МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

ТЕМА: «Алгоритм Флойда-Уоршелла»

Студентка гр. 8303 _____ Потураева М.Ю.

Студентка гр. 8303 _____ Колосова М.П.

Студент гр. 8382 _____ Гордиенко А.М.

Руководитель _____ Ефремов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студентка Потураева М.Ю. группы 8303

Студентка Колосова М.П. группы 8303

Студент Гордиенко А.М. группы 8382

Тема практики: «Алгоритм Флойда-Уоршелла»

Задание на практику:

Командная разработка визуализации алгоритма на языке Java с графическим интерфейсом.

Алгоритм: Алгоритм Флойда-Уоршелла.

Дата сдачи отчёта: 00.07.2020

Дата защиты отчёта: 00.07.2020

Студентка гр. 8303	_____	Потураева М.Ю.
Студентка гр. 8303	_____	Колосова М.П.
Студент гр. 8382	_____	Гордиенко А.М.
Руководитель	_____	Ефремов М.А.

АННОТАЦИЯ

Целью учебной практики является разработка графического приложения для визуализации алгоритма Флойда-Уоршелла, нахождения кратчайших путей между всеми вершинами графа, получения матрицы достижимости. Программа пишется на языке Java.

SUMMARY

The purpose of the training practice is to develop a graphical application for visualizing the Floyd-Warshell algorithm, finding shortest paths between all the vertices of the graph, and obtaining the reachability matrix. The program is written in Java.

ОГЛАВЛЕНИЕ

Оглавление

1.ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные требования к программе

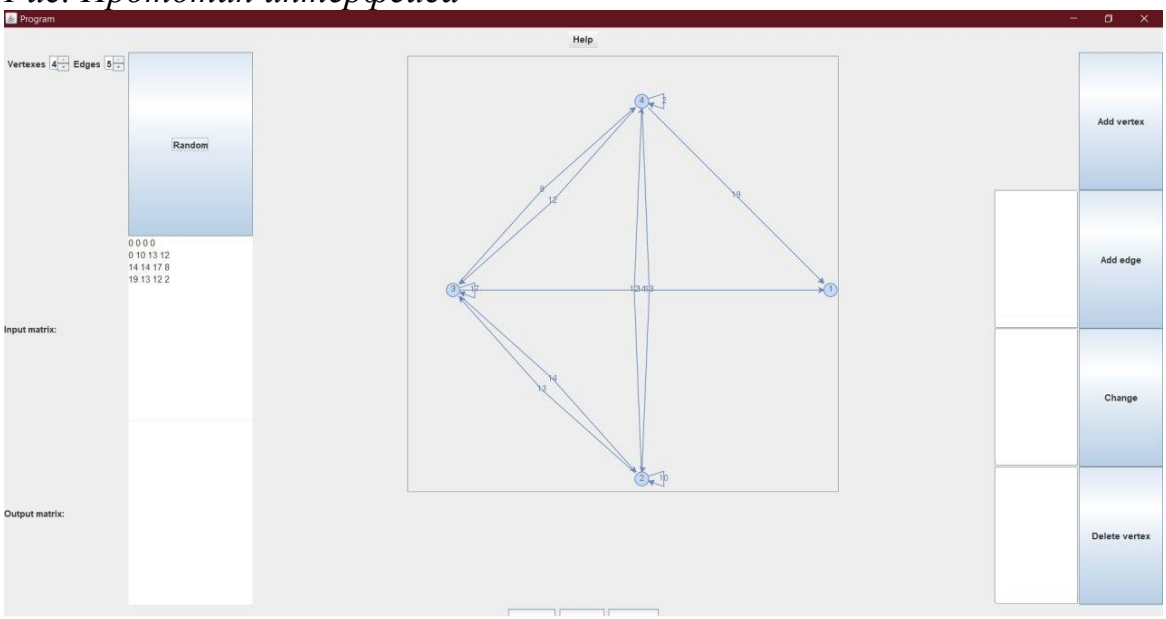
1.1.1. Требования к входным данным

На вход алгоритму подается строка в виде матрицы смежности вершин графа.

Пример входных данных: "2 3 1 5\n0 1 7 -3\n2 1 -4 3\n2 5 3 0"

1.1.2. Требования к визуализации

Рис. Прототип интерфейса



Программа имеет область для визуализации графа, справа и слева расположены кнопки для взаимодействия с ним.

1.1.3. Требования к алгоритму и данным

Алгоритм считывает введенный пользователем или случайно сгенерированный граф, который представлен в виде матрицы смежности.

