УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №7.2

по предмету «Основы алгоритмизации и программирования»

Вариант 3

Выполнил:

Бражалович А.И.

Гр. 351004

Проверил:

Данилова Г. В.

Минск 2024

**Задание:**

Преобразовать матрицу смежности в списки инцидентности.

**Код программы Delphi:**

Unit MainUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs,

Vcl.Menus, InstructionUnit, DeveloperUnit, Vcl.StdCtrls,

Vcl.Grids, Vcl.Imaging.Pngimage, Vcl.ExtCtrls;

Type

ERRORS\_CODE = (CORRECT, INCORRECT\_RANGE, EXTRA\_DATA, IS\_NOT\_READABLE,

IS\_NOT\_WRITEABLE, INCORRECT\_DATA\_IN\_FILE, INCORRECT\_NUMS\_AMOUNT);

TStringGridEx = Class(TStringGrid);

TMatrix = Array Of Array Of Integer;

TArr = Array Of Integer;

PVertex = ^TVertex;

TVertex = Record

Value: Char;

Next: PVertex;

End;

Type

TMainForm = Class(TForm)

MainMenu1: TMainMenu;

FileButton: TMenuItem;

OpenFileButton: TMenuItem;

SaveFileButton: TMenuItem;

LineSeparator: TMenuItem;

ExitButton: TMenuItem;

InstructionButton: TMenuItem;

DeveloperButton: TMenuItem;

OpenFile: TOpenDialog;

SaveFile: TSaveDialog;

OrderEdit: TEdit;

MatrixGrid: TStringGrid;

ResultButton: TButton;

OrderLabel: TLabel;

MatrixGridLabel: TLabel;

Task: TLabel;

ResultGrid: TStringGrid;

ResultLabel: TLabel;

Function ReadFileData(Var F: TextFile; Sender: TObject): ERRORS\_CODE;

Function FormHelp(Command: Word; Data: NativeInt; Var CallHelp: Boolean): Boolean;

Procedure InstructionButtonClick(Sender: TObject);

Procedure SaveOnClick(Sender: TObject);

Procedure DeveloperOnClick(Sender: TObject);

Procedure ExitOnClick(Sender: TObject);

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

Procedure OpenOnClick(Sender: TObject);

Procedure OrderEditChange(Sender: TObject);

Procedure OrderEditContextPopup(Sender: TObject; MousePos: TPoint; Var Handled: Boolean);

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

Procedure OrderEditKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

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

Procedure MatrixGridKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Procedure MatrixGridSetEditText(Sender: TObject; ACol, ARow: Integer; Const Value: String);

Procedure ResultButtonClick(Sender: TObject);

Procedure ResultEditChange(Sender: TObject);

Procedure ResultGridDrawCell(Sender: TObject; ACol, ARow: Integer; Rect: TRect; State:

TGridDrawState);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

MainForm: TMainForm;

Implementation

{$R \*.dfm}

Var

IsSaved: Boolean = True;

AdjacencyArr: Array Of PVertex;

Const

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

BACKSPACE = #8;

NONE = #0;

ERRORS: Array [ERRORS\_CODE] Of String = ('', 'Значение не попадает в диапазон!',

'Лишние данные в файле!',

'Файл закрыт для чтения!', 'Файл закрыт для записи!',

'Некорректный тип данных в файле!',

'Неправильное количество чисел в файле!');

MIN\_N = 1;

MAX\_N = 9;

MIN\_NUM = 0;

MAX\_NUM = 1;

Function IsValidRange(Text: String; MIN, MAX: Integer): Boolean;

Var

IsValidInput: Boolean;

Num: Real;

Begin

IsValidInput := True;

Num := StrToFloat(Text);

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

IsValidInput := False;

IsValidRange := IsValidInput;

End;

Procedure TMainForm.DeveloperOnClick(Sender: TObject);

Var

DeveloperForm: TDeveloperForm;

Begin

DeveloperForm := TDeveloperForm.Create(Self);

