode <> Nil) And (CurrNode^.Value <> Value) Do

CurrNode := CurrNode^.Next;

Result := CurrNode;

End;

Function TVertexList.GetValueArray: TValueArray;

Var

Arr: TValueArray;

Vertex: PVertex;

I: Integer;

Begin

Vertex := Self.Head;

SetLength(Arr, Self.Count);

I := 0;

While (Vertex <> Nil) Do

Begin

Arr[I] := Vertex.Value;

Inc(I);

Vertex := Vertex^.Next;

End;

GetValueArray := Arr;

End;

Function TVertexList.GetVertexArray: TVertArray;

Var

Arr: TVertArray;

Vertex: PVertex;

I: Integer;

Begin

Vertex := Self.Head;

SetLength(Arr, Self.Count);

I := 0;

While (Vertex <> Nil) Do

Begin

Arr[I] := Vertex^;

Inc(I);

Vertex := Vertex^.Next;

End;

GetVertexArray := Arr;

End;

Procedure TVertexList.Clear();

Var

CurrNode, TempNode: PVertex;

Begin

CurrNode := Head;

While CurrNode <> Nil Do

Begin

TempNode := CurrNode;

CurrNode := CurrNode^.Next;

TempNode^.Edges.Clear();

Dispose(TempNode);

End;

Head := Nil;

Count := 0;

End;

Function GetIndex(Value: Integer; Arr: TVertArray): Integer;

Var

Ind: Integer;

I: Integer;

Begin

Ind := -1;

For I := 0 To High(Arr) Do

Begin

If Arr[I].Value = Value Then

Ind := I;

End;

GetIndex := Ind;

End;

Function TVertexList.FindWay(StartValue: Integer): TWayArr;

Var

WayArr: TWayArr;

VertArr: TVertArray;

ValueArr: TValueArray;

Edge: PEdge;

I, J, Index: Integer;

Begin

VertArr := Self.GetVertexArray();

ValueArr := Self.GetValueArray();

SetLength(WayArr, Self.Count);

For I := 0 To High(WayArr) Do

Begin

If ValueArr[I] = StartValue Then

WayArr[I] := 0

Else

WayArr[I] := INF;

End;

For I := 1 To Self.Count - 1 Do

Begin

For J := 0 To High(VertArr) Do

Begin

Edge := VertArr[J].Edges.Head;

While (Edge <> Nil) Do

Begin

Index := GetIndex(Edge.Value, VertArr);

If WayArr[Index] + 1 < WayArr[J] Then

WayArr[J] := WayArr[Index] + 1;

Edge := Edge.Next;

End;

End;

End;

FindWay := WayArr;

End;

End.

Unit EdgeListUnit;

Interface

Type

TValue = Integer;

PEdge = ^TEdge;

TEdge = Record

Value: TValue;

Next: PEdge;

End;

TEdgeList = Record

Head: PEdge;

Procedure Create();

Procedure Add(Const Value: TValue);

Procedure Delete(Const Value: TValue);

Function Find(Const Value: TValue): PEdge;

Procedure Clear();

End;

Implementation

Procedure TEdgeList.Create();

Begin

Head := Nil;

End;

Function CreateNode(Const Value: TValue): PEdge;

Var

NewNode: PEdge;

Begin

New(NewNode);

NewNode^.Value := Value;

NewNode^.Next := Nil;

Result := NewNode;

End;

Procedure TEdgeList.Add(Const Value: TValue);

Var

NewNode, CurrNode: PEdge;

Begin

NewNode := CreateNode(Value);

If Head = Nil Then

Head := NewNode

Else If Head^.Value > Value Then

Begin

NewNode^.Next := Head;

Head := NewNode;

End

Else

Begin

CurrNode := Head;

While (CurrNode^.Next <> Nil) And (CurrNode^.Next^.Value <= Value) Do

CurrNode := CurrNode^.Next;

NewNode^.Next := CurrNode^.Next;

CurrNode^.Next := NewNode;

End;

End;

Procedure TEdgeList.Delete(Const Value: TValue);

Var

TempNode, CurrNode: PEdge;

Begin

If Head^.Value = Value Then

Begin

TempNode := Head;

Head := Head^.Next;

End

Else

Begin

CurrNode := Head;

While CurrNode^.Next^.Value <> Value Do

CurrNode := CurrNode^.Next;

TempNode := CurrNode^.Next;

CurrNode^.Next := CurrNode^.Next^.Next;

End;

Dispose(TempNode);

End;

Function TEdgeList.Find(Const Value: TValue): PEdge;

Var

CurrNode: PEdge;

Begin

CurrNode := Head;

While (CurrNode <> Nil) And (CurrNode^.Value <> Value) Do

CurrNode := CurrNode^.Next;

Result := CurrNode;

End;

Procedure TEdgeList.Clear();

Var

CurrNode, TempNode: PEdge;

Begin

CurrNode := Head;

While CurrNode <> Nil Do

Begin

TempNode := CurrNode;

CurrNode := CurrNode^.Next;

Dispose(TempNode);

End;

Head := Nil;

End;

End.