Алгоритм получает необходимые данные и проделывает следующие действия:

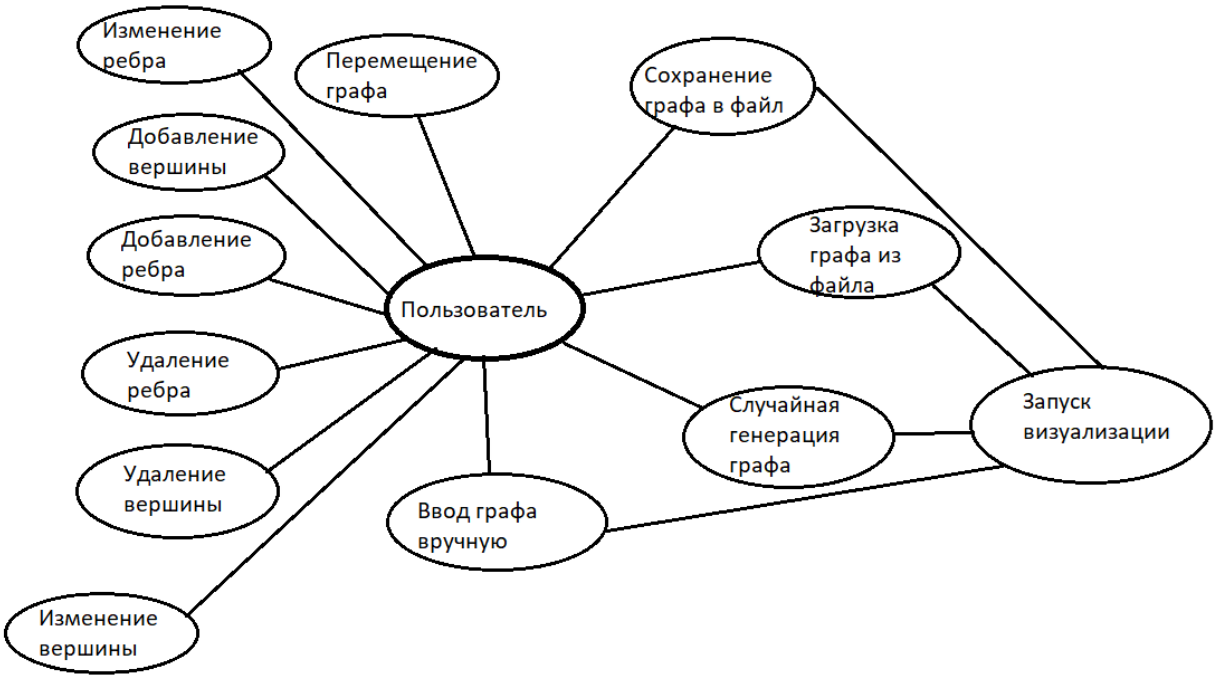
- Вычисляет количество вершин в графе
- Преобразует строку в матрицу

- Запускает визуализацию графа
- Запускает алгоритм Флойда-Уоршелла
- Выводит результат на экран

1.1.4. Требования к выходным данным

Выходными данными является матрица, полученная в результате применения алгоритма и визуализация графа.

Диаграмма сценариев использования:



2.РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ И ПЛАН РАБОТЫ

2.1. План разработки

- 2 июля-распределить роли в бригаде, определить функционал программы.
- 4 июля-создание интерфейса, но пока не рабочего, проектирование классов программы.
- 6 июля-случайная генерация графа, реализация алгоритма с отображением результата работы, создание плана тестирования.
- 8 июля-прототип визуализации, тестирование программы.
- 10 июля-готовая программа, завершение отчета.

