УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №6.2

по предмету «Основы алгоритмизации и программирования»

Вариант 4

Выполнил:

Бражалович А. И.

Гр. 351004

Проверил:

Данилова Г. В.

Минск 2024

**Задание:**

Комбинаторика. Имеется n населенных пунктов, перенумерованных от 1 до n. Некоторые пары пунктов соединены дорогами. Разработать рекурсивную процедуру определения, можно ли попасть по этим дорогам из 1-го пункта в n-й. Визуализировать!

**Код программы Delphi:**

Unit GraphLinkedList;

Interface

Uses

Vcl.Grids, Vcl.Graphics, Vcl.ExtCtrls, System.SysUtils, QueueLinkedList;

Type

TGraphMatrix = Array Of Array Of Boolean;

TPathArray = Array Of Boolean;

TArray = Array Of Integer;

TTownGraph = ^TGNode;

TGNode = Record

Number: Integer;

Next: TTownGraph;

X: Integer;

Y: Integer;

End;

Const

DIAMETR = 50;

RADIUS = 150;

CENTER\_X = 200;

CENTER\_Y = 200;

Procedure DrawTowns(PaintBox: TPaintBox);

Procedure AddTown();

Procedure Make();

Procedure AddLink(StartTown, EndTown: Integer);

Procedure DrawLines(Var PaintBox: TPaintBox);

Procedure ClearGraph();

Function DFS(CurrentCity, Target: Integer): Boolean;

Function FindPathDFS(Start, Target: Integer): TArray;

Procedure DrawPath(Var PaintBox: TPaintBox);

Var

CountOfTowns: Integer = 1;

Head, Town: TTownGraph;

GraphMatrix: TGraphMatrix;

Visited: TPathArray;

Path: TArray;

Implementation

Function DFS(CurrentCity, Target: Integer): Boolean;

Var

NeighborCity, J: Integer;

Ready: Boolean;

Begin

Visited[CurrentCity - 1] := True;

Ready := False;

SetLength(Path, Length(Path) + 1);

J := 0;

Path[Length(Path) - 1] := CurrentCity;

If CurrentCity = Target Then

Ready := True

Else

Begin

While (J < Length(GraphMatrix)) And (Not Ready) Do

Begin

If GraphMatrix[CurrentCity - 1, J] Then

Begin

NeighborCity := J + 1;

If Not Visited[NeighborCity - 1] Then

Ready := DFS(NeighborCity, Target);

End;

Inc(J);

End;

If Not Ready Then

SetLength(Path, Length(Path) - 1);

End;

DFS := Ready

End;

Function FindPathDFS(Start, Target: Integer): TArray;

Var

I: Integer;

Begin

SetLength(Visited, CountOfTowns);

Path := Nil;

For I := 0 To CountOfTowns - 1 Do

Visited[I] := False;

If DFS(Start, Target) Then

FindPathDFS := Path

Else

FindPathDFS := Nil;

End;

Procedure AddLink(StartTown, EndTown: Integer);

Begin

GraphMatrix[StartTown - 1, EndTown - 1] := True;

GraphMatrix[EndTown - 1, StartTown - 1] := True;

End;

Procedure Make();

Begin

New(Head);

Head.Next := Nil;

Head.Number := CountOfTowns;

End;

Procedure AddTown();

Var

I, J: Integer;

Begin

If Head = Nil Then

Make()

Else

Begin

Town := Head;

While Town.Next <> Nil Do

Town := Town^.Next;

New(Town.Next);

Town := Town.Next;

Inc(CountOfTowns);

Town.Number := CountOfTowns;

Town.Next := Nil;

End;

SetLength(GraphMatrix, CountOfTowns, CountOfTowns);

For I := 0 To High(GraphMatrix) Do

For J := 0 To High(GraphMatrix) Do

GraphMatrix[I, J] := False;

End;

Procedure DrawTowns(PaintBox: TPaintBox);

Var

DeltaAngle, Angle: Double;

I: Integer;

Begin

Town := Head;

DeltaAngle := 2 \* Pi / CountOfTowns;

With PaintBox.Canvas Do

Begin

For I := 0 To CountOfTowns - 1 Do

Begin

Angle := DeltaAngle \* I;

Town.X := Round(CENTER\_X + Radius \* Cos(Angle));

Town.Y := Round(CENTER\_Y + Radius \* Sin(Angle));

With PaintBox.Canvas Do

Begin

Pen.Color := ClBlack;

Ellipse(Town.X, Town.Y, Town.X + Diametr, Town.Y + Diametr);

TextOut(Town.X + (Diametr - TextWidth(IntToStr(Town.Number)))

Div 2, Town.Y + (Diametr - TextHeight(IntToStr(Town.Number)))

Div 2, IntToStr(Town.Number));

End;

Town := Town^.Next;

End;

End;

End;

Procedure DrawLines(Var PaintBox: TPaintBox);

Var

I, J: Integer;

StartT, EndT: Integer;

StartX, StartY, EndX, EndY: Integer;

Begin

DrawTowns(PaintBox);

For I := 0 To High(GraphMatrix) Do

For J := 0 To High(GraphMatrix) Do

Begin

If GraphMatrix[I, J] = True Then

Begin

StartT := I + 1;

EndT := J + 1;

Town := Head;

While Town.Number <> StartT Do

Town := Town.Next;

StartX := Town.X;

StartY := Town.Y;

Town := Head;

While Town.Number <> EndT Do

Town := Town.Next;

EndX := Town.X;

EndY := Town.Y;

With PaintBox.Canvas Do

Begin

Pen.Color := ClBlack;

Pen.Width := 3;

MoveTo(StartX + DIAMETR Div 2, StartY + DIAMETR Div 2);

LineTo(EndX + DIAMETR Div 2, EndY + DIAMETR Div 2);

End;

End;

End;

DrawTowns(PaintBox);

End;

Procedure DrawPath(Var PaintBox: TPaintBox);

Var

I, J: Integer;

StartT, EndT: Integer;