Unit PositiveNumberComponentUnit;

Const

ENTER = #13;

BACKSPACE = #8;

NONE = #0;

DIGITS = ['0' .. '9'];

DIGITS\_WITHOUT\_ZERO = ['1' .. '9'];

ALPHABET = ['A' .. 'Z', 'a' .. 'z'];

Var

CtrlPressed: Boolean = False;

Procedure PositiveNumberComponentContextPopup(Const Text: String;

Const SelStart, SelLength, MAX: Integer; Var Handled: Boolean);

Procedure PositiveNumberComponentExit(Text: String; Const MIN: Integer);

Procedure PositiveNumberComponentKeyDown(Const Text: String;

Const SelStart, SelLength, MAX: Integer; Var Key: Word; Shift: TShiftState);

Procedure PositiveNumberComponentKeyPress(Const Text: String;

Const SelStart, SelLength, MAX: Integer; Var Key: Char);

Procedure PositiveNumberComponentKeyUp();

Implementation

Function IsValidRange(Const Num, MAX: Integer): Boolean;

Begin

IsValidRange := Num <= MAX;

End;

Function IsPossiblePaste(Const Text: String; Const SelStart, SelLength,

MAX: Integer): Boolean;

Var

Num: Integer;

Begin

IsPossiblePaste := Clipboard.HasFormat(CF\_TEXT) And

(Length(Clipboard.AsText) <> 0) And

TryStrToInt(Copy(Text, 1, SelStart) + Clipboard.AsText + Copy(Text,

SelStart + SelLength + 1), Num) And

((SelStart = 0) And (Clipboard.AsText[1] <> '0') Or (SelStart > 0)) And

IsValidRange(StrToInt(Copy(Text, 1, SelStart) + Clipboard.AsText +

Copy(Text, SelStart + SelLength + 1)), MAX);

End;

Function IsValidChar(Const SelStart: Integer; Const Key: Char): Boolean;

Begin

IsValidChar := (SelStart = 0) And CharInSet(Key, DIGITS\_WITHOUT\_ZERO) Or

(SelStart > 0) And CharInSet(Key, DIGITS);

End;

Procedure PositiveNumberComponentContextPopup(Const Text: String;

Const SelStart, SelLength, MAX: Integer; Var Handled: Boolean);

Begin

If Not IsPossiblePaste(Text, SelStart, SelLength, MAX) Or (SelLength = 0)

And (SelStart = 1) And (Length(Text) > 1) And (Text[2] = '0') Or

(SelLength > 0) And (SelStart = 0) And (SelLength <> Length(Text)) And

(Text[SelLength + 1] = '0') Then

Handled := True;

End;

Procedure PositiveNumberComponentExit(Text: String; Const MIN: Integer);

Begin

If (Text <> '') And (StrToInt(Text) < MIN) Then

Text := '';

End;

Procedure PositiveNumberComponentKeyDown(Const Text: String;

Const SelStart, SelLength, MAX: Integer; Var Key: Word; Shift: TShiftState);

Begin

If (Shift = [SsCtrl]) And (UpCase(Chr(Key)) = 'X') Then

Begin

If (SelLength = 0) And (SelStart = 1) And (Length(Text) > 1) And

(Text[2] = '0') Or (SelLength > 0) And (SelStart = 0) And

(SelLength <> Length(Text)) And (Text[SelLength + 1] = '0') Then

Key := Ord(NONE);

End

Else If Key = VK\_DELETE Then

Begin

If (SelLength = 0) And (SelStart = 0) And (Length(Text) > 1) And

(Text[2] = '0') Or (SelLength > 0) And (SelStart = 0) And

(SelLength <> Length(Text)) And (Text[SelLength + 1] = '0') Then

Key := Ord(NONE);

End

Else If (Shift = [SsCtrl]) And (UpCase(Chr(Key)) = 'V') Or

(Shift = [SsShift]) And (Key = VK\_INSERT) Then

Begin

If Not IsPossiblePaste(Text, SelStart, SelLength, MAX) Then

Key := Ord(NONE);

End;

If (Shift = [SsCtrl]) And CharInSet(Chr(Key), ALPHABET) Then

CtrlPressed := True;

