Минобрнауки России

ФГБОУ ВПО «НИУ МЭИ» АВТИ

Кафедра математического и компьютерного моделирования

**Лабораторная работа №8**

**Технологии программирования**

**"Разработка программы для работы с графом на языке Java"**

**Работу выполнил:**

Солонин Егор А-14-19

Вариант 14

**Работу принял:**

Князев А. В.

Москва 2021

**Задание:**

Разработать класс для представления неориентированного графа и поиска всех циклов или путей между двумя заданными вершинами.

Должен использоваться метод поиска “в глубину”.

Класс должен включать объект класса "Стек" для представления стека.

Необходимо разработать класс "Стек" для представления стека.

Должен использоваться не рекурсивный алгоритм поиска путей или циклов.

Разработать программу на языке Java, реализующую конкретное задание.

Должны использоваться элементы библиотеки Swing.

Программа должна обеспечивать ввод описания графа с клавиатуры (с помощью мыши).

Программа должна обеспечивать представление исходного графа и результатов в графическом виде.

В программе должны использоваться меню и диалоговые окна.

Программа должна сначала найти и запомнить все пути или циклы, а затем показывать их по запросу.

В описаниях заданий используются следующие обозначения:

- стек\_спис – стек на основе связанного списка;

- граф\_матр – для описания графа используется матрица смежности;

|  |  |  |
| --- | --- | --- |
| 14 | Солонин Е.В. | Найти все пути между двумя вершинами, проходящие хотя бы через одну периферийную вершину (стек\_спис, граф\_матр). |

**Описание работы программы:**

При открытии приложения пользователь имеет возможность:

1. Задать вершины графа щелчком мыши
2. Задать ребра графа с помощью перетягивания мыши с одной вершины на другую
3. Выбрать стартовую вершину (опция меню)
4. Выбрать конечную вершину (опция меню)
5. Найти все пути в графе из одной вершины в другую, проходящие хотя бы через одну периферийную (опция меню – результат в диалоговом окне)
6. Отобразить все периферийные вершины ярко-розовым цветом (опция меню)
7. Отобразить альтернативный путь (опция меню)

**Алгоритмы операций на псевдокоде:**

Абстрактный класс Graph имеет:

* Stack<char> currentPath\_stack – стек вершин
* protected final int N – максимальное число вершин в графе
* Конструктор
* string[] FindAllPaths(char start, char finish, int[,] AdjacencyMatrix) – метод поиска всех путей в графе
  + start – начальная вершина
  + finish – конечная вершина
  + AdjacencyMatrix – матрица смежности
* string GetPath(Stack<char> stack, char finish) – построение пути по стеку вершин
  + finish – конечная вершина
  + stack – стек вершин
* abstract char FindConnectable(char curr, bool[] visits, int startSearch) – абстрактный метод для поиска связной еще не посещённой вершины
  + curr – текущая вершина
  + visits – массив посещений вершин
  + startSearch – индекс стартовой вершина для обхода
* abstract string[] FindValidPaths(string[] allPaths, char[] PerVertexes) – абстрактный метод для поиска всех путей удовлетворяющих некоторому условию
  + allPaths – все пути в графе
  + PerVertexes – периферийные вершины
* abstract void FromString(string data) - абстрактный метод для заполнения матрицы смежности
  + data – вершины и ребра в одной строке

Дочерний класс GraphMatrix extends Graph имеет:

* public int[,] AdjacencyMatrix – матрица смежности
* char[] perVertexes – периферийные вершины
* Конструктор
* void Clear() – очистить матрицу смежности
* private int[,] GetDistMatrix() – метод, позволяющий получить матрицу расстояний
* private int[] GetEccentricity() – метод, позволяющий получить массив эксцентриситетов графа
* private char[] GetPeriphericVertexes() – метод, позволяющий получить массив периферийных вершин
* void FromString() – перегрузка метода из родительского класса
* string[] FindAllPaths() - перегрузка метода из родительского класса
* char FindConnectable() - перегрузка метода из родительского класса
* string[] FindValidPaths() - перегрузка метода из родительского класса

**Алгоритмы**:

**FindAllPaths()** - метод поиска всех путей в графе:

1. Stack<string> *allPaths* – заводим стек куда будем добавлять все найденный пути
2. bool[] *visits* – заводим массив посещений вершин и отмечаем все вершины как не посещенные
3. Добавляем в стек *currentPath\_stack* точку старта пути *start*
4. Отмечаем вершину start посещенной
5. *startSearch* = 0 - индекс начала поиска вершины
6. Пока *currentPath\_stack* не пуст
   1. Получаем текущую вершину *curr* с вершины стека *currentPath\_stack*
   2. Получаем следующую не посещённую смежную вершину next с помощью метода *FindConnectable*
   3. Если *next* - вершина конца пути, добавляем путь в стек *allPaths*
   4. Иначе если следующая вершина найдена
      1. Добавляем в стек *next*
      2. Отмечаем *next* посещенной
      3. Сбрасываем индекс начала поиска вершины *startSearch*
   5. Если следующая вершина не найдена или вершина совпадает с концом пути
      1. Увеличиваем индекс начала поиска вершины *startSearch*
      2. Отмечаем *curr* посещенной
      3. Удаляем из *currentPath\_stack* вершину
7. Конец цикла, возврат *allPaths* в виде массива