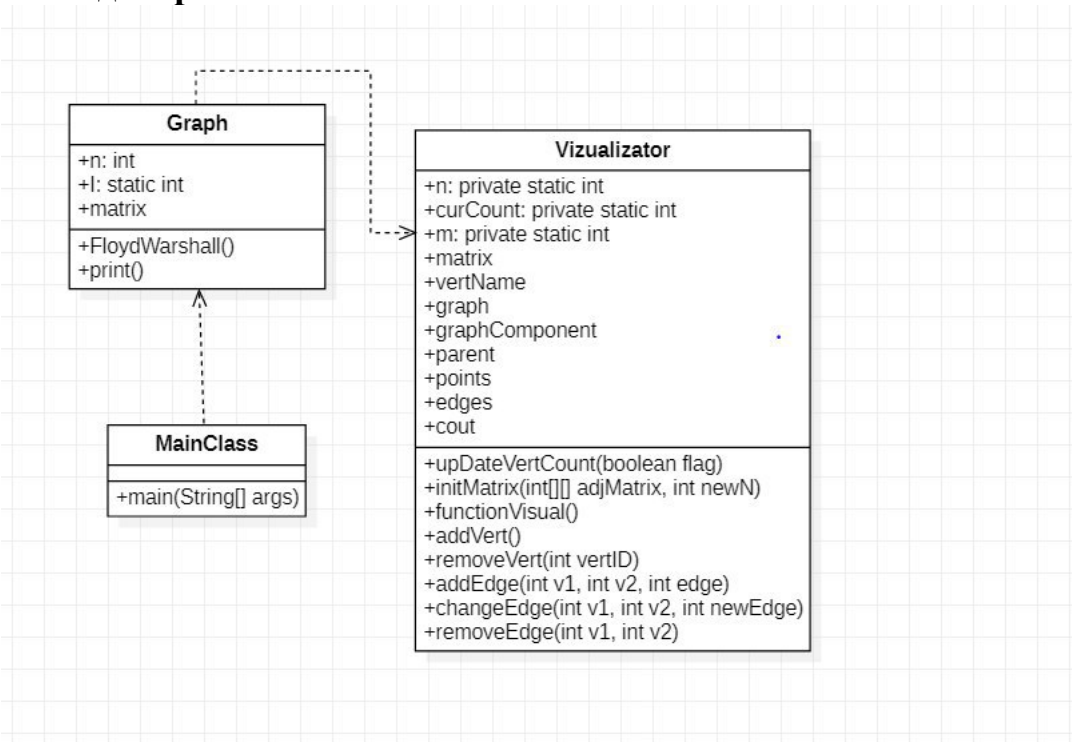
2.2. Распределение ролей в бригаде

- Потураева М.Ю-реализация юнит-тестирования, обработка исключений, составление технической документации.
- Колосова М.П.- разработчик визуализации, реализация отрисовки графа.
- Гордиенко А.М.- написание основного приложения, прописывание большинства взаимодействий с приложением.

3.ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Использование структур данных

UML диаграмма классов:



Для реализации проекта потребовалось разработать следующие структуры данных:

Класс Graph, который хранит информацию о графе, с которым будет работать алгоритм:

- Поля:
  - int n – количество вершин в графе
- Методы:
  - public Graph(String str)- инициализация графа
  - public void FloydWarshall() – алгоритм Флойда-Уоршелла
  - public String print()- вывод результата работы алгоритма
  - public void FWStep()- метод для пошаговой визуализации
  - public int[][] getMatrix()- получение матрицы смежности

- public void updateMatrix()-обновление матрицы
- public void deleteEdge(int v1, int v2)-удаление ребра
- public void changeEdge(int v1, int v2, int newEdge)-изменение ребра
- public void addEdge(int v1, int v2, int newEdge)-добавление ребра
- public void deleteVert(int v)-удаление вершины

Класс MainClass, который запускает алгоритм :

- Методы:
  - public static void main(String[] var0)- запуск алгоритма.

Ввод-вывод графов осуществляется с помощью класса GUI.

Класс Vizualizator, который наследуется от класса JPanel и имеет возможность размещаться на фрэйме:

- Поля:
  - int n – количество вершин в графе
  - int m — количество ребер в графе
  - int i, j , k — индексы матрицы достижимости на текущем шаге
  - int[][]matrix — заданная матрица
  - int[][] vertName — массив вершин с их статусами(-1 — удалена, 0 — не созданаб 1 — размещена на поле)
  - private mxGraph — класс графа, содержащий модель графа
  - private mxGraphComponent graphComponent- компонента графа, которая добавляется в JPanel
  - Object parent — объект-родитель класса графа
  - Object points[0] — массив объектов-вершин графа, содержащихся в моделе графа
  - private HashMap <Object, HashMap<Object, Object>> edges — объекты-ребра графа

- Методы:
  - public void initMatrix(int[][] adjMatrix, int newN) — инициализация матрицы, по которой будет строиться модель графа
  - private void upDateVertCount(boolean flag) — изменение размеров массивов при увеличении количества вершин
  - public void functionVisual() - функция построения и отображения модели графа
  - public void addVert() - функция добавления вершины в модель графа
  - public void removeVert(int vertID) — функция удаления вершины из модели графа
  - public void addEdge(int v1, int v2, int edge) — функция добавления ребра в модель графа
  - public void changeEdge(int v1, int v2, int newEdge) — функция изменения ребра в модели графа
  - public void removeEdge(int v1, int v2) — функция удаления ребра в модели графа
  - public void displayStepResult(int[][] matr) — функция для отображения процесса по шагам
  - public void displayResult(int[][] matr) — функция отображения конечного графа, построенного по матрице достижимости

Класс MainClass, который запускает алгоритм :

- Методы:
  - public static void main(String[] var0)- запуск алгоритма.

Взаимодействие с пользователем осуществляется с помощью класса GUI.