End;

Procedure PositiveNumberComponentKeyPress(Const Text: String;

Const SelStart, SelLength, MAX: Integer; Var Key: Char);

Begin

If Key = BACKSPACE Then

Begin

If (SelLength = 0) And (SelStart = 1) And (Length(Text) > 1) And

(Text[2] = '0') Or (SelLength > 0) And (SelStart = 0) And

(SelLength <> Length(Text)) And (Text[SelLength + 1] = '0') Then

Key := NONE;

End

Else If Not CtrlPressed Then

Begin

If Not(IsValidChar(SelStart, Key) And

IsValidRange(StrToInt(Copy(Text, 1, SelStart) + Key + Copy(Text,

SelStart + SelLength + 1)), MAX)) Then

Key := NONE;

End;

End;

Procedure PositiveNumberComponentKeyUp();

Begin

CtrlPressed := False;

End;

End.

Unit FileUnit;

Function ReadFileGraph(Var InputFile: TextFile;

Var TempGraph: TVertexList): Boolean;

Procedure WriteFileGraph(Var OutputFile: TextFile; Graph: TVertexList);

Implementation

Uses

System.SysUtils,

MenuUnit;

Function ReadFileVerteces(Var InputFile: TextFile;

Var TempGraph: TVertexList): Boolean;

Var

IsCorrect: Boolean;

TempString: String;

Value: TValue;

Begin

IsCorrect := True;

ReadLn(InputFile, TempString);

While IsCorrect And (TempString <> '') Do

Begin

IsCorrect := TryStrToInt(TempString, Value) And

(TempGraph.Find(Value) = Nil) And (TempGraph.Count < MAX\_VERTEX);

If IsCorrect Then

TempGraph.Add(Value);

ReadLn(InputFile, TempString);

End;

Result := IsCorrect;

End;

Function ReadFileEdges(Var InputFile: TextFile;

Var TempGraph: TVertexList): Boolean;

Var

IsCorrect: Boolean;

Value1, Value2: TValue;

Vertex1, Vertex2: PVertex;

Begin

IsCorrect := True;

Value1 := 0;

Value2 := 0;

Vertex1 := Nil;

Vertex2 := Nil;

While IsCorrect And Not SeekEOF(InputFile) Do

Begin

Try

Read(InputFile, Value1);

Read(InputFile, Value2);

Except

IsCorrect := False;

End;

If IsCorrect Then

Begin

Vertex1 := TempGraph.Find(Value1);

Vertex2 := TempGraph.Find(Value2);

End;

IsCorrect := (Vertex1 <> Nil) And (Vertex2 <> Nil);

If IsCorrect And (Vertex1^.Edges.Find(Value2) = Nil) Then

Begin

Vertex1^.Edges.Add(Value2);

Vertex2^.Edges.Add(Value1);

End;

End;

Result := IsCorrect;

End;

Function ReadFileGraph(Var InputFile: TextFile;

Var TempGraph: TVertexList): Boolean;

Var

IsCorrect: Boolean;

Begin

Reset(InputFile);

IsCorrect := ReadFileVerteces(InputFile, TempGraph);

If IsCorrect Then

IsCorrect := ReadFileEdges(InputFile, TempGraph);

CloseFile(InputFile);

Result := IsCorrect;

End;

Procedure WriteFileVerteces(Var OutputFile: TextFile; Graph: TVertexList);

Var

CurrVertex: PVertex;

Begin

CurrVertex := Graph.Head;

While CurrVertex <> Nil Do

Begin

WriteLn(OutputFile, CurrVertex^.Value);

CurrVertex := CurrVertex^.Next;

End;

WriteLn(OutputFile);

End;

Procedure WriteFileEdges(Var OutputFile: TextFile; Graph: TVertexList);

Var

CurrVertex: PVertex;

CurrEdge: PEdge;

Begin

CurrVertex := Graph.Head;

While CurrVertex <> Nil Do

Begin

CurrEdge := CurrVertex^.Edges.Head;

While CurrEdge <> Nil Do

Begin

WriteLn(OutputFile, IntToStr(CurrVertex^.Value) + ' ' +

IntToStr(CurrEdge^.Value));

CurrEdge := CurrEdge^.Next;

End;

CurrVertex := CurrVertex^.Next;

End;

End;

Procedure WriteFileGraph(Var OutputFile: TextFile; Graph: TVertexList);

Begin

ReWrite(OutputFile);

WriteFileVerteces(OutputFile, Graph);

WriteFileEdges(OutputFile, Graph);

CloseFile(OutputFile);

End;

End.