**FindConnectable()** - поиск связной еще не посещённой вершины

1. *curr* – текущая вершина
2. *visits* – массив посещений вершин
3. *startSearch* – индекс стартовой вершина для обхода
4. Цикл по *i* со стартовой вершины до N
   1. Если нет ребра между вершиной *curr* и *i*, переходим к следующей вершине
   2. Если вершина *i* посещенная, переходим к следующей вершине
   3. Иначе возвращаем вершину
5. Возвращаем ‘!’ – не нашли подходящих вершин

**GetDistMatrix()** - метод, позволяющий получить матрицу расстояний

1. Заводим матрицу расстояний *DistMatrix*, копируя ее из матрицы смежности
2. Заменяем все нулевые недиагональные элементы на числа, которые точно превосходят максимальное расстояние в графе, т.е. на N+1(максимальное число вершин в графе)
3. По алгоритму Флойда–Уоршелла заполняем матрицу расстояний
4. Возврат *DistMatrix*

**GetEccentricity()** – метод, позволяющий получить массив эксцентриситетов графа

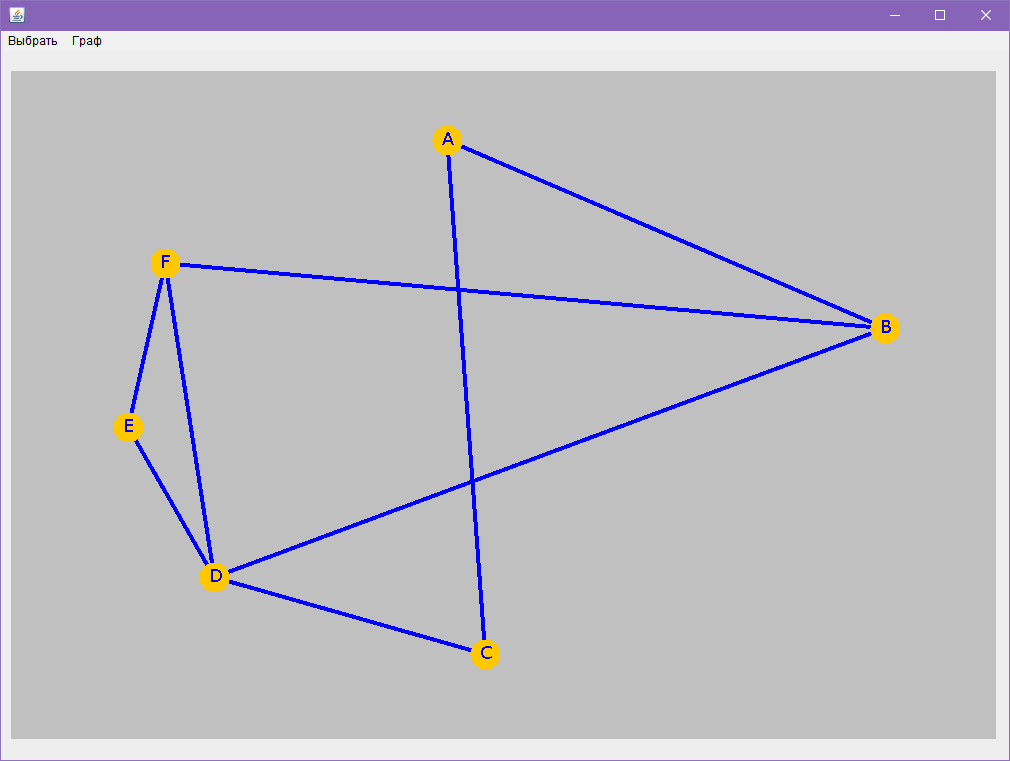
1. Заводим массив эксцентриситетов
2. Находим максимальный элемент в каждой строке матрицы расстояний и добавляем его в массив эксцентриситетов

**GetPeriphericVertexes() –** метод, позволяющий получить массив периферийных вершин

1. Заводим массив периферийных вершин
2. Добавляем туда все вершины с максимальным эксцентриситетом

**FindValidPaths()** - поиска всех путей, которые содержат хотя бы одну периферийную вершину

1. *allPaths* – массив всех путей
2. *PerVertexes* – массив периферийных вершин
3. Заводим стек *PathsWithCondition* – пути, содержащие периферийную вершину
4. Для каждого *path* в *allPaths*
   1. Для каждой *perVertex* в *PerVertexes*
      1. Если *path* содержит *perVertex*
         1. Добавляем путь в *PathsWithCondition*
         2. Переходим к сдедующему *path*
5. Возврат *PathsWithCondition* в виде массива

**Тесты работы программы:**

Возьмем граф, изображенный на рисунке.