StartX, StartY, EndX, EndY: Integer;

Begin

DrawLines(PaintBox);

For I := 0 To High(Path) - 1 Do

Begin

StartT := Path[I];

EndT := Path[I + 1];

Town := Head;

While Town.Number <> StartT Do

Town := Town.Next;

StartX := Town.X;

StartY := Town.Y;

Town := Head;

While Town.Number <> EndT Do

Town := Town.Next;

EndX := Town.X;

EndY := Town.Y;

With PaintBox.Canvas Do

Begin

Pen.Color := ClRed;

Pen.Width := 3;

MoveTo(StartX + DIAMETR Div 2, StartY + DIAMETR Div 2);

LineTo(EndX + DIAMETR Div 2, EndY + DIAMETR Div 2);

End;

End;

DrawTowns(PaintBox);

End;

Procedure ClearGraph();

Var

Buff: TTownGraph;

Begin

GraphMatrix := Nil;

Buff := Head;

While Buff <> Nil Do

Begin

Town := Buff^.Next;

Dispose(Buff);

Buff := Town

End;

Head := Nil;

CountOfTowns := 1;

Visited := Nil;

Path := Nil;

End;

Exports Make, DrawTowns, AddTown, AddLink, ClearGraph, DFS,

FindPathDFS, DrawPath;

Begin

End.

Unit Developer;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls;

Type

TDeveloperForm = Class(TForm)

DeveloperLabel: TLabel;

Procedure Button1Click(Sender: TObject);

Procedure FormCreate(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

DeveloperForm: TDeveloperForm;

Implementation

{$R \*.dfm}

Procedure TDeveloperForm.Button1Click(Sender: TObject);

Begin

Close;

End;

Procedure CenterDeveloperFormOnScreen(DeveloperForm: TDeveloperForm);

Begin

DeveloperForm.Left := (Screen.Width - DeveloperForm.Width) Div 2;

DeveloperForm.Top := (Screen.Height - DeveloperForm.Height) Div 2;

End;

Procedure TDeveloperForm.FormCreate(Sender: TObject);

Begin

CenterDeveloperFormOnScreen(Self);

DeveloperLabel.Caption := 'Разработчик: Бражалович Александр Иванович' +

#13#10 + 'Группа: 351005' + #13#10 + 'Tg: @Sunn4es';

DeveloperLabel.Update;

End;

End.

Unit Instruction;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls;

Type

TInstructionForm = Class(TForm)

InstructionLabel: TLabel;

Label1: TLabel;

Procedure CloseButtonClick(Sender: TObject);

Procedure FormCreate(Sender: TObject);

Function FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

InstructionForm: TInstructionForm;

Implementation

{$R \*.dfm}

Procedure CenterFormOnScreen(InstructionForm: TInstructionForm);

Begin

InstructionForm.Left := (Screen.Width - InstructionForm.Width) Div 2;

InstructionForm.Top := (Screen.Height - InstructionForm.Height) Div 2;

End;

Procedure TInstructionForm.CloseButtonClick(Sender: TObject);

Begin

Close;

End;

Procedure TInstructionForm.FormCreate(Sender: TObject);

Begin

CenterFormOnScreen(Self);

InstructionLabel.Caption :=

'1. Для добавления города нажмите на кнопку "Добавить город".' + #13#10 +

'2. Для добавления новой дороги между городами, нажмите на кнопку "Добавить дорогу".'

+ #13#10 + '3. Чтобы найти путь нажмите на кнопку "Найти путь".' + #13#10

+ 'Работа с файлом:' + #13#10 +

'1. На первой строке должно быть указано количество городов.' + #13#10 +

'2. На следующих строках располагается матрица смежности для указания дорог.'

+ #13#10 +

'3. На последней строке файла написаны два номера города, от которого и к которому'

+ #13#10 + ' вы хотите найти путь.';

End;

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

Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

End;

End.

Unit FindPath;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.Grids, Instruction,

MainForm, GraphLinkedList;

Type

TFindPathForm = Class(TForm)

StartLabel: TLabel;

EndEdit: TEdit;

StartEdit: TEdit;

StartTLabel: TLabel;

EndLabel: TLabel;

AddButton: TButton;

CancelButton: TButton;

Procedure CancelButtonClick(Sender: TObject);

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

Shift: TShiftState);

Procedure EndEditChange(Sender: TObject);

Procedure StartEditChange(Sender: TObject);

Procedure FormCreate(Sender: TObject);

Procedure EndEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure EndEditKeyPress(Sender: TObject; Var Key: Char);

Procedure StartEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure StartEditKeyPress(Sender: TObject; Var Key: Char);

Procedure AddButtonClick(Sender: TObject);

Procedure EndEditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Procedure StartEditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Function FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Function CanAdd(): Boolean;

Private

{ Private declarations }

Public

{ Public declarations }

End;

// TArray = Array of Integer;

Var

FindPathForm: TFindPathForm;

CanAddTown: Boolean = True;

Const

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

Implementation

{$R \*.dfm}

Procedure CenterFormOnScreen(FindPathForm: TFindPathForm);

Begin

FindPathForm.Left := (Screen.Width - FindPathForm.Width) Div 2;

FindPathForm.Top := (Screen.Height - FindPathForm.Height) Div 2;

End;

Procedure CheckComboButtons(Var Key: Char; Edit: TEdit;

Const Chariki: TSysCharSet);

Begin

If (Key = #22) Or ((Key = 'v') And (GetKeyState(VK\_CONTROL) < 0)) Then

Key := #0;

If Not CharInSet(Key, Chariki) And (Key <> #8) Then

Key := #0;

End;

Procedure CheckShftAndArrows(Var Key: Word; Shift: TShiftState);

Begin

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

Key := 0;

If (Key = VK\_LEFT) Or (Key = VK\_UP) Then

Key := 0

Else If (Key = VK\_RIGHT) Or (Key = VK\_DOWN) Then

Key := 0;

End;