3.2.Описание работы алгоритма

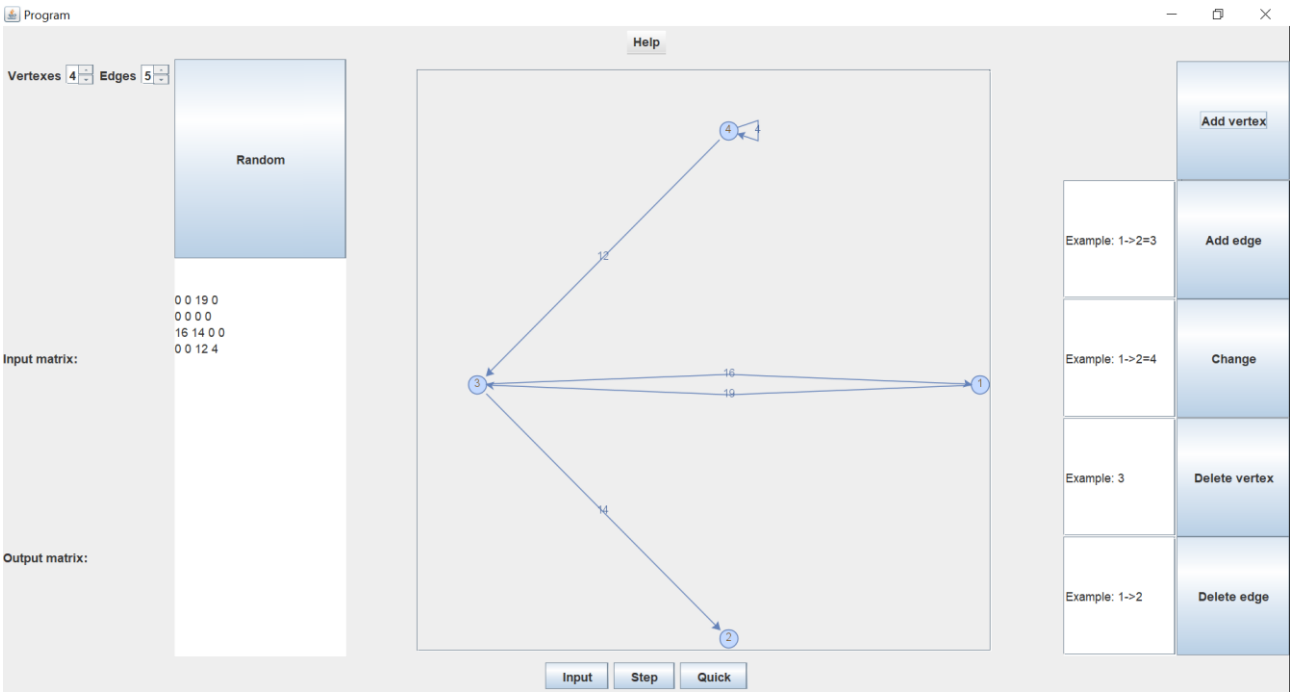
Программа реализовывает алгоритм Флойда-Уоршелла, который заключается в нахождении кратчайших путей в графе от любой вершины.

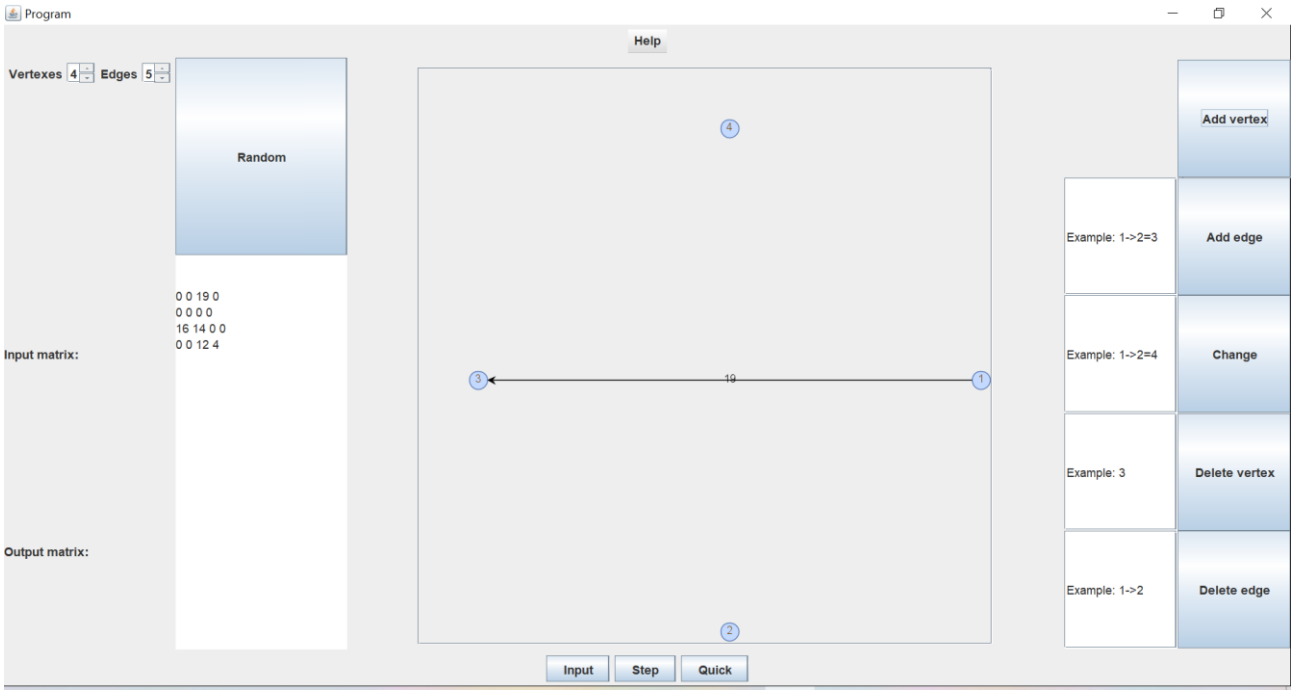
На вход программе подается матрица, которая помещается в двумерный массив для дальнейшей обработки. Массив передается в метод FloydWarshall, где реализована основная идея алгоритма. На каждой итерации цикла проверяем не нашелся ли путь через промежуточную вершину меньше, чем он был до этого, если такой имеется, то изменяем значение текущего элемента массива.