Требуется найти все пути из вершины D в вершину F, проходящие хотя бы через одну из периферийных вершин.

Периферийные вершины: A, E

Будут найдены пути:

D-C-A-B-F

D-E-F

|  |  |  |
| --- | --- | --- |
| Входное условие | Правильные классы эквивалентности | Неправильные классы эквивалентности |
| Число периферийных вершин в пути | >0 *(1)* | Ни одной *(2)* |

|  |  |  |
| --- | --- | --- |
| X1 | X2 | F |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

X1 – путь проходит через вершину А

X2 – путь проходит через вершину Е

Таким образом, здесь должен быть один тест, где истинно хотя бы одно из условий, и один тест, в котором ложны оба условия.

1. Путь проходит через А или E, и программа должна находить пути (причём их должно быть как минимум два), удовлетворяющие этим условиям

D-C-A-B-F

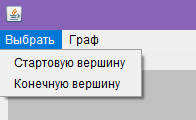
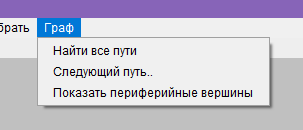
D-E-F

1. Существует путь, для которых одновременно не выполняется условие X1 и X2, и программа отвергает этот путь.

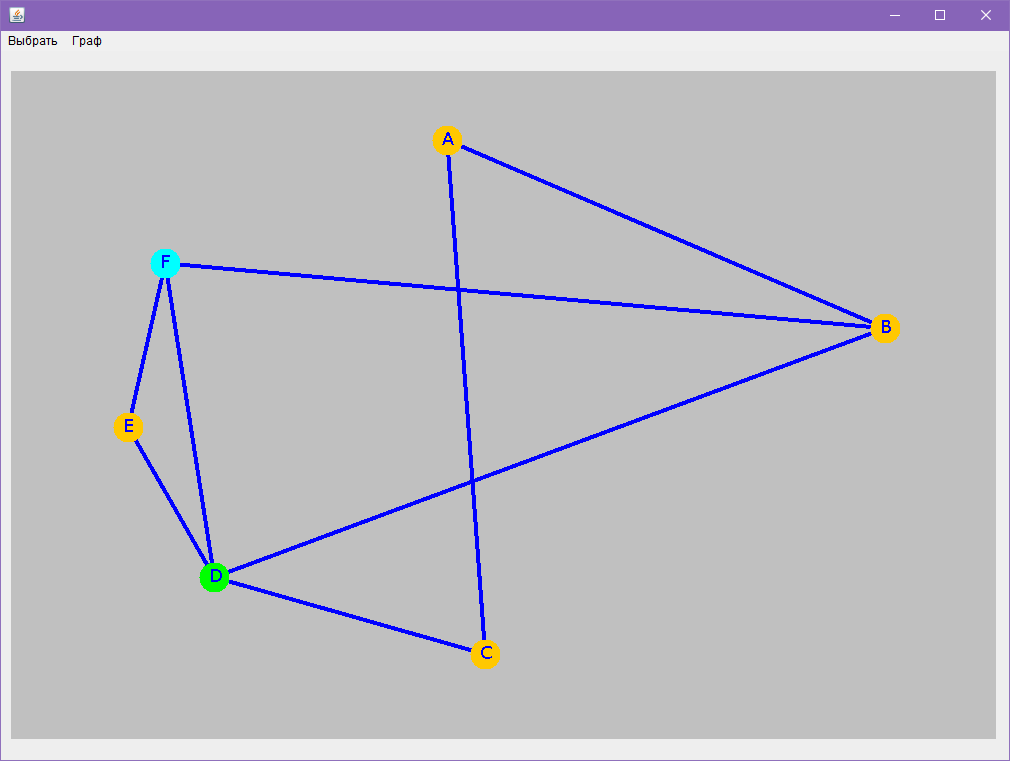
Например путь: D-F или D-B-F

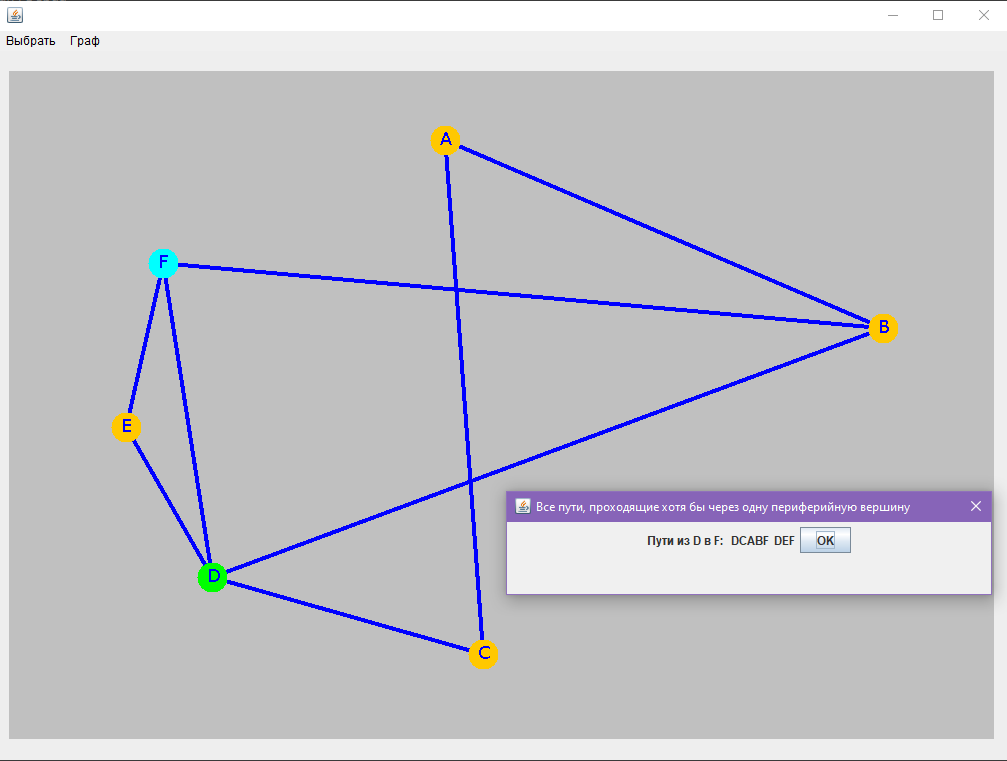
*Таким образом, данный граф с указанным заданием является полным тестом.*

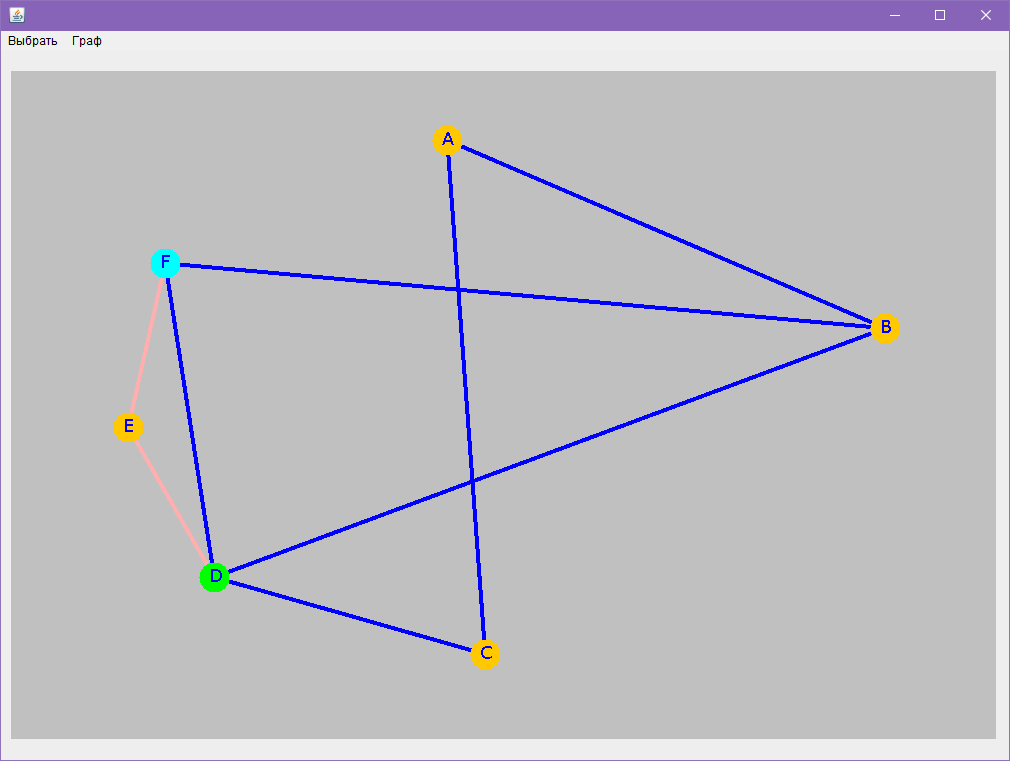
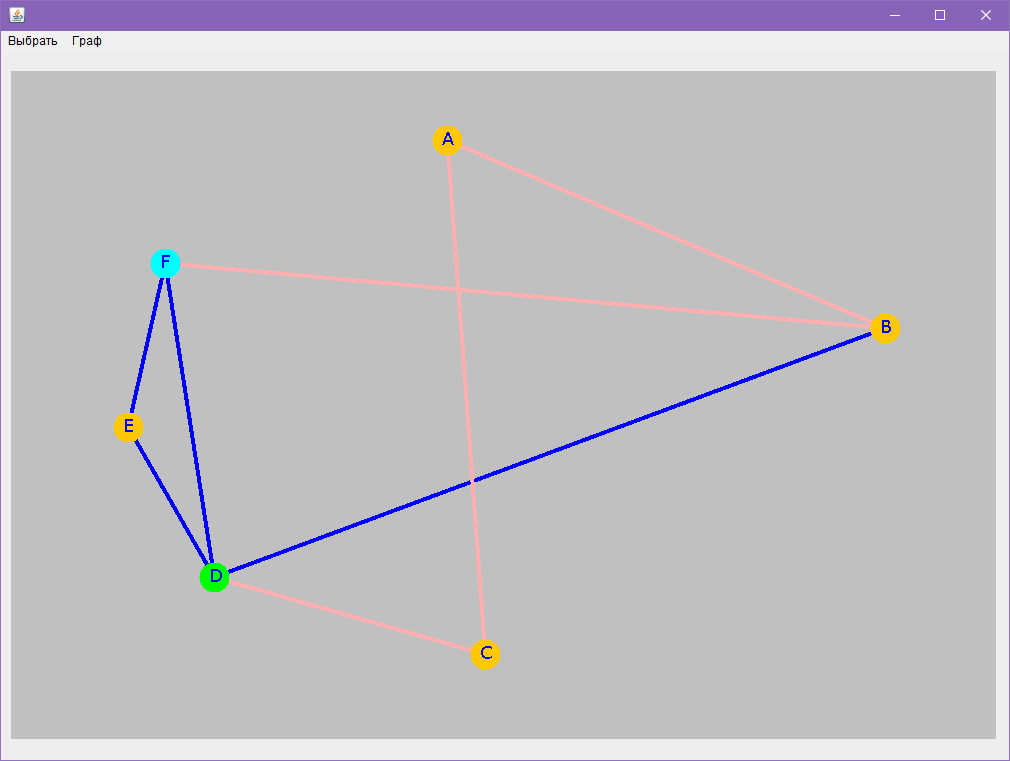
**Распечатки экранов:**