Unit AdjacencyMatrixUnit;

Uses

MenuUnit;

Procedure TAdjacencyMatrixForm.CreateParams(Var Params: TCreateParams);

Begin

Inherited;

Params.ExStyle := Params.ExStyle Or WS\_EX\_APPWINDOW;

End;

Function TAdjacencyMatrixForm.FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

Result := False;

End;

Procedure TAdjacencyMatrixForm.FormShow(Sender: TObject);

Var

CurrVertex: PVertex;

CurrEdge: PEdge;

I, J: Integer;

Begin

AdjacencyMatrixStringGrid.ColCount := Graph.Count + 1;

AdjacencyMatrixStringGrid.RowCount := Graph.Count + 1;

CurrVertex := Graph.Head;

I := 1;

While CurrVertex <> Nil Do

Begin

AdjacencyMatrixStringGrid.Cells[0, I] := IntToStr(CurrVertex.Value);

AdjacencyMatrixStringGrid.Cells[I, 0] := IntToStr(CurrVertex.Value);

CurrVertex := CurrVertex^.Next;

Inc(I);

End;

CurrVertex := Graph.Head;

I := 1;

While CurrVertex <> Nil Do

Begin

CurrEdge := CurrVertex^.Edges.Head;

J := 1;

While J <> AdjacencyMatrixStringGrid.ColCount Do

Begin

If (CurrEdge <> Nil) And

(IntToStr(CurrEdge^.Value) = AdjacencyMatrixStringGrid.Cells

[J, 0]) Then

Begin

AdjacencyMatrixStringGrid.Cells[J, I] := '1';

CurrEdge := CurrEdge^.Next;

End

Else

AdjacencyMatrixStringGrid.Cells[J, I] := '0';

Inc(J);

End;

CurrVertex := CurrVertex^.Next;

Inc(I);

End;

End;

Procedure TAdjacencyMatrixForm.FormKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

End.

Unit WaysUnit;

Interface

Uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Grids, VertexListUnit,

Vcl.ExtCtrls, System.Math;

Type

TWayForm = Class(TForm)

WaysGrid: TStringGrid;

PaintBox: TPaintBox;

Procedure FormShow(Sender: TObject);

Procedure FormInit(Graph: TVertexList; Value: Integer);

Procedure PaintBoxPaint(Sender: TObject);

Procedure WaysGridSelectCell(Sender: TObject; ACol, ARow: Integer;

Var CanSelect: Boolean);

Private

Graph: TVertexList;

Value: Integer;

Public

{ Public declarations }

End;

Var

WayForm: TWayForm;

Implementation

{$R \*.dfm}

Procedure Clear(Grid: TStringGrid);

Var

I: Integer;

J: Integer;

Begin

For I := 0 To Grid.ColCount Do

For J := 0 To Grid.RowCount Do

Grid.Cells[I, J] := '';

Grid.ColCount := 0;

Grid.RowCount := 0;

End;

Procedure TWayForm.FormInit(Graph: TVertexList; Value: Integer);

Begin

Self.Graph := Graph;

Self.Value := Value;

End;

Procedure TWayForm.FormShow(Sender: TObject);

Var

WayArr: TWayArr;

Vertex: PVertex;

I, Ind: Integer;

Begin

WayArr := Graph.FindWay(Value);

Clear(WaysGrid);

With WaysGrid Do

Begin

Ind := 0;

ColCount := 4;

RowCount := Graph.Count - 1;

Vertex := Graph.Head;

For I := 0 To RowCount - 1 Do

Begin

Cells[0, I] := IntToStr(Value);

Cells[1, I] := '-->';

If Value = Vertex^.Value Then

Begin

Vertex := Vertex.Next;

Inc(Ind);

End;

Cells[2, I] := IntTostr(Vertex^.Value);

If WayArr[Ind] = INF Then

Cells[3, I] := 'Нет'

Else

Cells[3, I] := IntToStr(WayArr[Ind]);

Inc(Ind);

Vertex := Vertex.Next;

End;

End;

End;

Procedure TWayForm.PaintBoxPaint(Sender: TObject);

Var

CenterX, CenterY, Distance: Integer;

RotateAngle: Real;

Begin

PaintBox.Canvas.Brush.Color := ClWhite;

PaintBox.Canvas.FillRect(PaintBox.Canvas.ClipRect);

If Graph.Count <> 0 Then

Begin

PaintBox.Canvas.Pen.Width := 1;

CenterX := PaintBox.Width Div 2;

CenterY := PaintBox.Height Div 2;

Distance := Round(LogN(2, Graph.Count) \*

(PaintBox.Width \* VertexSize / 100));

RotateAngle := 2 \* Pi / Graph.Count;

DrawVerteces(PaintBox, Graph, CenterX, CenterY, Distance, RotateAngle);

End;

End;

Procedure TWayForm.WaysGridSelectCell(Sender: TObject; ACol, ARow: Integer;

Var CanSelect: Boolean);

Begin

PaintBoxPaint(Self);

End;

End.