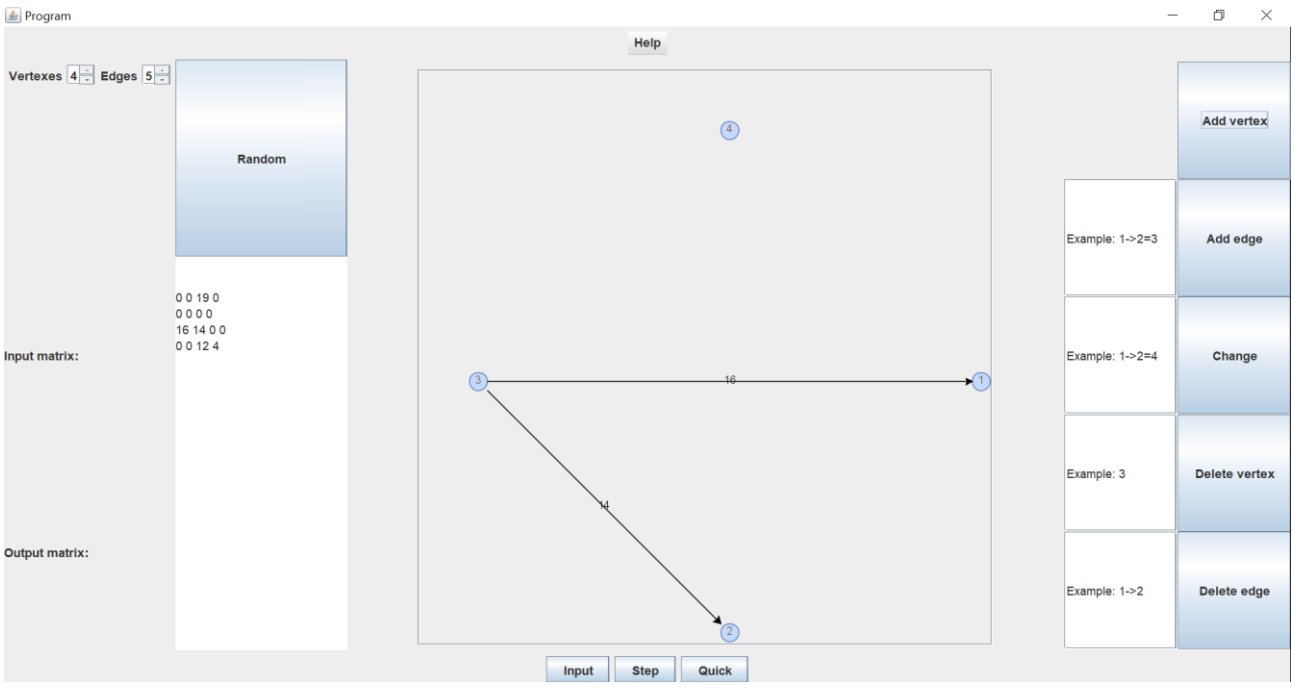
На выходе получаем матрицу с кратчайшими путями между любой парой вершин графа.

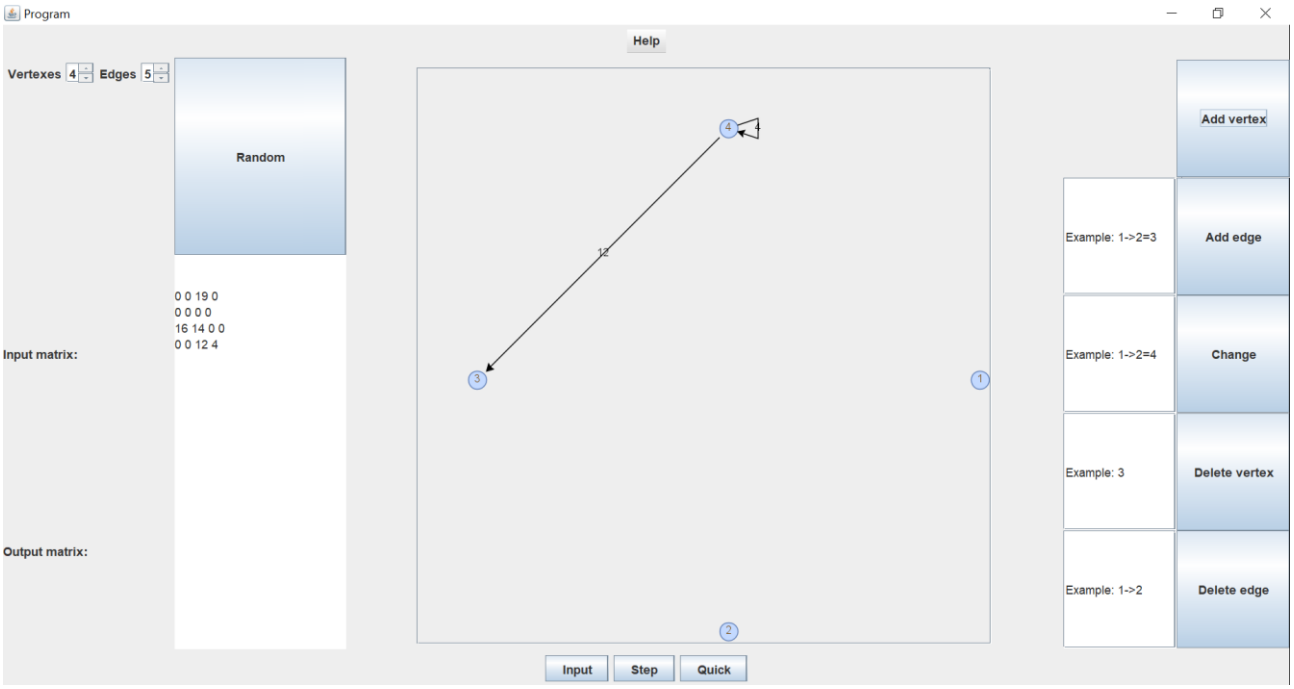
3.3. Пошаговое отображение работы алгоритма