**

***Зеленым цветом отмечена стартовая вершина, голубым - конечная***



****

** ***Телесным цветом показаны пути***

**Листинг программы:**

**Файл App.java**

public class App {

public static void main(String[] args) throws Exception {

new Visualization();

}

}

**Файл Vertex.java**

import java.awt.Color;

public class Vertex {

int x;

int y;

char title;

Color c;

Vertex(int x, int y, char title, Color c) {

this.x = x;

this.y = y;

this.title = title;

this.c = c;

}

Vertex(int x, int y) {

this.x = x;

this.y = y;

}

}

**Файл Edge.java**

import java.awt.Color;

public class Edge {

Vertex start;

Vertex end;

String title;

Color c;

Edge(Vertex start, Vertex end, Color c) {

this.start = start;

this.end = end;

this.title = "" + start.title + end.title;

this.c = c;

}

public boolean eq(Vertex A, Vertex B) {

if (this.start.x == A.x && this.start.y == A.y && this.end.x == B.x && this.end.y == B.y)

return true;

if (this.end.x == A.x && this.end.y == A.y && this.start.x == B.x && this.start.y == B.y)

return true;

return false;

}

public boolean eq(char A, char B) {

if (this.title.charAt(0) == A && this.title.charAt(1) == B)

return true;

if (this.title.charAt(1) == A && this.title.charAt(0) == B)

return true;

return false;

}

}

**Файл Node.java**

public class Node<T> {

T value;

Node<T> next;

public Node(T value) {

this.value = value;

}

}

**Файл Stack.java**

import java.util.Arrays;

public class Stack<T> {

Node<T> head;

int count;

private IllegalArgumentException err411 = new IllegalArgumentException("Error 411: Stack is empty");

public boolean IsEmpty() {

return count == 0;

}

public int Count() {

return count;

}

public void Push(T item) {

Node<T> node = new Node<T>(item);

node.next = head;

head = node;

count++;

}

public T Pop() throws IllegalArgumentException {

if (IsEmpty())

throw err411;

Node<T> temp = head;

head = head.next;

count--;

return temp.value;

}

public Stack<T> Reverse() {

Stack<T> copy = new Stack<T>();

Node<T> node = head;

for (; node != null; node = node.next)

copy.Push(node.value);

return copy;

}

public String[] ToArray() {

@SuppressWarnings("unchecked")

T[] arr = (T[]) new Object[count];

int i = 0;

Node<T> node = head;

for (; node != null; node = node.next, i++)

arr[i] = node.value;

String[] stringArray = Arrays.copyOf(arr, arr.length, String[].class);

return stringArray;

}

public T Peek() throws IllegalArgumentException {

if (IsEmpty())

throw err411;

return head.value;

}

}

**Файл Graph.java**

public abstract class Graph {

Stack<Character> currentPath\_stack = new Stack<Character>();

protected final int N = 26;

protected final int ASCII\_SHIFT = 65;

public Graph(String verts, String edges) {

FromString(verts + ';' + edges);