Procedure CheckRange(Var Key: Char; Edit: TEdit; Const MAX, MIN: Integer);

Var

Value: Integer;

Begin

Value := 0;

If TryStrToInt(Edit.Text + Key, Value) Then

Begin

If (Value > MAX) Or (Value < MIN) Then

Key := #0;

End;

End;

Procedure TFindPathForm.AddButtonClick(Sender: TObject);

Var

Path: TArray;

I: Integer;

PathStr: String;

Begin

Path := FindPathDFS(StrToInt(StartEdit.Text), StrToInt(EndEdit.Text));

If Path = Nil Then

ShowMessage('Дороги нет!');

IsEdited := True;

Saved := False;

Close;

End;

Procedure TFindPathForm.CancelButtonClick(Sender: TObject);

Begin

StartEdit.Text := '';

EndEdit.Text := '';

Close;

End;

Procedure TFindPathForm.FormCreate(Sender: TObject);

Begin

AddButton.Enabled := False;

End;

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

Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

FormHelp := False;

End;

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

Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

Function TFindPathForm.CanAdd(): Boolean;

Begin

CanAdd := (StartEdit.Text <> '') And (EndEdit.Text <> '') And

(StartEdit.Text <> EndEdit.Text)

End;

Procedure TFindPathForm.StartEditChange(Sender: TObject);

Begin

AddButton.Enabled := CanAdd();

End;

Procedure TFindPathForm.StartEditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Begin

Handled := True;

End;

Procedure TFindPathForm.StartEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

CheckShftAndArrows(Key, Shift);

End;

Procedure TFindPathForm.StartEditKeyPress(Sender: TObject; Var Key: Char);

Begin

CheckComboButtons(Key, StartEdit, NUMBERS);

CheckRange(Key, StartEdit, CountOfTowns, 1);

End;

Procedure TFindPathForm.EndEditChange(Sender: TObject);

Begin

AddButton.Enabled := CanAdd();

End;

Procedure TFindPathForm.EndEditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Begin

Handled := True;

End;

Procedure TFindPathForm.EndEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

CheckShftAndArrows(Key, Shift);

End;

Procedure TFindPathForm.EndEditKeyPress(Sender: TObject; Var Key: Char);

Begin

CheckComboButtons(Key, EndEdit, NUMBERS);

CheckRange(Key, EndEdit, CountOfTowns, 1);

End;

End.

Unit AddLine;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.Grids, Instruction,

MainForm;

Type

TAddLineForm = Class(TForm)

StartLabel: TLabel;

EndEdit: TEdit;

StartEdit: TEdit;

StartTLabel: TLabel;

EndLabel: TLabel;

AddButton: TButton;

CancelButton: TButton;

Procedure CancelButtonClick(Sender: TObject);

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

Shift: TShiftState);

Procedure EndEditChange(Sender: TObject);

Procedure StartEditChange(Sender: TObject);

Procedure FormCreate(Sender: TObject);

Procedure EndEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure EndEditKeyPress(Sender: TObject; Var Key: Char);

Procedure StartEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure StartEditKeyPress(Sender: TObject; Var Key: Char);

Procedure AddButtonClick(Sender: TObject);

Procedure EndEditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Procedure StartEditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Function FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Function CanAdd(): Boolean;

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

AddLineForm: TAddLineForm;

CanAddTown: Boolean = True;

Const

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

Implementation

Uses

GraphLinkedList;

{$R \*.dfm}

Procedure CenterFormOnScreen(AddLineForm: TAddLineForm);

Begin

AddLineForm.Left := (Screen.Width - AddLineForm.Width) Div 2;

AddLineForm.Top := (Screen.Height - AddLineForm.Height) Div 2;

End;

Procedure CheckComboButtons(Var Key: Char; Edit: TEdit;

Const Chariki: TSysCharSet);

Begin

If (Key = #22) Or ((Key = 'v') And (GetKeyState(VK\_CONTROL) < 0)) Then

Key := #0;

If Not CharInSet(Key, Chariki) And (Key <> #8) Then

Key := #0;

End;

Procedure CheckShftAndArrows(Var Key: Word; Shift: TShiftState);

Begin

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

Key := 0;

If (Key = VK\_LEFT) Or (Key = VK\_UP) Then

Key := 0

Else If (Key = VK\_RIGHT) Or (Key = VK\_DOWN) Then

Key := 0;

End;

Procedure CheckRange(Var Key: Char; Edit: TEdit; Const MAX, MIN: Integer);

Var

Value: Integer;

Begin

Value := 0;

If TryStrToInt(Edit.Text + Key, Value) Then

Begin

If (Value > MAX) Or (Value < MIN) Then

Key := #0;

End;

End;

Procedure TAddLineForm.AddButtonClick(Sender: TObject);

Begin

AddLink(StrToInt(StartEdit.Text), StrToInt(EndEdit.Text));

CanAddTown := False;

Close;

End;

Procedure TAddLineForm.CancelButtonClick(Sender: TObject);

Begin

StartEdit.Text := '';

EndEdit.Text := '';

Close;

End;

Procedure TAddLineForm.FormCreate(Sender: TObject);

Begin

AddButton.Enabled := False;

End;

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

Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

FormHelp := False;

End;

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

Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

Function TAddLineForm.CanAdd(): Boolean;

Begin

CanAdd := (StartEdit.Text <> '') And (EndEdit.Text <> '') And

(StartEdit.Text <> EndEdit.Text)

End;

Procedure TAddLineForm.StartEditChange(Sender: TObject);

Begin

AddButton.Enabled := CanAdd();

End;

Procedure TAddLineForm.StartEditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Begin

Handled := True;

End;

Procedure TAddLineForm.StartEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

CheckShftAndArrows(Key, Shift);

End;

Procedure TAddLineForm.StartEditKeyPress(Sender: TObject; Var Key: Char);

Begin

CheckComboButtons(Key, StartEdit, NUMBERS);

CheckRange(Key, StartEdit, CountOfTowns, 1);

End;

Procedure TAddLineForm.EndEditChange(Sender: TObject);

Begin

AddButton.Enabled := CanAdd();

End;

Procedure TAddLineForm.EndEditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Begin

Handled := True;

End;

Procedure TAddLineForm.EndEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

CheckShftAndArrows(Key, Shift);

End;

Procedure TAddLineForm.EndEditKeyPress(Sender: TObject; Var Key: Char);

Begin

CheckComboButtons(Key, EndEdit, NUMBERS);

CheckRange(Key, EndEdit, CountOfTowns, 1);

End;

End.