DeveloperForm.ShowModal;

DeveloperForm.Free;

End;

Procedure TMainForm.InstructionButtonClick(Sender: TObject);

Var

InstructionForm: TInstructionForm;

Begin

InstructionForm := TInstructionForm.Create(Self);

InstructionForm.ShowModal;

InstructionForm.Free;

End;

Procedure FillGrid(RowNum: Integer; Grid: TStringGrid);

Var

I: Integer;

Begin

Grid.ColCount := RowNum + 1;

Grid.RowCount := RowNum + 1;

If RowNum < 5 Then

Begin

Grid.Width := (Grid.DefaultColWidth + 4 - Grid.GridLineWidth) \*

Grid.ColCount;

Grid.Height := (Grid.DefaultRowHeight + 4 - Grid.GridLineWidth) \*

Grid.RowCount;

End

Else

Begin

Grid.Width := (Grid.DefaultColWidth + 4 - Grid.GridLineWidth) \* 5 + 32;

Grid.Height := (Grid.DefaultRowHeight + 4 - Grid.GridLineWidth) \* 5 + 32;

End;

Grid.Cells[0, 0] := '/';

For I := 1 To RowNum Do

Begin

Grid.Cells[0, I] := Chr(Ord('A') + I - 1);

Grid.Cells[I, 0] := Chr(Ord('A') + I - 1);

End;

End;

Procedure ClearGrid(Grid: TStringGrid);

Var

J, I: Integer;

Begin

For I := 0 To Grid.ColCount - 1 Do

For J := 0 To Grid.RowCount - 1 Do

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

End;

Procedure CheckRightKeyPress(Var Key: Char; SelLength: Integer; Text: String; Const MIN, MAX: Integer);

Var

IsValidInput: Boolean;

Begin

IsValidInput := (CharInSet(Key, DIGITS) Or (Key = BACKSPACE)) And

(Key <> '0');

If (SelLength > 0) And (SelLength < Length(Text)) Then

Key := NONE;

If (IsValidInput) And (Key <> BACKSPACE) Then

IsValidInput := IsValidRange(Text + Key, MIN, MAX);

If Not IsValidInput Then

Key := NONE;

End;

Procedure DeleteLists();

Var

CurrentNode: PVertex;

I: Integer;

Begin

For I := 0 To Length(AdjacencyArr) - 1 Do

Begin

While AdjacencyArr[I].Next <> Nil Do

Begin

CurrentNode := AdjacencyArr[I].Next;

AdjacencyArr[I].Next := CurrentNode.Next;

Dispose(CurrentNode);

End;

Dispose(AdjacencyArr[I]);

End;

SetLength(AdjacencyArr, 0);

End;

Procedure TMainForm.OrderEditChange(Sender: TObject);

Var

Num: Integer;

Begin

If (OrderEdit.Text = '') Or Not TryStrToInt(OrderEdit.Text, Num) Then

Begin

MatrixGridLabel.Visible := False;

MatrixGrid.Visible := False;

ResultGrid.Visible := False;

ResultLabel.Visible := False;

ClearGrid(MatrixGrid);

End

Else

Begin

MatrixGridLabel.Visible := True;

MatrixGrid.Visible := True;

ClearGrid(MatrixGrid);

FillGrid(StrToInt(OrderEdit.Text), MatrixGrid);

End;

If Length(AdjacencyArr) > 0 Then

DeleteLists;

SaveFileButton.Enabled := False;

IsSaved := True;

ResultButton.Enabled := False;

End;

Procedure TMainForm.OrderEditContextPopup(Sender: TObject; MousePos: TPoint; Var Handled: Boolean);

Begin

Handled := True;

End;

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

Begin

If (Key = VK\_DELETE) Then

Key := 0;

If ((SsShift In Shift) Or (SsCtrl In Shift)) And

Not((Key = VK\_RIGHT) Or (Key = VK\_LEFT)) Then

Key := 0;

End;

Procedure TMainForm.OrderEditKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

CheckRightKeyDown(Key, Shift);