**Код программы Java**

package lab72;

import java.util.Scanner;

public class Main {

static VertexList VList;

enum ErrCode {

SUCCESS,

INCORRECT\_DATA,

SUCH\_ELEMENT\_ALREADY\_EXIST,

EDGE\_NOT\_CORRECT,

VERTEX\_NOT\_CORRECT,

GRAPH\_NOT\_EXIST,

}

enum Choice {

createGraph("Создать новый граф"),

addVertex("Добавить вершину"),

addEdge("Добавить ребро"),

print("Вывести граф"),

matrix("Матрица смежности"),

deleteVertex("Удалить вершину"),

deleteEdge("Удалить ребро"),

findWays("Найти кратчайшие"),

close("Закрыть");

private final String inf;

Choice (String infLine) {

this.inf = infLine;

}

private String getInf(){return this.ordinal() + ") " + this.inf;}

}

static final String[] ERRORS = {"Удача",

"Данные некорректные или число слишком большое (должно быть от %d до %d)\n",

"Такая вершина уже существует",

"Некорректное ребро",

"Некорректная вершина",

"Сначала стоит создать граф)",};

static final String INFORMATION\_TEXT = """

Инструкция:

-- Вершины графа должны быть от 1 до 99

-- Вершины графа не могут повторяться

""",

ATTENTION\_TEXT = """

Внимание! Если граф уже существует он удалиться, вы уверены?

1) Да

2) Нет

""";

static final int MIN\_VERT = 1,

MAX\_VERT = 99;

static ErrCode enterOneNum(int[] numberArr, Scanner input, final int MIN, final int MAX) {

int number = 0;

ErrCode err = ErrCode.SUCCESS;

try {

number = Integer.parseInt(input.nextLine());

} catch (NumberFormatException e) {

err = ErrCode.INCORRECT\_DATA;

}

if ((err == ErrCode.SUCCESS) && (number < MIN || number > MAX))

err = ErrCode.INCORRECT\_DATA;

numberArr[0] = err == ErrCode.SUCCESS ? number : 0;

return err;

}

static int getNumConsole(Scanner input, final int MIN, final int MAX) {

ErrCode err;

int[] numberArr = {0};

do {

err = enterOneNum(numberArr, input, MIN, MAX);

if (err != ErrCode.SUCCESS) {

System.err.printf(ERRORS[err.ordinal()], MIN, MAX);

System.out.println("Введите снова");

}

} while (err != ErrCode.SUCCESS);

return numberArr[0];

}

static void printMenu() {

Choice[] choices = Choice.values();

for (Choice choice : choices) {

System.out.println(choice.getInf());

}

}

static void printInf(Scanner input) {

System.out.println(INFORMATION\_TEXT);

System.out.println("нажмите enter чтобы продолжить");

input.nextLine();

}

static Choice getChoice(Scanner input) {

printMenu();

int choice;

int maxChoice = Choice.values().length - 1;

choice = getNumConsole(input, 0, maxChoice);

return Choice.values()[choice];

}

static void doFunction(Choice choice, Scanner input) {

switch (choice) {

case createGraph -> {

System.out.println(ATTENTION\_TEXT);

int localChoice = getNumConsole(input, 1, 2);

if (localChoice == 1)

VList = new VertexList();

}

case addVertex -> {

if (VList != null) {

System.out.println("Введите новую вершину: ");

int newVert = getNumConsole(input, MIN\_VERT, MAX\_VERT);

if (!VList.addVertex(newVert)) {

System.err.println(ERRORS[ErrCode.SUCH\_ELEMENT\_ALREADY\_EXIST.ordinal()]);

}

}

else

System.err.println(ERRORS[ErrCode.GRAPH\_NOT\_EXIST.ordinal()]);

}

case addEdge -> {

if (VList != null) {

System.out.println("Введите первую вершину: ");

int startVert = getNumConsole(input, MIN\_VERT, MAX\_VERT);

System.out.println("Введите вторую вершину: ");

int endVert = getNumConsole(input, MIN\_VERT, MAX\_VERT);

if (!VList.addEdge(startVert, endVert)) {

System.err.println(ERRORS[ErrCode.EDGE\_NOT\_CORRECT.ordinal()]);