Рис. Визуализация случайного графа с 4 вершинами и 5 ребрами




Program
Help
Vertexes 4 Edges 5
Random
0 0 19 0
0 0 0 0
16 14 0 0
0 0 12 4
Input matrix:
Output matrix:
4
3
19
1
2
Add vertex
Example: 1->2=3
Add edge
Example: 1->2=4
Change
Example: 3
Delete vertex
Example: 1->2
Delete edge
Input
Step
Quick


Program
Help
Vertexes 4 Edges 5
Random
0 0 19 0
0 0 0 0
16 14 0 0
0 0 12 4
Input matrix:
Output matrix:
4
3
16
1
14
2
Add vertex
Example: 1->2=3
Add edge
Example: 1->2=4
Change
Example: 3
Delete vertex
Example: 1->2
Delete edge
Input
Step
Quick


Program
Help
Vertexes 4
Edges 5
Random
0 0 19 0
0 0 0 0
16 14 0 0
0 0 12 4
Input matrix:
Output matrix:
4
1
12
3
2
Add vertex
Example: 1->2=3
Add edge
Example: 1->2=4
Change
Example: 3
Delete vertex
Example: 1->2
Delete edge
Input
Step
Quick

4. ТЕСТИРОВАНИЕ

4.1. План тестирования

4.1.1. Функционал, который будет протестирован

- Корректный результат работы алгоритма
- Изменение графа

4.1.2. Тестовые единицы

- Ввод графа с помощью GUI:
  - Добавление вершины
  - Удаление вершины
  - Добавление ребра
  - Удаление ребра
  - Редактирование ребра

4.1.3. Подход к тестированию

Тестирование будет реализовано с помощью фреймворка JUnit, с помощью которого удобно писать тесты для небольших частей кода.

Таблица 1. Тестовые случаи

Тестируемая функция	Входные данные	Ожидаемый результат
Vizualizator.addEdge()	Добавить существующее ребро (3-4 с весом 5).	Выбрасывается исключение.
Vizualizator.addEdge()	Добавить ребро с весом 5 между существующей (3) и удаленной (4) вершиной.	Выбрасывается исключение.
Vizualizator.addEdge()	Добавить ребро между двумя удаленными	Выбрасывается исключение.

	вершинами (3- 4, с весом 5)	
Vizualizator.addEdge()	Добавить ребро между двумя существующими вершинами (3-4, с весом 5)	Должен сработать и добавить ребро в граф.
Vizualizator.removeVert()	Удалить существующую вершину(4),удалить вершину второй раз(4).	Первое удаление сработает, второе выбросит исключение.
Vizualizator.removeEdge()	Удалить несуществующее ребро(3-4).	Выбросит исключение.
Vizualizator.removeEdge()	Удалить ребро между вершинами (3-4)	Программа сработает корректно.
void changeEdge()	Изменить ребро(3-4 с весом 5 на вес 10),изменить несуществующее ребро(4-5 с веса 5 на вес 10).	Первый раз ребро должно измениться, во втором случае выбрасывается исключение.

ЗАКЛЮЧЕНИЕ

Разработка поставленной задачи была выполнена в соответствии с

планом. Было спроектировано и запрограммировано приложения для визуализации работы алгоритма Флойда-Уоршелла. Приложения использует введенную или случайно сгенерированную матрицу, позволяет добавлять, удалять, изменять ребра и вершины в графе.

Также был разработан графический интерфейс, визуально отображающий результаты работы алгоритма и позволяющий управлять возможностями приложения. Основной алгоритм был покрыт Unit тестами.