}

public String[] FindAllPaths(char start, char finish, int[][] AdjacencyMatrix) {

Stack<String> allPaths = new Stack<String>();

boolean[] visits = new boolean[N];

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

visits[i] = false;

currentPath\_stack.Push(start);

visits[start - ASCII\_SHIFT] = true;

int startSearch = 0;

while (!currentPath\_stack.IsEmpty()) {

int curr = currentPath\_stack.Peek() - ASCII\_SHIFT;

char next = FindConnectable(curr, visits, startSearch);

if (next == finish) {

String path = GetPath(currentPath\_stack, finish);

allPaths.Push(path);

} else if (next != '!') {

currentPath\_stack.Push(next);

visits[next - ASCII\_SHIFT] = true;

startSearch = 0;

}

if (next == '!' || next == finish) {

startSearch = curr + 1;

visits[curr] = false;

currentPath\_stack.Pop();

}

}

return allPaths.ToArray();

}

private String GetPath(Stack<Character> stack, char finish) {

String path = "";

Stack<Character> temp = stack.Reverse();

while (!temp.IsEmpty())

path += temp.Pop();

path += finish;

return path;

}

public abstract char FindConnectable(int curr, boolean[] visits, int startSearch);

public abstract String[] FindValidPaths(String[] allPaths, char[] AppropriateVertexes);

public abstract void FromString(String data);

}

**Файл GraphMatrix.java**

class GraphMatrix extends Graph

{

public int[][] AdjacencyMatrix;

char[] perVertexes;

public GraphMatrix(String verts, String edges) {

super(verts, edges);

perVertexes = GetPeriphericVertexes();

}

public char[] getPerVertexes()

{

return perVertexes;

}

public void Clear()

{

AdjacencyMatrix = new int[N][N];

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

for (int j = 0; j < N; j++)

AdjacencyMatrix[i][j] = -1;

}

public void FromString(String data)

{

Clear();

String[] temp = data.split(";");

String vertexesData = temp[0];

String edgesData = temp[1];

// char[] separators = new char[] { ' ', '\r', '\n', ',' };

String[] vertexes = vertexesData.split(" ");

String[] edges = edgesData.split(" ");

for (int i = 0; i < vertexes.length; i++)

for (int j = 0; j < vertexes.length; j++)

{

int row = vertexes[i].charAt(0) - ASCII\_SHIFT;

int col = vertexes[j].charAt(0) - ASCII\_SHIFT;

AdjacencyMatrix[row][col] = 0;

AdjacencyMatrix[col][row] = 0;

}

for (String edge : edges)

{

int start = edge.charAt(0) - ASCII\_SHIFT;

int finish = edge.charAt(1) - ASCII\_SHIFT;

AdjacencyMatrix[start][finish] = 1;

AdjacencyMatrix[finish][start] = 1;

}

}

public String[] FindAllPaths(char start, char finish, int[][] adj)

{

String[] allPaths = super.FindAllPaths(start, finish, AdjacencyMatrix);

return FindValidPaths(allPaths, perVertexes);

//return allPaths;

}

public String[] FindAllPaths(char start, char finish)

{

return FindAllPaths(start, finish, null);

}

private int[][] GetDistMatrix()

{

int[][] DistMatrix = AdjacencyMatrix.clone();

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

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

if (DistMatrix[k][i] == 0 && i != k)

DistMatrix[k][i] = N + 1;

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

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

for (int j = 0; j < N; ++j)

if (DistMatrix[i][k] != -1 && DistMatrix[k][j] != -1 && DistMatrix[i][j] != -1)

DistMatrix[i][j] = Math.min(DistMatrix[i][j],

DistMatrix[i][k] + DistMatrix[k][j]);

return DistMatrix;

}

private int[] GetEccentricity()

{

int[][] DistMatrix = GetDistMatrix();

int[] ecc = new int[N];

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

{

int max = 0;

for (int j = 0; j < N; ++j)

if (DistMatrix[i][j] > max && DistMatrix[i][j] <= N)

max = DistMatrix[i][j];

ecc[i] = max;

}

return ecc;

}

private char[] GetPeriphericVertexes()

{

int[] ecc = GetEccentricity();

int max = 0;

int PerVertexesCount = 1;

for (int j = 0; j < N; ++j)

{

if (ecc[j] > max)

{

max = ecc[j];

PerVertexesCount = 1;

}

else if (ecc[j] == max)

PerVertexesCount++;

}

char[] PerVertexes = new char[PerVertexesCount];

int index = 0;

for (int j = 0; j < N; ++j)

if (ecc[j] == max)

{

PerVertexes[index] = (char)(j + ASCII\_SHIFT);

index++;

}

return PerVertexes;

}

public char FindConnectable(int curr, boolean[] visits, int startSearch)

{

for (int i = startSearch; i < N; i++)

{

if (AdjacencyMatrix[curr][i] != 1) continue;

if (visits[i]) continue;

return (char)(i + ASCII\_SHIFT);