Unit MainForm;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Menus, Instruction, Developer,

Vcl.StdCtrls, Vcl.ExtDlgs, Vcl.Grids, Vcl.ExtCtrls;

Type

ERRORS\_LIST = (CORRECT, NOT\_READABLE, NOT\_WRITEABLE, FILE\_EMPTY, LINE\_ERR,

INPUT\_ERR, NUMBER\_ERR, MATRIX\_ERR, PATH\_ERR);

TMainTaskForm = Class(TForm)

MainFormMenu: TMainMenu;

FileMenu: TMenuItem;

InstructionMenu: TMenuItem;

DeveloperMenu: TMenuItem;

OpenMenu: TMenuItem;

SaveMenu: TMenuItem;

N1: TMenuItem;

QuitMenu: TMenuItem;

OpenFile: TOpenDialog;

SaveTextFile: TSaveTextFileDialog;

AddButton: TButton;

AddLineButton: TButton;

FindRouteButton: TButton;

ClearButton: TButton;

ScrollBox: TScrollBox;

TownPaintBox: TPaintBox;

Label1: TLabel;

Procedure DeveloperMenuClick(Sender: TObject);

Procedure InstructionMenuClick(Sender: TObject);

Procedure GetDataFromFile(Var F: TextFile; Sender: TObject);

Function FileReading(Var F: TextFile): ERRORS\_LIST;

Procedure QuitMenuClick(Sender: TObject);

Procedure FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Procedure SaveMenuClick(Sender: TObject);

Procedure OpenMenuClick(Sender: TObject);

Function FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Procedure AddButtonClick(Sender: TObject);

Procedure AddLineButtonClick(Sender: TObject);

Procedure ClearButtonClick(Sender: TObject);

Procedure FindRouteButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Const

ERRORS: Array [ERRORS\_LIST] Of String = ('', 'Файл закрыт для чтения!',

'Файл закрыт для записи!', 'Файл пуст!', 'Неверное число строк в файле',

'Неверное формат!', 'Неверное значение!',

'Неверное значение в матрице смежности!', 'В пути указан неверный путь!');

Var

MainTaskForm: TMainTaskForm;

IsEdited: Boolean = False;

Saved: Boolean = True;

Implementation

Uses

AddLine,

GraphLinkedList,

FindPath;

{$R \*.dfm}

Var

PerformCloseQuery: Boolean = True;

CtrlPressed: Boolean = False;

Procedure TMainTaskForm.AddLineButtonClick(Sender: TObject);

Var

AddLineForm: TAddLineForm;

Begin

If CountOfTowns > 1 Then

Begin

AddLineForm := TAddLineForm.Create(Self);

AddLineForm.ShowModal;

AddLineForm.Free;

DrawLines(TownPaintBox);

End

Else

Application.MessageBox('Количество городов меньше двух!', 'Ошибка',

MB\_OK + MB\_ICONERROR);

AddButton.Enabled := CanAddTown;

End;

Procedure TMainTaskForm.ClearButtonClick(Sender: TObject);

Begin

ClearGraph();

With TownPaintBox.Canvas Do

FillRect(ClipRect);

AddButton.Enabled := True;

Saved := True;

IsEdited := False;

End;

Procedure TMainTaskForm.DeveloperMenuClick(Sender: TObject);

Var

DeveloperForm: TDeveloperForm;

Begin

DeveloperForm := TDeveloperForm.Create(Self);

DeveloperForm.ShowModal;

DeveloperForm.Free;

End;

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

Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

InstructionMenuClick(Self)

End;

Procedure TMainTaskForm.InstructionMenuClick(Sender: TObject);

Var

InstructionForm: TInstructionForm;

Begin

InstructionForm := TInstructionForm.Create(Self);

InstructionForm.ShowModal;

InstructionForm.Free;

End;

Function IsReadable(Var F: TextFile): ERRORS\_LIST;

Var

ERRORS: ERRORS\_LIST;

Begin

ERRORS := CORRECT;

Try

Try

Reset(F);

Finally

CloseFile(F);

End;

Except

ERRORS := NOT\_READABLE;

End;

IsReadable := ERRORS;

End;

Function CheckNumOfLines(Var F: TextFile): ERRORS\_LIST;

Var

I: Integer;

Str: String;

NumOfString: Integer;

Error: ERRORS\_LIST;

Begin

I := 1;

Str := '';

Error := CORRECT;

Reset(F);

Readln(F, Str);

If TryStrToInt(Str, NumOfString) Then

Begin

While Not EOF(F) Do

Begin

Readln(F, Str);

Inc(I);

End;

CloseFile(F);

If (I <> NumOfString + 2) Then

Error := LINE\_ERR;

End

Else

Error := NUMBER\_ERR;

CheckNumOfLines := Error;

End;

Function CheckFileData(Var F: TextFile): ERRORS\_LIST;

Var

Error: ERRORS\_LIST;

Num, I, J: Integer;

TownsCount: Integer;

StartT, EndT: Integer;

GraphMat: TGraphMatrix;

Begin

Error := CORRECT;

Reset(F);

Try

Readln(F, TownsCount);

Except

Error := NUMBER\_ERR;

End;

If Error = Correct Then

Begin

I := 0;

J := 0;

SetLength(GraphMat, TownsCount, TownsCount);

While (I < TownsCount) And (Error = CORRECT) Do

Begin

While (J < TownsCount) And (Error = Correct) Do

Begin

Try

Read(F, Num);