Поставленные задачи были выполнены полностью. Таким образом разработка приложения была завершена успешно с полным выполнением плана и реализацией дополнительного функционала.

ПРИЛОЖЕНИЕ А. Исходный код.

Файл MainClass.java

```
import GUI.*;

public class MainClass {
    public static void main(String[] args) {
        GUI gui = new GUI();
        gui.setVisible(true);
    }
}
```

Файл Graph.java

```
package Graph;

import java.util.Scanner;

public class Graph {
    int n;
    int curI, curJ, curK;
    private int[][] matrix;
    static int I = 99999999; // Integer.MAX_VALUE


    public Graph(String var1) throws Exception {
        curK = 0;
        curI = 0;
        curJ = 0;
        Scanner var2 = new Scanner(var1);
        var2.useDelimiter("\n");
        int i = 0, j = 0;
        String tmp = var2.next();
        String[] arr = tmp.split(" ");
        n = arr.length;
        matrix = new int[n][n];
        for (String it : arr) {
            try{
                matrix[i][j] = Integer.parseInt(it);}
            catch(Exception e){
                System.err.println("You entered the wrong data type:use the int type.");}
            if(matrix[i][j]<0)
                throw new Exception("The matrix cannot negative numbers!");
            j++;
        }
        i++;
        j=0;
        while(var2.hasNext()) {
            tmp=var2.next();
            for (String it : tmp.split(" ")) {
                matrix[i][j]=Integer.parseInt (it);
                j++;
            }
            i++;
            j=0;
        }
        var2.close();
    }

    public void FloydWarshall() {
        int i, j, k;
        for (i = 0; i < n; i++) {
```

```
            for (j = 0; j < n; j++) {
                if (i == j)
                    matrix[i][j] = 0;
                else if (matrix[i][j] == 0)
                    matrix[i][j] = I;
            }
        }
        for (k = 0; k < n; k++) {
            for (i = 0; i < n; i++) {
                for (j = 0; j < n; j++) {
                    if ((matrix[i][k] + matrix[k][j] < matrix[i][j]) && (i != j) ) {
                        matrix[i][j] = matrix[i][k] + matrix[k][j];
                    }
                }
            }
        }
    }

    public void FWStep() {

        if(curK == n){
            System.out.println("finish");
            return;
        }

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i == j)
                    matrix[i][j] = 0;
                else if (matrix[i][j] == 0)
                    matrix[i][j] = I;
            }
        }

        if(curI < n && curJ >= n) {
            curJ = 0;
            curI++;
        }
        if(curK < n && curI >= n) {
            curI = 0;
            curK++;
        }

        System.out.println(curK);
        System.out.println(curI);

        for (; curK < n; curK++) {
            for (; curI < n; curI++) {
                for (curJ = 0; curJ < n; curJ++) {
                    //System.out.println(curJ);
                    if ((matrix[curI][curK] + matrix[curK][curJ] < matrix[curI][curJ]) ) {
                        matrix[curI][curJ] = matrix[curI][curK] + matrix[curK][curJ];
                        System.out.println("((((");
                    }

                }
                return;

            }
        }
        /*if(curJ < n) {
            if ((matrix[curI][curK] + matrix[curK][curJ] < matrix[curI][curJ]) && (curI != curJ) ) {
                changeMatrix(matrix[curI][curK] + matrix[curK][curJ]);
```

```
        }
        curJ++;
    }
    else if(curJ == n) { // обязательно выполняется
        curJ = 0;
        if(curI < n) {
            // то же самое
            if ((matrix[curI][curK] + matrix[curK][curJ] < matrix[curI][curJ]) && (curI != curJ) ) {
                changeMatrix(matrix[curI][curK] + matrix[curK][curJ]);
            }
            curI++;
        }
        else if(curI == n)  { // обязательно выполняется
            // то же самое
            curI = 0;
            if(curK < n) {
                if ((matrix[curI][curK] + matrix[curK][curJ] < matrix[curI][curJ]) && (curI != curJ) ) {
                    changeMatrix(matrix[curI][curK] + matrix[curK][curJ]);
                }
                curK++;
            }
            else {
                return;
            }
        }

    }*/

}

public String print(){
    String var1 = "";
    for ( int i = 0; i < n; ++i) {
        for ( int j = 0; j < n; ++j) {
            if(j==0){
                if(matrix[i][j]==I)
                    var1 += "I";
                else
                    var1 +=matrix[i][j];}
            else{
                if(matrix[i][j]==I)
                    var1 += " " + "I";
                else
                    var1 += " " + matrix[i][j];}
        }
        if(i!=n-1)
            var1+="\n";
    }
    System.out.println(var1);
    return var1;
}


public int[][] getMatrix() {
    return matrix;
}

public void changeMatrix(int value) {
    matrix[curI][curJ] = value;
}
public int getN() { return n;}
public int getJ() { return curJ;}
public int getI() { return curI;}
```

```
    public int getK() { return curK;}

    public void setMatrix(int[][] newMatrix) {

    }

    public void updateMatrix(){

        n = n + 1;
        matrix = java.util.Arrays.copyOf(matrix, n);

        for(int i = 0; i < n-1; i++) {
            matrix[i] = java.util.Arrays.copyOf(matrix[i], n);
        }
        matrix[n-1] = new int[n];

    }

    public void deleteEdge(int v1, int v2){
        if(v1 > 0 && v1 <= n && v2 > 0 && v2 <= n){
            System.out.println("start delete");

            matrix[v1-1][v2-1] = 0;
        }
    }
    public void changeEdge(int v1, int v2, int newEdge){
        if(v1 > 0 && v1 <= n && v2 > 0 && v2 <= n && matrix[v1-1][v2-2] != 0){//ребро существует
            matrix[v1-1][v2-2] = newEdge;
        }
    }

    public void addEdge(int v1, int v2, int newEdge){
        if(v1 > 0 && v1 <= n && v2 > 0 && v2 <= n && matrix[v1-1][v2-2] == 0){//ребро не существует
            matrix[v1-1][v2-2] = newEdge;
        }
    }

    public void deleteVert(int v){
        //убираем все связи вершины
        if(v > 0 && v <= n){
            for(int i = 0; i < n; i++){
                matrix[i][v-1] = -1;
                matrix[v-1][i] = -1;
            }
        }

    }
}
```