End;

Procedure TMainForm.OrderEditKeyPress(Sender: TObject; Var Key: Char);

Var

SelLength: Integer;

Text: String;

Begin

SelLength := OrderEdit.SelLength;

Text := OrderEdit.Text;

CheckRightKeyPress(Key, SelLength, Text, MIN\_N, MAX\_N);

End;

Procedure TMainForm.MatrixGridKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

CheckRightKeyDown(Key, Shift);

End;

Function IsAllCellFill(Grid: TStringGrid): Boolean;

Var

IsFilled: Boolean;

Col, Row, Num: Integer;

Begin

IsFilled := MainForm.OrderEdit.Text <> '';

Row := 1;

While IsFilled And (Row < Grid.RowCount) Do

Begin

Col := 1;

While IsFilled And (Col < Grid.ColCount) Do

Begin

If (MainForm.MatrixGrid.Cells[Col, Row] = '') Or

Not TryStrToInt(MainForm.MatrixGrid.Cells[Col, Row], Num) Then

IsFilled := False;

Inc(Col);

End;

Inc(Row);

End;

IsAllCellFill := IsFilled;

End;

Procedure TMainForm.MatrixGridKeyPress(Sender: TObject; Var Key: Char);

Var

MatrixGrid: TStringGridEx;

Text: String;

Begin

MatrixGrid := TStringGridEx(Sender);

If Assigned(MatrixGrid.InplaceEditor) Then

Begin

Text := MatrixGrid.InplaceEditor.Text;

If (Length(Text) > 0) And (Key <> BACKSPACE) Then

Key := NONE

Else If (Key <> '1') And (Key <> '0') And (Key <> BACKSPACE) Then

Key := NONE;

End;

End;

Procedure TMainForm.MatrixGridSetEditText(Sender: TObject; ACol, ARow: Integer; Const Value: String);

Begin

ResultButton.Enabled := IsAllCellFill(MatrixGrid);

ClearGrid(ResultGrid);

ResultGrid.Visible := False;

ResultLabel.Visible := False;

SaveFileButton.Enabled := False;

IsSaved := True;

If Length(AdjacencyArr) > 0 Then

DeleteLists;

End;

Procedure TMainForm.ExitOnClick(Sender: TObject);

Begin

Close;

End;

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

Var

Confirmation: Integer;

Begin

If (IsSaved = 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

SaveOnClick(Sender);

If IsSaved = 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;

Function TMainForm.FormHelp(Command: Word; Data: NativeInt; Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

End;

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

Var

Error: ERRORS\_CODE;

Begin

Error := CORRECT;

Try

Try

Reset(F);

Finally

CloseFile(F);

End;

Except

Error := IS\_NOT\_READABLE;

End;

IsReadable := Error;

End;

Function ReadFileNum(Var F: TextFile; Var Num: Integer; Const MIN, MAX: Integer) : ERRORS\_CODE;

Var

Error: ERRORS\_CODE;

Begin

Error := CORRECT;

Try

Read(F, Num);

Except

Error := INCORRECT\_DATA\_IN\_FILE;

End;

If (Error = CORRECT) And Not(IsValidRange(IntToStr(Num), MIN, MAX)) Then

Error := INCORRECT\_RANGE;

ReadFileNum := Error;

End;

Function CheckSpaceInFile(BufStr: String): ERRORS\_CODE;

Var

I: Integer;

Error: ERRORS\_CODE;

Begin

I := 1;

Error := CORRECT;

While (Error = CORRECT) And (I <= Length(BufStr)) Do

Begin

If BufStr[I] <> ' ' Then

Error := EXTRA\_DATA;

Inc(I);

End;

CheckSpaceInFile := Error;

End;

Function TMainForm.ReadFileData(Var F: TextFile; Sender: TObject): ERRORS\_CODE;

Var

Error: ERRORS\_CODE;

Num, Order: Integer;

Matrix: TMatrix;

I, J: Integer;

BufStr: String;