}

}

else

System.err.println(ERRORS[ErrCode.GRAPH\_NOT\_EXIST.ordinal()]);

}

case print -> {

if (VList != null)

VList.print();

else

System.err.println(ERRORS[ErrCode.GRAPH\_NOT\_EXIST.ordinal()]);

System.out.println();

}

case matrix -> {

if (VList != null)

VList.printMatrix();

else

System.err.println(ERRORS[ErrCode.GRAPH\_NOT\_EXIST.ordinal()]);

System.out.println();

}

case deleteEdge -> {

if (VList != null) {

System.out.println("Введите первую вершину: ");

int startVert = getNumConsole(input, MIN\_VERT, MAX\_VERT);

System.out.println("Введите вторую вершину: ");

int endVert = getNumConsole(input, MIN\_VERT, MAX\_VERT);

if (!VList.deleteEdge(startVert, endVert)) {

System.err.println(ERRORS[ErrCode.EDGE\_NOT\_CORRECT.ordinal()]);

}

}

else

System.err.println(ERRORS[ErrCode.GRAPH\_NOT\_EXIST.ordinal()]);

}

case deleteVertex -> {

if (VList != null) {

System.out.println("Введите вершину: ");

int newVert = getNumConsole(input, MIN\_VERT, MAX\_VERT);

if (!VList.deleteVertex(newVert)) {

System.err.println(ERRORS[ErrCode.SUCH\_ELEMENT\_ALREADY\_EXIST.ordinal()]);

}

}

else

System.err.println(ERRORS[ErrCode.GRAPH\_NOT\_EXIST.ordinal()]);

}

case findWays -> {

if (VList != null) {

System.out.println("Введите стартовую вершину: ");

int vert = getNumConsole(input, MIN\_VERT, MAX\_VERT);

if (VList.containce(vert)) {

VList.findWay(vert);

} else

System.err.println(ERRORS[ErrCode.VERTEX\_NOT\_CORRECT.ordinal()]);

}

else

System.err.println(ERRORS[ErrCode.GRAPH\_NOT\_EXIST.ordinal()]);

}

}

}

public static void main(String[] args){

Scanner input = new Scanner(System.in);

printInf(input);

Choice choice;

do {

choice = getChoice(input);

if (choice != Choice.close)

doFunction(choice, input);

} while (choice != Choice.close);

input.close();

}

}

VertexList

package lab72;

import java.util.Arrays;

import java.util.HashMap;

import java.util.Map;

class Vertex {

EdgesList edges;

int value;

Vertex next;

Vertex (int newValue) {

this.value = newValue;

this.edges = new EdgesList();

this.next = null;

}

boolean isEdgeWith(int value) {

Edge edge = edges.head;

while (edge != null && edge.value != value) {

edge = edge.next;

}

return edge != null;

}

}

public class VertexList {

Vertex head;

int count;

VertexList () {

this.head = null;

this.count = 0;

}

boolean addVertex(int value) {

boolean isAdded;

if (this.head == null) {

this.head = new Vertex(value);

count++;

isAdded = true;

} else {

Vertex vertex = this.head;

while (vertex.next != null && vertex.value != value) {

vertex = vertex.next;

}

if (vertex.value != value) {

vertex.next = new Vertex(value);

isAdded = true;

count++;

} else {isAdded = false;};

}

return isAdded;

}

boolean containce(int value) {

Vertex vertex = this.head;

while (vertex != null && vertex.value != value) {

vertex = vertex.next;

}

return vertex != null;

}

boolean addEdge(int startVertex, int endVertex) {

boolean isAdded = false;

Vertex vertex = this.head;

if (vertex != null && containce(startVertex) && containce(endVertex) && startVertex != endVertex) {

while (vertex.value != startVertex && vertex.value != endVertex)

vertex = vertex.next;

vertex.edges.add(vertex.value == startVertex ? endVertex : startVertex);

vertex = vertex.next;

while (vertex.value != startVertex && vertex.value != endVertex)

vertex = vertex.next;

vertex.edges.add(vertex.value == startVertex ? endVertex : startVertex);

isAdded = true;

}

return isAdded;

}

private void deleteEdges(Vertex vertex) {

Edge edge = vertex.edges.head;

while (edge != null) {

Vertex head = this.head;

while (head.value != edge.value) {

head = head.next;

}

head.edges.delete(vertex.value);

edge = edge.next;

}

}

boolean deleteVertex(int value) {

boolean isDeleted = false;

if (this.count != 0 && containce(value)) {

isDeleted = true;

count--;

if (this.head.value == value) {

deleteEdges(this.head);

this.head = this.head.next;