If (Num = 1) Then

GraphMat[I, J] := True

Else If Num = 0 Then

GraphMat[I, J] := False

Else

Error := MATRIX\_ERR;

Except

Error := NUMBER\_ERR;

End;

Inc(J);

End;

Readln(F);

Inc(I);

J := 0;

End;

If Error = Correct Then

Begin

For I := 0 To High(GraphMat) Do

For J := 0 To High(GraphMat) Do

If GraphMat[I, J] <> GraphMat[J, I] Then

Error := MATRIX\_ERR;

End;

If Error = CORRECT Then

Begin

Try

Read(F, StartT, EndT);

If (StartT > TownsCount) Or (StartT < 0) Or (EndT > TownsCount)

Or (EndT < 0) Or (StartT = EndT) Then

Error := PATH\_ERR;

Except

Error := NUMBER\_ERR;

End;

End;

End;

CloseFile(F);

CheckFileData := Error;

End;

Procedure TMainTaskForm.GetDataFromFile(Var F: TextFile; Sender: TObject);

Var

I, J, Num, Count: Integer;

StartT, EndT: Integer;

Begin

I := 0;

J := 0;

Reset(F);

Readln(F, Count);

For Num := 1 To Count Do

AddTown();

While (I < CountOfTowns) Do

Begin

While (J < CountOfTowns) Do

Begin

Read(F, Num);

If (Num = 1) Then

GraphMatrix[I, J] := True

Else If Num = 0 Then

GraphMatrix[I, J] := False;

Inc(J);

End;

Readln(F);

Inc(I);

J := 0;

End;

Read(F, StartT, EndT);

CloseFile(F);

If FindPathDFS(StartT, EndT) = Nil Then

ShowMessage('Дороги между данными городами нет!');

Saved := False;

IsEdited := True;

SaveMenu.Enabled := Not Saved;

End;

Function TMainTaskForm.FileReading(Var F: TextFile): ERRORS\_LIST;

Var

ERRORS: ERRORS\_LIST;

Begin

ERRORS := CORRECT;

Reset(F);

If EOF(F) Then

ERRORS := FILE\_EMPTY;

CloseFile(F);

If (ERRORS = CORRECT) Then

ERRORS := CheckNumOfLines(F);

If (ERRORS = CORRECT) Then

ERRORS := CheckFileData(F);

If (ERRORS = CORRECT) Then

Begin

ClearButtonClick(Self);

GetDataFromFile(F, Self);

DrawPath(TownPaintBox);

End;

FileReading := ERRORS;

End;

Procedure TMainTaskForm.FindRouteButtonClick(Sender: TObject);

Var

FindPathForm: TFindPathForm;

Begin

If CountOfTowns > 1 Then

Begin

FindPathForm := TFindPathform.Create(Self);

FindPathForm.ShowModal;

FindPathForm.Free;

DrawPath(TownPaintBox);

End

Else

Application.MessageBox('Количество городов меньше двух!', 'Ошибка',

MB\_OK + MB\_ICONERROR);

AddButton.Enabled := CanAddTown;

SaveMenu.Enabled := Not Saved;

End;

Procedure TMainTaskForm.OpenMenuClick(Sender: TObject);

Var

Error: ERRORS\_LIST;

F: TextFile;

Num, FileName: String;

Begin

If OpenFile.Execute Then

Begin

FileName := OpenFile.FileName;

AssignFile(F, FileName);

Error := IsReadable(F);

If Error = CORRECT Then

Error := FileReading(F);

If Error <> CORRECT Then

Application.MessageBox(PWideChar(ERRORS[Error]), 'Ошибка',

MB\_OK Or MB\_ICONINFORMATION);

End;

End;

Procedure TMainTaskForm.AddButtonClick(Sender: TObject);

Begin

If CountOfTowns < 15 Then

Begin

With TownPaintBox.Canvas Do

FillRect(ClipRect);

AddTown();

DrawTowns(TownPaintBox);

End

Else

AddButton.Enabled := False;

End;

Procedure TMainTaskForm.SaveMenuClick(Sender: TObject);

Var

Error: ERRORS\_LIST;

F: TextFile;

FileName, PathStr: String;

I: Integer;

Begin

If SaveTextFile.Execute Then

Begin

FileName := SaveTextFile.FileName;

FileName := ChangeFileExt(FileName, '.txt');

AssignFile(F, FileName);

If FileExists(FileName) Then

Begin

Error := IsReadable(F);

If Error = CORRECT Then

Begin

Rewrite(F);

If Path <> Nil Then

Begin

For I := 0 To High(Path) Do

PathStr := Concat(PAthStr + IntToStr(Path[I]) + ', ');

Write(F, 'Ваш путь: ', PathStr);

End

Else

Write(F, 'Путь не найден!');

CloseFile(F);

Saved := True;

End;

If Error <> CORRECT Then

Begin

Application.MessageBox(PWideChar(ERRORS[Error]), 'Ошибка',

MB\_OK Or MB\_ICONINFORMATION);

Saved := False;

End;

End

Else

Begin

Rewrite(F);

If Path <> Nil Then

Begin

For I := 0 To High(Path) Do

PathStr := Concat(PAthStr + IntToStr(Path[I]) + ', ');

Write(F, 'Ваш путь: ', PathStr);

End

Else

Write(F, 'Путь не найден!');

CloseFile(F);

Saved := True;

End;

End;

End;

Procedure TMainTaskForm.FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Var

Confirmation: Integer;

Begin

If PerformCloseQuery Then

Begin

If (Saved = False) Then

Begin

Confirmation := Application.MessageBox

('Вы не сохранили файл, хотите ли сохранить?', 'Выход',

MB\_YESNOCANCEl + MB\_ICONQUESTION + MB\_DEFBUTTON2);

Case Confirmation Of

MrYes:

Begin

SaveMenuClick(Sender);

If Saved = True Then

CanClose := True

Else

FormCloseQuery(Sender, CanClose);

End;

MrNo:

CanClose := True;

MrCancel:

CanClose := False;

End;

End

Else

Begin

Confirmation := Application.MessageBox

('Вы действительно хотите выйти?', 'Выход',

MB\_YESNO + MB\_ICONQUESTION + MB\_DEFBUTTON2);

CanClose := Confirmation = IDYES;

End;

End;

End;

Procedure TMainTaskForm.QuitMenuClick(Sender: TObject);

Var

Confirmation: Integer;

Begin

PerformCloseQuery := False;

If (Saved = False) Then

Begin

Confirmation := Application.MessageBox

('Вы не сохранили файл, хотите ли сохранить?', 'Выход',

MB\_YESNOCANCEl + MB\_ICONQUESTION + MB\_DEFBUTTON2);

Case Confirmation Of

MrYes:

Begin

SaveMenuClick(Sender);

If Saved = True Then

Close

Else

QuitMenuClick(Sender);

End;

MrNo:

Close;

End;

End

Else

Begin

Confirmation := Application.MessageBox('Вы действительно хотите выйти?',

'Выход', MB\_YESNO + MB\_ICONQUESTION + MB\_DEFBUTTON2);

If Confirmation = IDYES Then

Close;

End;

PerformCloseQuery := True;

End;

End.

**Код программы Java:**

import java.io.File;

import java.io.FileWriter;

import java.util.Objects;

import java.util.Scanner;

public class Main {

enum ErrCode {

CORRECT,

FILE\_NOT\_EXIST,

NOT\_TXT,

INPUT\_ERR,

NOT\_READABLE,

FILE\_EMPTY,

NOT\_WRITEABLE,

RANGE\_ERR,

EXTRA\_DATA,

SAME\_TOWNS

}

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

"Такого файла не существует!",

"Файл не .txt!",

"Проверьте корректность ввода данных!",

"Файл закрыт для чтения!",

"Файл пуст!",

"Файл закрыт для записи!",

"Данные не входят в диапазон!",

"Лишние данные в файле!",

"Пуь не может начинаться и заканчиваться в одном месте!"

};

static final String INSTRUCTION = "\n1. Номера телефонов должны начинаться с кода.\n" +

"2. Чтобы добавлять контанты из файла нужно записывать количество городов на первой строке.\n " +

" Матрицу смежности для графа с городами на второй.\n" +

" Начальный и конечный город через пробел на последней строке.\n" +

"3. При добавлении нового города старые дороги пропадают";

enum ChooseAction {

addTown("Добавить город"),

addRoad("Добавить дорогу"),

addCityFromFile("Добавить города и дороги из файла"),

findPathBetweenTowns("Найти путь от А до Б"),

clearGraph("Очистить города"),

saveResult("Сохранить путь в файл"),

exitProg("Заверишть программу");

private final String info;

ChooseAction (String inf) {

this.info = inf;

}

private String getInf() {

return this.ordinal() + ") " + this.info;

}

}

static class StackList {

int value;

StackList next;

}

static StackList top = null;

public static void iStack() {

top = null;

}

public static void pushStack(int value) {

StackList newItem = new StackList();

newItem.value = value;

newItem.next = top;

top = newItem;

}

public static void popStack() {

StackList removeItem = top;

top = removeItem.next;

}

public static String stackToStr() {

String pathStr = " ";

StackList item = top;

if (item == null) {

pathStr = "Путь не найден!";

} else {

while (item != null) {

pathStr = pathStr + " ," + item.value;

item = item.next;

}

StringBuilder reversed = new StringBuilder(pathStr).reverse();

pathStr = "Ваш путь: " + reversed;

}

return pathStr;

}

static class GrLinkedList {

int number;

GrLinkedList next;

}

static GrLinkedList head = null;

static int countOfTowns = 1;

static boolean[][] graphMatrix;

static boolean[] visited;

static String pathStr;

public static void make() {

head = new GrLinkedList();

head.next = null;

head.number = countOfTowns;

}

public static void addLink(int startTown, int endTown) {

graphMatrix[startTown - 1][endTown - 1] = true;

graphMatrix[endTown - 1][startTown - 1] = true;

}

public static void addTown() {

GrLinkedList town;

if (head == null)

make();

else {

town = head;

while (town.next != null)

town = town.next;

town.next = new GrLinkedList();

town = town.next;

countOfTowns++;

town.number = countOfTowns;

town.next = null;

}

graphMatrix = new boolean[countOfTowns][countOfTowns];

for (int i = 0; i < graphMatrix.length - 1; i++) {

for (int j = 0; j < graphMatrix.length - 1; j++)

graphMatrix[i][j] = false;

}

}

public static void clearGraph() {

GrLinkedList buff;

graphMatrix = null;

buff = head;

while (buff != null) {

head = buff.next;

buff = null;

buff = head;

}

head = null;

countOfTowns = 1;

visited = null;

}

public static String findPathDfs(int startPoint, int endPoint) {

visited = new boolean[countOfTowns];

iStack();

for (int i = 0; i < countOfTowns; i++)

visited[i] = false;

if (dfs(startPoint, endPoint)) {

return stackToStr();

} else {

return stackToStr();

}

}

public static boolean dfs(int startPoint, int endPoint) {

visited[startPoint - 1] = true;

boolean ready = false;

int j = 0;

int neighbor = 0;

pushStack(startPoint);

if (startPoint == endPoint) {

ready = true;

} else {

while ((j < graphMatrix.length) && !ready) {

if (graphMatrix[startPoint - 1][j]) {

neighbor = j + 1;

if (!visited[neighbor - 1]) {

if (dfs(neighbor, endPoint)) {

ready = true;

}

}

}

j++;

}

if (!ready) {

popStack();

}

}

return ready;

}

public static boolean doMenu (Scanner input) {

boolean close = false;

System.out.print("Введите действие: ");

ChooseAction option = getChoice(input);

System.out.println();

switch (option) {

case addTown -> {

addTown();

System.out.println("Город успешно добавлен!");

System.out.println("Количество городов: " + countOfTowns + "\n");