Begin

I := 0;

J := 0;

Reset(F);

Error := ReadFileNum(F, Num, MIN\_N + 1, MAX\_N);

If Error = CORRECT Then

Begin

Readln(F, BufStr);

Error := CheckSpaceInFile(BufStr);

End;

If Error = CORRECT Then

OrderEdit.Text := IntToStr(Num);

Order := Num;

SetLength(Matrix, Order, Order);

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

Begin

J := 0;

While (Error = CORRECT) And Not EOF(F) And (J < Order) Do

Begin

Error := ReadFileNum(F, Matrix[I, J], MIN\_NUM, MAX\_NUM);

Inc(J);

End;

Readln(F, BufStr);

Error := CheckSpaceInFile(BufStr);

Inc(I);

End;

If (Error = CORRECT) And (J <> Order) Then

Error := INCORRECT\_NUMS\_AMOUNT;

If Error = CORRECT Then

Begin

FillGrid(Num, MatrixGrid);

For I := 1 To MatrixGrid.RowCount - 1 Do

Begin

For J := 1 To MatrixGrid.ColCount - 1 Do

MatrixGrid.Cells[J, I] := IntToStr(Matrix[I - 1, J - 1]);

End;

ResultButton.Enabled := True;

End;

CloseFile(F);

ReadFileData := Error;

End;

Procedure CreateAdjacencyLists(Matrix: TMatrix);

Var

I, J, K: Integer;

Temp: PVertex;

Begin

SetLength(AdjacencyArr, Length(Matrix));

For I := 0 To Length(AdjacencyArr) - 1 Do

Begin

New(AdjacencyArr[I]);

AdjacencyArr[I].Next := Nil;

Temp := AdjacencyArr[I];

For J := 0 To Length(Matrix) - 1 Do

Begin

For K := 1 To Matrix[I, J] Do

Begin

New(Temp.Next);

Temp := Temp.Next;

Temp.Value := Chr(Ord('A') + J);

Temp.Next := Nil;

End;

End;

End;

End;

Procedure MakeResultGrid(RowNum: Integer; Grid: TStringGrid);

Begin

Grid.ColCount := 2;

Grid.RowCount := RowNum;

Grid.ColWidths[0] := Grid.DefaultColWidth;

Grid.ColWidths[1] := Grid.DefaultColWidth + 20 \* RowNum;

If RowNum < 4 Then

Begin

Grid.Height := (Grid.DefaultRowHeight + 4 - Grid.GridLineWidth) \*

Grid.RowCount + 2;

Grid.Width := Grid.ColWidths[0] + Grid.ColWidths[1] + 10;

End

Else

Begin

Grid.Height := (Grid.DefaultRowHeight + 4 - Grid.GridLineWidth) \* 3 + 2;

Grid.Width := Grid.ColWidths[0] + Grid.ColWidths[1] + 42;

End;

Grid.Visible := True;

End;

Procedure FillResultGrid(RowNum: Integer; Grid: TStringGrid);

Var

I: Integer;

ResultStr: String;

Temp: PVertex;

Begin

For I := 0 To RowNum - 1 Do

Begin

ResultStr := '';

Grid.Cells[0, I] := Chr(Ord('A') + I) + ':';

Temp := AdjacencyArr[I];

While Temp.Next <> Nil Do

Begin

If ResultStr <> '' Then

ResultStr := ResultStr + ', ';

ResultStr := ResultStr + Temp.Next.Value;

Temp := Temp.Next;

End;

Grid.Cells[1, I] := ResultStr;

End;

End;

Procedure TMainForm.ResultButtonClick(Sender: TObject);

Var

Matrix: TMatrix;

I, J: Integer;

Begin

SetLength(Matrix, MatrixGrid.RowCount - 1, MatrixGrid.ColCount - 1);

For I := 1 To MatrixGrid.RowCount - 1 Do

For J := 1 To MatrixGrid.ColCount - 1 Do

Matrix[I - 1][J - 1] := StrToInt(MatrixGrid.Cells[J, I]);