} else {

Vertex vertex = this.head;

while (vertex.next.value != value) {

vertex = vertex.next;

}

deleteEdges(vertex.next);

vertex.next = vertex.next.next;

}

}

return isDeleted;

}

boolean deleteEdge(int startVert, int endVert) {

boolean isDeleted = false;

if (containce(startVert) && containce(endVert)) {

isDeleted = true;

Vertex vertex = this.head;

while (vertex.value != startVert && vertex.value != endVert)

vertex = vertex.next;

vertex.edges.delete(vertex.value == startVert ? endVert : startVert);

vertex = vertex.next;

while (vertex.value != startVert && vertex.value != endVert)

vertex = vertex.next;

vertex.edges.delete(vertex.value == startVert ? endVert : startVert);

}

return isDeleted;

}

void print() {

Vertex vertex = this.head;

while (vertex != null) {

System.out.print("Вершина: " + vertex.value);

System.out.print(" --> ");

Edge edge = vertex.edges.head;

while (edge != null) {

System.out.print(edge.value + " ");

edge = edge.next;

}

System.out.println();

vertex = vertex.next;

}

}

void printMatrix() {

Vertex vertex = this.head;

StringBuilder line = new StringBuilder(" ");

for (int i = 0; i < this.count; i++){

line.append(String.format("%-3d", vertex.value));

vertex = vertex.next;

}

System.out.println(line);

Vertex mainVertex = this.head;

for (int i = 1; i <= this.count; i++){

line = new StringBuilder(String.format("%-3d", mainVertex.value));

vertex = this.head;

for (int j = 1; j <= this.count; j++){

int isEdgeWithVert = mainVertex.isEdgeWith(vertex.value) ? 1 : 0;

line.append(String.format("%-3d", isEdgeWithVert));

vertex = vertex.next;

}

System.out.println(line);

mainVertex = mainVertex.next;

}

}

private Vertex[] getArray() {

Vertex vertex = this.head;

Vertex[] array = new Vertex[this.count];

int i = 0;

while (vertex != null) {

array[i++] = vertex;

vertex = vertex.next;

}

return array;

}

Vertex getByValue(int value) {

Vertex vertex = this.head;

while (vertex != null && vertex.value != value) {

vertex = vertex.next;

}

return vertex;

}

void findWay(int startVal) {

final int INF = 2000000000;

HashMap<Vertex, Integer> wayMap = new HashMap<>();

Vertex[] vertexArray = getArray();

for (int i = 0; i < this.count; i++)

wayMap.put(vertexArray[i], vertexArray[i].value == startVal ? 0 : INF);

for (int i = 0; i < this.count; i++)

for (Vertex vertex : vertexArray) {

Edge edge = vertex.edges.head;

while (edge != null) {

Vertex currVert = getByValue(edge.value);

if (wayMap.get(vertex) + 1 < wayMap.get(currVert))

wayMap.put(currVert, wayMap.get(vertex) + 1);

edge = edge.next;

}

}

System.out.println(wayMap);

Vertex startVert = getByValue(startVal);

for (Vertex key : wayMap.keySet()) {

if (key != startVert) {

String wayLen = wayMap.get(key) == INF ? "нет" : wayMap.get(key).toString();

System.out.println("Путь из " + startVert.value + " в " + key.value + " --> " + wayLen);

}

}

}

}

EdgeList

package lab72;

class Edge {

int value;

Edge next;

Edge (int value) {

this.value = value;

this.next = null;

}

}

public class EdgesList {

Edge head;

EdgesList () {

this.head = null;

}

boolean add(int value) {

boolean isAdded;

if (this.head == null) {

this.head = new Edge(value);

isAdded = true;

} else {

Edge edge = this.head;

while (edge.next != null && edge.value != value) {

edge = edge.next;

}

if (edge.value != value) {

edge.next = new Edge(value);

isAdded = true;

} else {isAdded = false;};

}

return isAdded;

}

void delete(int value) {

if (this.head.value == value) {

this.head = this.head.next;

} else {

Edge edge = this.head;

while (edge.next.value != value) {

edge = edge.next;

}

edge.next = edge.next.next;