}

case addCityFromFile -> {

pathStr = readFileMatrix(input);

visualiseGraph();

System.out.println(pathStr);

}

case addRoad -> {

if (countOfTowns > 1) {

addNewRoad(input);

System.out.println("Дорога успешно добавлена!");

visualiseGraph();

} else {

System.out.println("Количество городов должно быть больше двух!");

}

}

case findPathBetweenTowns -> {

if (countOfTowns > 1) {

pathStr = getPath(input);

System.out.println(pathStr);

} else {

System.out.println("Количество городов должно быть больше двух!");

}

}

case clearGraph -> {

clearGraph();

System.out.println("Горорда очищены!");

}

case saveResult -> {

if (!Objects.equals(pathStr, ""))

saveUpDownList(input, pathStr);

else

System.out.println("Вы ещё не находили путь!");

}

case exitProg ->

close = true;

}

return close;

}

public static void printMenu() {

ChooseAction[] choices = ChooseAction.values();

for (ChooseAction choice : choices) {

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

}

}

public static void addNewRoad (Scanner input) {

int startPoint;

int endPoint;

do {

System.out.print("Введите стартовый город: ");

startPoint = getNumConsole(input, 1, countOfTowns);

System.out.print("Введите конечный город: ");

endPoint = getNumConsole(input, 1, countOfTowns);

if (startPoint == endPoint) {

System.out.println("Дорога не может начинаться и заканчиваться в одном городе!");

}

} while (startPoint == endPoint);

addLink(startPoint, endPoint);

}

public static void visualiseGraph () {

System.out.println("Ваши дороги: ");

System.out.print(" ");

for (int k = 0; k < countOfTowns; k++)

System.out.print(k + 1 + " ");

System.out.println();

for (int i = 0; i < countOfTowns; i++) {

System.out.print(i + 1 + "| ");

for (int j = 0; j < countOfTowns; j++) {

if (graphMatrix[i][j]) {

System.out.print("1");

} else {

System.out.print("0");

}

System.out.print(" ");

}

System.out.println();

}

}

public static String getPath (Scanner input) {

String pathStr;

int startPoint;

int endPoint;

do {

System.out.print("Введите стартовый город: ");

startPoint = getNumConsole(input, 1, countOfTowns);

System.out.print("Введите конечный город: ");

endPoint = getNumConsole(input, 1, countOfTowns);

if (startPoint == endPoint) {

System.out.println("Дорога не может начинаться и заканчиваться в одном городе!");

}

} while (startPoint == endPoint);

pathStr = findPathDfs(startPoint, endPoint);

return pathStr;

}

public static void printError (ErrCode error) {

System.out.println("\n" + ERRORS[error.ordinal()] + "\nПовторите попытку\n");

}

static ErrCode readOneNum(Scanner inputScanner, int[] numberArr, final int MIN, final int MAX, boolean isFile) {

int number = 0;

ErrCode error;

error = ErrCode.CORRECT;

try {

if (isFile)

number = (inputScanner.nextInt());

else

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

} catch (NumberFormatException e) {

error = ErrCode.INPUT\_ERR;

}

if (error == ErrCode.CORRECT && ((number < MIN) || (number > MAX)))

error = ErrCode.RANGE\_ERR;

numberArr[0] = error == ErrCode.CORRECT ? number : 0;

return error;

}

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

ErrCode err;

int[] numberArr = {0};

do {

err = readOneNum(input, numberArr, MIN, MAX, false);

if (err != ErrCode.CORRECT) {

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

System.out.println("\nВведите снова");

}

} while (err != ErrCode.CORRECT);

return numberArr[0];

}

static ChooseAction getChoice(Scanner input) {

int choice;

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

choice = getNumConsole(input, 0, maxChoice);

return ChooseAction.values()[choice];

}

public static String readPath (Scanner inputScanner) {

String pathTofile;

ErrCode error;

do {

System.out.print("Введите путь к txt файлу: ");

pathTofile = inputScanner.nextLine();

if (pathTofile.isEmpty()) {

pathTofile = inputScanner.nextLine();

}

if (!pathTofile.endsWith(".txt")) {

error = ErrCode.NOT\_TXT;

} else {

error = ErrCode.CORRECT;

}

if (error != ErrCode.CORRECT)

printError(error);

} while (error != ErrCode.CORRECT);

return pathTofile;

}

public static File fileReading (Scanner inputScanner) {

ErrCode error;

String pathToFile;

File file;

do {

error = ErrCode.CORRECT;

pathToFile = readPath(inputScanner);

file = new File(pathToFile);

if (!file.exists())

error = ErrCode.FILE\_NOT\_EXIST;

if ((error == ErrCode.CORRECT) && (!file.canRead()))

error = ErrCode.NOT\_READABLE;

if ((error == ErrCode.CORRECT) && (file.length() == 0))

error = ErrCode.FILE\_EMPTY;

if (error != ErrCode.CORRECT)

printError(error);

} while (error != ErrCode.CORRECT);

return file;

}

public static File fileWriting(Scanner inputScanner) {

ErrCode error;

File file;

String pathToFile;

do {

pathToFile = readPath(inputScanner);

file = new File(pathToFile);

error = ErrCode.CORRECT;

if (!file.exists())

error = ErrCode.FILE\_NOT\_EXIST;

if ((error == ErrCode.CORRECT) && !file.canWrite())

error = ErrCode.NOT\_WRITEABLE;

if (error != ErrCode.CORRECT)

printError(error);

} while (error != ErrCode.CORRECT);

return file;

}

static ErrCode checkSpaceInFile(String bufStr) {

int i;

ErrCode error;

i = 0;

error = ErrCode.CORRECT;

while ((error == ErrCode.CORRECT) && (i < bufStr.length())) {

if (bufStr.charAt(i) != ' ')

error = ErrCode.EXTRA\_DATA;

i++;

}

return error;

}

public static boolean[][] readMatrix(Scanner inputScanner, int[] numberArr, int m, int n, int option) {

int i;

int j;

boolean[][] matrix;