CreateAdjacencyLists(Matrix);

MakeResultGrid(MatrixGrid.RowCount - 1, ResultGrid);

FillResultGrid(MatrixGrid.RowCount - 1, ResultGrid);

ResultLabel.Visible := True;

IsSaved := False;

SaveFileButton.Enabled := True;

End;

Procedure TMainForm.ResultEditChange(Sender: TObject);

Begin

If ResultGrid.Visible = False Then

Begin

IsSaved := True;

SaveFileButton.Enabled := False;

End

Else

Begin

IsSaved := False;

SaveFileButton.Enabled := True;

End;

End;

Procedure TMainForm.ResultGridDrawCell(Sender: TObject; ACol, ARow: Integer; Rect: TRect; State: TGridDrawState);

Begin

If ACol = 1 Then

Begin

Text := ResultGrid.Cells[ACol, ARow];

ResultGrid.Canvas.Brush.Color := ClWindow;

ResultGrid.Canvas.FillRect(Rect);

DrawText(ResultGrid.Canvas.Handle, PChar(Text), Length(Text), Rect,

DT\_CENTER Or DT\_VCENTER Or DT\_SINGLELINE);

End;

End;

Procedure TMainForm.OpenOnClick(Sender: TObject);

Var

Error: ERRORS\_CODE;

F: TextFile;

FileName: String;

Begin

If OpenFile.Execute Then

Begin

FileName := OpenFile.FileName;

AssignFile(F, FileName);

Error := IsReadable(F);

If Error = CORRECT Then

Error := ReadFileData(F, Sender);

If Error <> CORRECT Then

Application.MessageBox(PWideChar(ERRORS[Error]), 'Ошибка', MB\_OK Or MB\_ICONINFORMATION);

End;

End;

Function IsWriteable(Var F: TextFile): ERRORS\_CODE;

Var

Error: ERRORS\_CODE;

Begin

Error := CORRECT;

Try

Try

Append(F);

Finally

CloseFile(F);

End;

Except

Error := Is\_NOT\_WRITEABLE;

End;

IsWriteable := Error;

End;

Procedure TMainForm.SaveOnClick(Sender: TObject);

Var

Error: ERRORS\_CODE;

F: TextFile;

FileName: String;

I: Integer;

Begin

If SaveFile.Execute Then

Begin

FileName := SaveFile.FileName;

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

AssignFile(F, FileName);

If FileExists(FileName) Then

Begin

Error := IsWriteable(F);

If Error = CORRECT Then

Begin

Append(F);

Writeln(F, #13#10, 'Списки инцидентности: ');

For I := 0 To Length(AdjacencyArr) - 1 Do

Writeln(F, Chr(Ord('A') + I) + ': ' +

ResultGrid.Cells[1, I]);

CloseFile(F);

IsSaved := True;

End

Else

Begin

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

MB\_OK Or MB\_ICONINFORMATION);

IsSaved := False;

End;

End

Else

Begin

Rewrite(F);

Writeln(F, 'Списки инцидентности: ');

For I := 0 To Length(AdjacencyArr) - 1 Do

Writeln(F, Chr(Ord('A') + I) + ': ' + ResultGrid.Cells[1, I]);

CloseFile(F);

IsSaved := True;

End;

End;

End;

End.

Unit InstructionUnit;

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)

CloseButton: TButton;

Button1: TButton;

InstructionLabel1: TLabel;

InstructionLabel4: TLabel;

InstructionLabel2: TLabel;

Procedure CloseButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

InstructionForm: TInstructionForm;

Implementation

{$R \*.dfm}

Procedure TInstructionForm.CloseButtonClick(Sender: TObject);

Begin

Close;

End;

End.