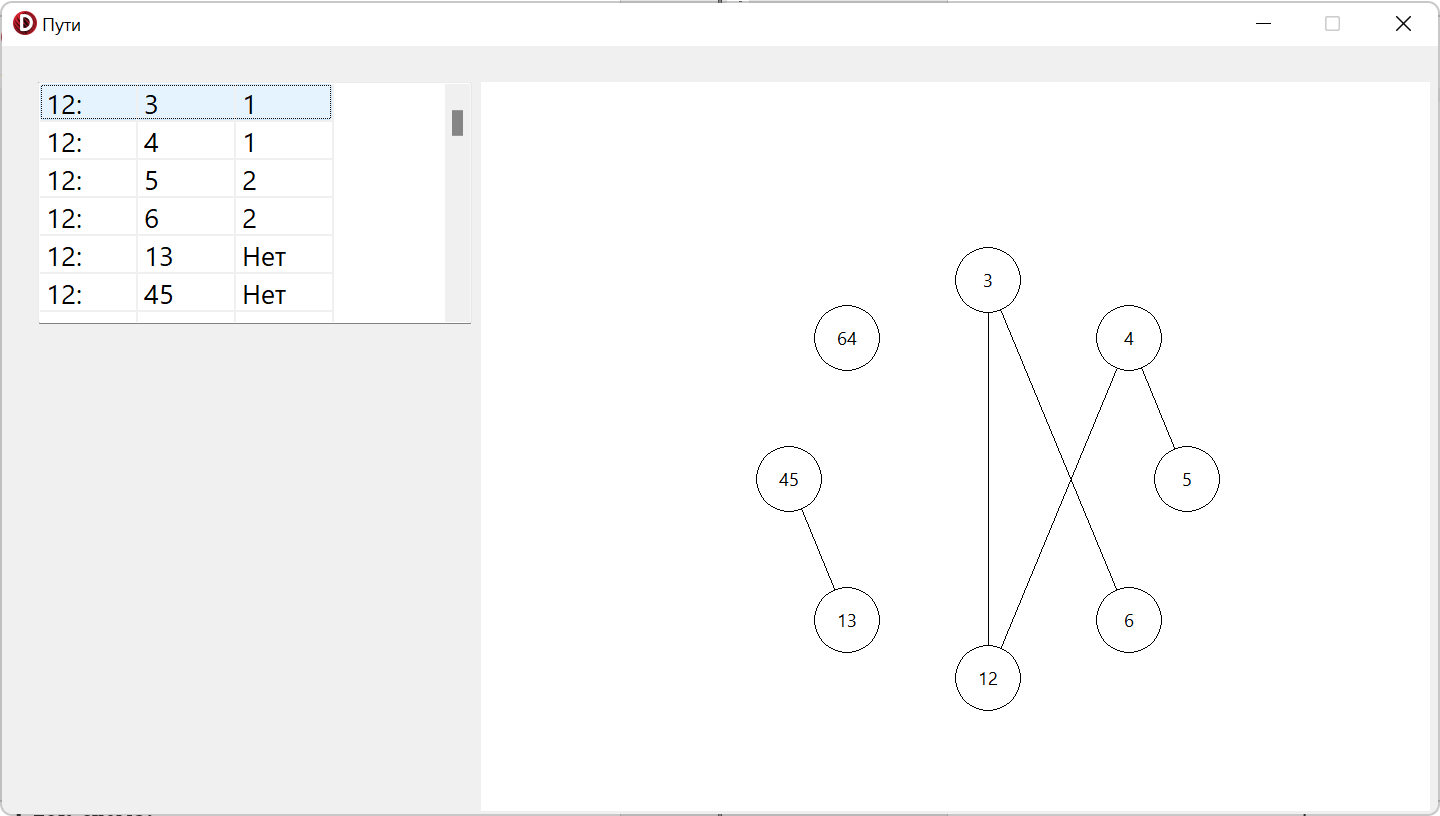
}

}

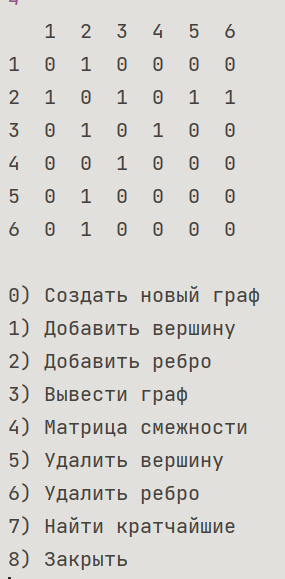
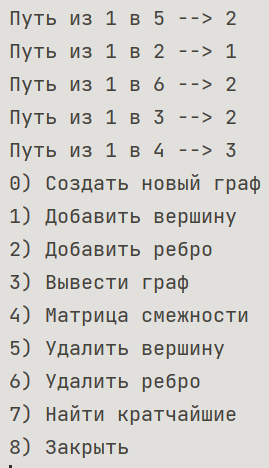
}

**Скриншоты:**

**Delphi****:**



**Java:**

** **

**Блок-схема:**