String bufStr;

ErrCode error;

error = ErrCode.CORRECT;

i = 0;

matrix = new boolean[n][m];

while (i < n && error == ErrCode.CORRECT) {

j = 0;

while (j < m && error == ErrCode.CORRECT) {

if (option == 1) {

error = readOneNum(inputScanner, numberArr, 0, 1, true);

if (error == ErrCode.CORRECT) {

matrix[i][j] = numberArr[0] != 0;

}

j++;

}

}

if (option == 1 && error == ErrCode.CORRECT) {

bufStr = inputScanner.nextLine();

error = checkSpaceInFile(bufStr);

}

i++;

}

if (error != ErrCode.CORRECT) {

matrix = null;

printError(error);

}

return matrix;

}

public static String readFileMatrix(Scanner inputScanner) {

int[] numberArr = new int[1];

int[] numberCountOfTowns = new int[1];

int[] numberStart = new int[1];

int[] numberEnd = new int[1];

ErrCode error;

File file;

do {

file = fileReading(inputScanner);

try(Scanner scanFile = new Scanner(file)) {

error = readOneNum(scanFile, numberCountOfTowns, 1, 30, true);

if (error != ErrCode.CORRECT)

printError(error);

else {

countOfTowns = numberCountOfTowns[0];

graphMatrix = readMatrix(scanFile, numberArr, countOfTowns, countOfTowns, 1);

error = readOneNum(scanFile, numberStart, 1, countOfTowns, true);

if (error == ErrCode.CORRECT) {

error = readOneNum(scanFile, numberEnd, 1, countOfTowns, true);

}

if (numberStart[0] == numberEnd[0]) {

error = ErrCode.SAME\_TOWNS;

}

}

} catch (Exception e) {

error = ErrCode.NOT\_READABLE;

printError(error);

}

if (graphMatrix == null)

error = ErrCode.INPUT\_ERR;

if (error != ErrCode.CORRECT) {

printError(error);

}

} while (error != ErrCode.CORRECT);

return findPathDfs(numberStart[0], numberEnd[0]);

}

public static void saveUpDownList (Scanner inputScanner, String pathStr) {

File file = fileWriting(inputScanner);

try (FileWriter writer = new FileWriter(file, true)) {

writer.write(pathStr);

System.out.println("Путь сохранен успешно.\n");

} catch (Exception e) {

printError(ErrCode.NOT\_WRITEABLE);

}

}

public static void main(String[] args) {

boolean isExit;

Scanner input = new Scanner(System.in);

System.out.println(INSTRUCTION);

do {

printMenu();

isExit = doMenu(input);

} while (!isExit);

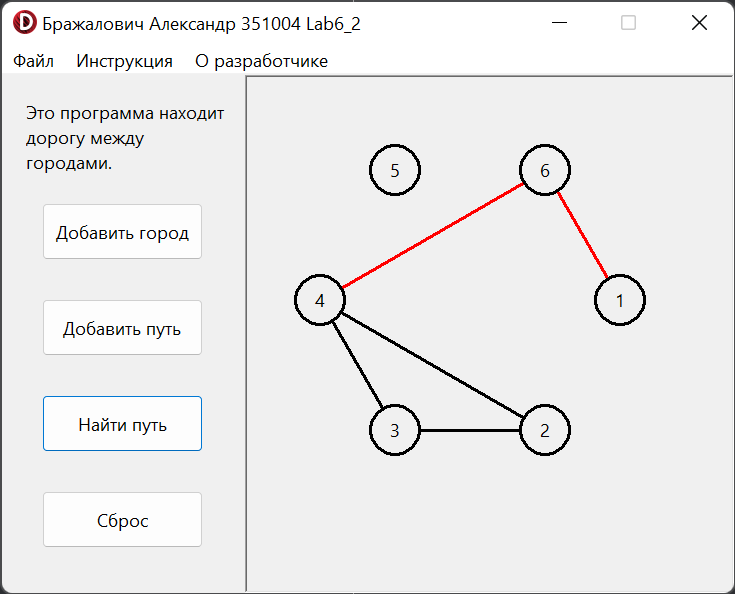
input.close();

}

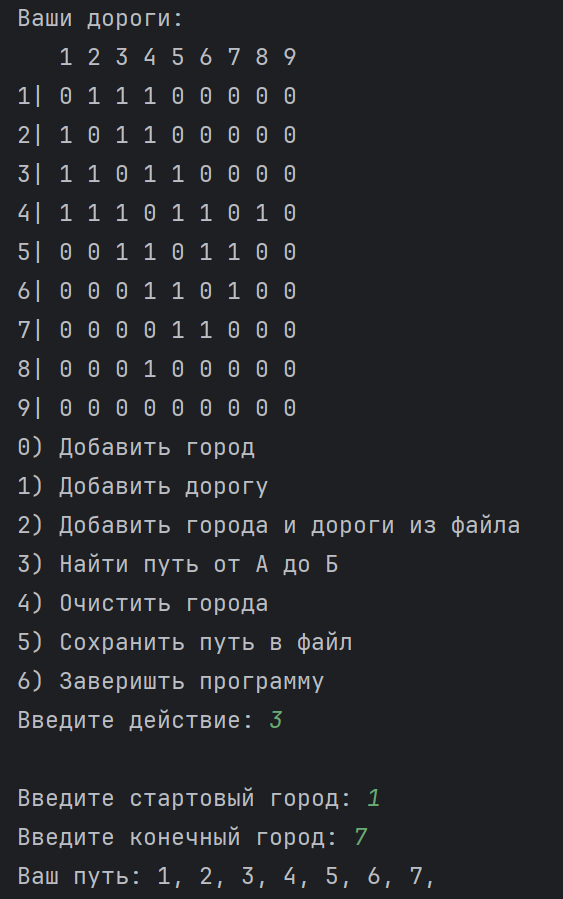
}

**Скриншоты:**

**Delphi:**

****

**Java:**

****

**Блок-схема:**