Unit DeveloperUnit;

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)

CloseButton: TButton;

DeveloperLabel1: TLabel;

DeveloperLabel2: TLabel;

Procedure CloseButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

DeveloperForm: TDeveloperForm;

Implementation

{$R \*.dfm}

Procedure TDeveloperForm.CloseButtonClick(Sender: TObject);

Begin

Close;

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;

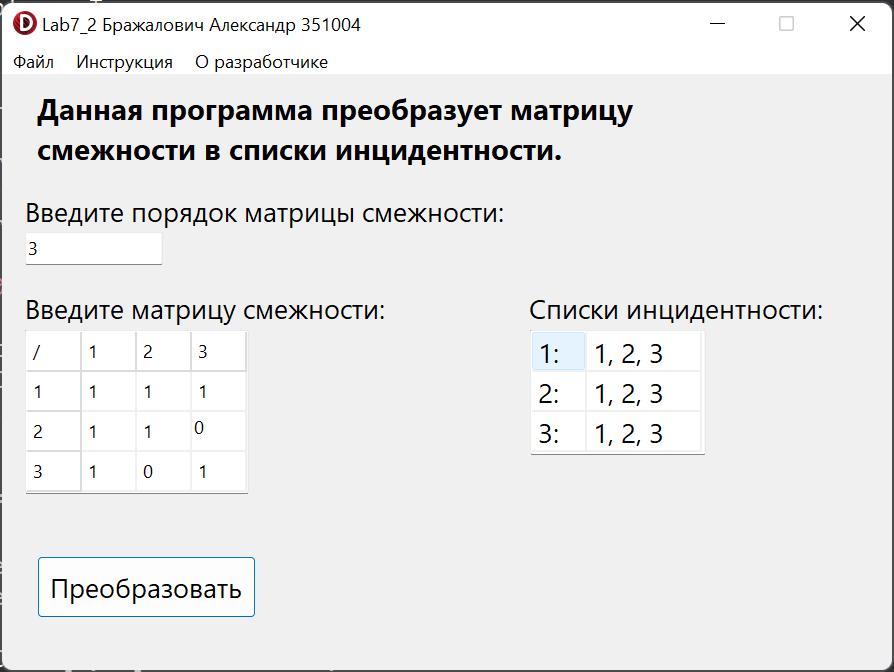
}

}

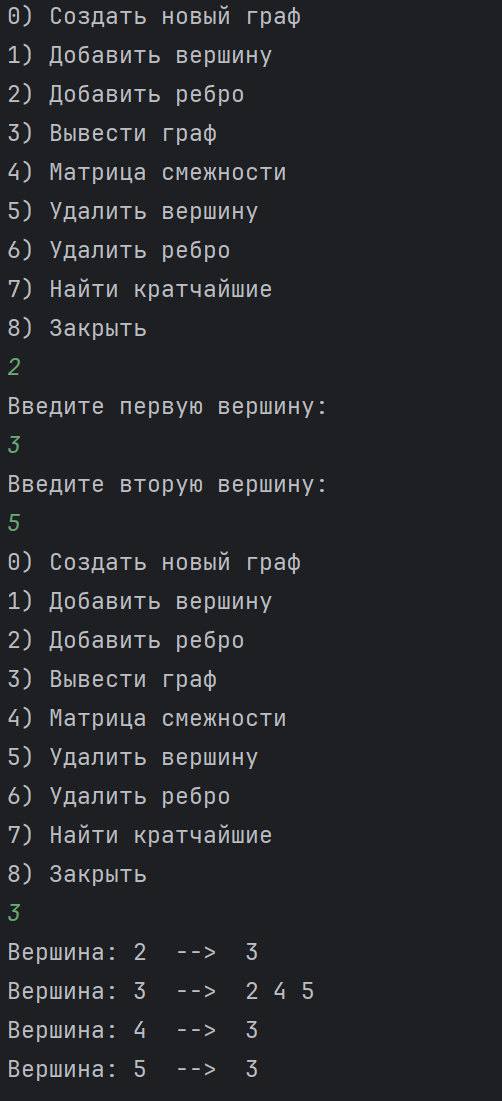
}

**Скриншоты:**

**Delphi:**

****

**Java:**

****

**Блок-схема:**