}

return '!';

}

public String[] FindValidPaths(String[] allPaths, char[] PerVertexes)

{

Stack<String> PathsWithCondition = new Stack<String>();

for (String path : allPaths) {

for (char perVertex : PerVertexes)

if (path.indexOf(perVertex)!= -1)

{

PathsWithCondition.Push(path);

break;

}

}

return PathsWithCondition.ToArray();

}

}

**Файл Visualization.java**

import javax.swing.\*;

import java.awt.\*;

import java.awt.Color;

import java.awt.event.\*;

import java.awt.geom.Line2D;

public class Visualization extends JFrame implements MouseListener, MouseMotionListener {

JPanel panel;

final int ASCII\_SHIFT = 65;

final int radius = 15;

Stack<Vertex> verts;

Stack<Edge> edges;

Vertex start, end;

boolean pickStart, pickEnd;

Vertex pathsStart, pathsEnd;

String[] Paths;

GraphMatrix graph;

int currPathIndex = 0;

Visualization() {

verts = new Stack<Vertex>();

edges = new Stack<Edge>();

start = null;

end = null;

pathsStart = null;

pathsEnd = null;

pickStart = false;

pickEnd = false;

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setSize(1024, 768);

this.setLayout(null);

initPanel();

initMenu();

this.setVisible(true);

}

void initPanel() {

panel = new JPanel();

panel.setBounds(10, 20, 985, 668);

panel.setBackground(Color.LIGHT\_GRAY);

panel.setOpaque(true);

panel.addMouseListener(this);

panel.addMouseMotionListener(this);

this.add(panel);

}

void initMenu() {

MenuBar mb = new MenuBar();

setMenuBar(mb);

Menu file = new Menu("Выбрать");

MenuItem menuStart = new MenuItem("Стартовую вершину");

file.add(menuStart);

MenuItem menuEnd = new MenuItem("Конечную вершину");

file.add(menuEnd);

mb.add(file);

Menu paths = new Menu("Граф");

MenuItem menuPaths = new MenuItem("Найти все пути");

paths.add(menuPaths);

MenuItem menuNext = new MenuItem("Следующий путь..");

paths.add(menuNext);

MenuItem menuPerVerts = new MenuItem("Показать периферийные вершины");

paths.add(menuPerVerts);

mb.add(paths);

menuStart.addActionListener(new menuStartHandler());

menuEnd.addActionListener(new menuEndHandler());

menuPaths.addActionListener(new menuPathsHandler(this));

menuNext.addActionListener(new menuNextHandler());

menuPerVerts.addActionListener(new menuPerVertsHandler());

}

class menuStartHandler implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

pickStart = !pickStart;

}

}

class menuEndHandler implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

pickEnd = !pickEnd;

}

}

class menuPerVertsHandler implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

char[] perVerts = graph.perVertexes;

for (char perVert : perVerts)

FindVertex(perVert).c = Color.MAGENTA;

repaint();

}

}

class menuPathsHandler implements ActionListener {

JFrame jf;

menuPathsHandler(JFrame jf) {

this.jf = jf;

}

@Override

public void actionPerformed(ActionEvent e) {

graph = new GraphMatrix(vertsToString(), edgesToString());

if (pathsStart != null && pathsEnd != null)

Paths = graph.FindAllPaths(pathsStart.title, pathsEnd.title);

ParamDialog d = new ParamDialog(jf, "Все пути, проходящие хотя бы через одну периферийную вершину");

d.setVisible(true);

for (String path : Paths)

System.out.println(path);

repaintEdges(Paths[0]);

repaint();

}

}

class menuNextHandler implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

currPathIndex++;

if (currPathIndex < Paths.length)

repaintEdges(Paths[currPathIndex]);

repaint();

}

}

class ParamDialog extends Dialog {

String lbl1 = "Путей с заданным условием не найдено";

String lbl2 = "Пути из " + pathsStart.title + " в " + pathsEnd.title + ": ";

JLabel text;

ParamDialog(JFrame parent, String title) {

super(parent, title, true);

addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent we) {

dispose();

setVisible(false);

}

});

setLayout(new FlowLayout());

setSize(500, 300);

JButton btOk = new JButton("OK");

if (Paths.length == 0) {

text = new JLabel(lbl1);

add(text);

} else {

text = new JLabel(lbl2);

add(text);

for (String path : Paths) {

text = new JLabel(path);

add(text);

}

}

add(btOk);

btOk.addActionListener(new OkListener(this));

}

class OkListener implements ActionListener {

ParamDialog pd;

OkListener(ParamDialog pard) {

pd = pard;

}

public void actionPerformed(ActionEvent ae) {

pd.dispose();

pd.setVisible(false);

}

}

}

public String vertsToString() {

Stack<Vertex> vertsCopy = verts.Reverse();

String res = "";

while (!vertsCopy.IsEmpty()) {

Vertex curr = vertsCopy.Pop();

res += curr.title + " ";

}

return res;

}

public String edgesToString() {

Stack<Edge> edgesCopy = edges.Reverse();

String res = "";

Edge curr = edgesCopy.Pop();

res += "" + curr.start.title + "" + curr.end.title;

while (!edgesCopy.IsEmpty()) {

curr = edgesCopy.Pop();

res += " " + curr.start.title + "" + curr.end.title;

}

return res;

}

Vertex FindVertex(int x, int y) {

Stack<Vertex> vertsCopy = verts.Reverse();

while (!vertsCopy.IsEmpty()) {

Vertex curr = vertsCopy.Pop();

if (curr.x - radius < x && x < curr.x + radius && curr.y - radius < y && y < curr.y + radius)

return curr;

}

return null;

}

Vertex FindVertex(char title) {

Stack<Vertex> vertsCopy = verts.Reverse();

while (!vertsCopy.IsEmpty()) {

Vertex curr = vertsCopy.Pop();

if (curr.title == title)

return curr;

}

return null;

}

Edge FindEdge(Vertex start, Vertex end) {

Stack<Edge> edgesCopy = edges.Reverse();

while (!edgesCopy.IsEmpty()) {

Edge curr = edgesCopy.Pop();

if (curr.eq(start, end))

return curr;

}

return null;

}

Edge FindEdge(char start, char end) {

Stack<Edge> edgesCopy = edges.Reverse();

while (!edgesCopy.IsEmpty()) {

Edge curr = edgesCopy.Pop();

if (curr.eq(start, end))

return curr;

}

return null;

}

void repaintEdges(String path) {

Stack<Edge> edgesCopy = edges.Reverse();

while (!edgesCopy.IsEmpty())

edgesCopy.Pop().c = Color.BLUE;

System.out.println(path);

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

FindEdge(path.charAt(i), path.charAt(i + 1)).c = Color.PINK;

}

public void drawCircle(int x, int y, char title, Color clr) {

Graphics g = panel.getGraphics();

g.setColor(clr);

g.fillOval(x - radius, y - radius, 2 \* radius, 2 \* radius);

g.setColor(Color.BLUE);

g.setFont(new Font("Verdana", Font.PLAIN, 18));

g.drawString(title + "", x - radius / 3, y + radius / 3);

}

public void drawLine(int x1, int y1, int x2, int y2, Color clr) {

Graphics2D g2 = (Graphics2D) panel.getGraphics();

g2.setColor(clr);

g2.setStroke(new BasicStroke(4));

g2.draw(new Line2D.Float(x1, y1, x2, y2));

}

public void drawAllVertexes() {

Stack<Vertex> vertsCopy = verts.Reverse();

while (!vertsCopy.IsEmpty()) {

Vertex curr = vertsCopy.Pop();

drawCircle(curr.x, curr.y, curr.title, curr.c);

}

}

public void drawAllEdges() {

Stack<Edge> edgesCopy = edges.Reverse();

while (!edgesCopy.IsEmpty()) {

Edge curr = edgesCopy.Pop();

drawLine(curr.start.x, curr.start.y, curr.end.x, curr.end.y, curr.c);

}

}

public void paint(Graphics g) {

super.paint(g);

drawAllEdges();

if (start != null && end != null)

drawLine(start.x, start.y, end.x, end.y, Color.BLUE);

drawAllVertexes();

}

@Override

public void mouseClicked(MouseEvent e) {

int x = e.getX();

int y = e.getY();

char title = (char) (verts.Count() + ASCII\_SHIFT);

Vertex v = new Vertex(x, y, title, Color.orange);

Vertex foundVertex = FindVertex(x, y);

if (foundVertex == null) {

verts.Push(v);

drawCircle(x, y, title, Color.orange);

} else if (pickStart && pathsStart == null) {

pathsStart = foundVertex;

foundVertex.c = Color.GREEN;

pickStart = false;

repaint();

} else if (pickEnd && pathsEnd == null) {

pathsEnd = foundVertex;

foundVertex.c = Color.CYAN;

pickEnd = false;

repaint();

}

}

@Override

public void mousePressed(MouseEvent e) {

int x = e.getX();

int y = e.getY();

start = FindVertex(x, y);

}

@Override

public void mouseReleased(MouseEvent e) {

int x = e.getX();

int y = e.getY();

end = FindVertex(x, y);

if (start != null && end != null)

if (start.x != end.x && start.y != end.y) {

Edge foundEdge = FindEdge(start, end);

if (foundEdge == null) {

Edge ed = new Edge(start, end, Color.BLUE);

edges.Push(ed);

}

}

repaint();

start = null;

end = null;

}

@Override

public void mouseEntered(MouseEvent e) { }

@Override

public void mouseExited(MouseEvent e) { }

@Override

public void mouseDragged(MouseEvent e) {

int x = e.getX();

int y = e.getY();

end = new Vertex(x, y);

if (start != null)

repaint();

}

@Override

public void mouseMoved(MouseEvent e) {

}

}