Файл GUI.java

```
package GUI;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.IOException;
import java.util.Random;
import java.util.Random.*;
import java.util.Scanner;

import Graph.*;
```

```
public class GUI {
    private Graph graph;
    private boolean permission = true;
    private JLabel labelIn = new JLabel("Input matrix:");
    private JTextArea textIn = new JTextArea("2 3 1 5\n0 1 7 4\n2 1 1 3\n2 5 3 0");
    private JLabel labelOut = new JLabel("Output matrix:");
    private JTextArea textOut = new JTextArea();

    private JSpinner spinnerV = new JSpinner();
    private JSpinner spinnerE = new JSpinner();

    private JButton InputButton = new JButton("Input");
    private JButton RandomButton = new JButton("Random");
    private JButton StepButton = new JButton("Step"); // by step by step by step ...
    private JButton Quick = new JButton("Quick");

    private JMenuBar menubar = new JMenuBar();
    private JMenu helpMenu = new JMenu("Help");
    private JMenuItem about = new JMenuItem("About");
    private JMenuItem info = new JMenuItem("Info");

    private JButton addVertex = new JButton("Add vertex");
    private JTextField addEdgeText = new JTextField("", 7);
    private JButton addEdgeButton = new JButton("Add edge");
    private JTextField changeEdgeText = new JTextField("", 7);
    private JButton changeButton = new JButton("Change");
    private JTextField delVertexText = new JTextField("", 7);
    private JButton delVertexButton = new JButton("Delete vertex");
    private JTextField delEdgeText = new JTextField("", 5);
    private JButton delEdgeButton = new JButton("Delete edge");

    private Vizualizator visual = null;

    private Container container = new Container();
    Container centerContainer = new Container();

    public GUI() {
        JFrame frame = new JFrame("Program");
        frame.setBounds(0, 0, 1240, 640);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        container = frame.getContentPane();
        container.setLayout(new BorderLayout());

        Container northContainer = new Container(); // менюшка
        northContainer.setLayout(new FlowLayout());
        helpMenu.add(about);
        helpMenu.add(info);
        menubar.add(helpMenu);
        northContainer.add(menubar);

        Container westContainer = new Container(); // ввод
        westContainer.setLayout(new GridLayout(3, 2));
        Container rand = new Container();
        rand.setLayout(new FlowLayout());
        rand.add(new JLabel("Vertexes"));
        rand.add(spinnerV);
        rand.add(new JLabel("Edges"));
        rand.add(spinnerE);
        westContainer.add(rand);
        RandomButton.addActionListener(new RandomActionListener());
```

```
        westContainer.add(RandomButton);
        westContainer.add(labelIn);
        westContainer.add(textIn);
        westContainer.add(labelOut);
        westContainer.add(textOut);


        //centerContainer = new Container(); // контейнер под размещение в нем графа
        //centerContainer.
        centerContainer.setLayout(new FlowLayout());


        Container eastContainer = new Container();

        eastContainer.setLayout(new GridLayout(5, 2));
        eastContainer.add(new JPanel());
        addVertex.addActionListener(new addVertexListener());
        eastContainer.add(addVertex);
        eastContainer.add(addEdgeText);
        addEdgeButton.addActionListener(new addEdgeListener());
        eastContainer.add(addEdgeButton);
        eastContainer.add(changeEdgeText);
        changeButton.addActionListener(new changeButtonListener());
        eastContainer.add(changeButton);
        eastContainer.add(delVertexText);
        delVertexButton.addActionListener(new delVertexListener());
        eastContainer.add(delVertexButton);
        eastContainer.add(delEdgeText);
        delEdgeButton.addActionListener(new delEdgeListener());
        eastContainer.add(delEdgeButton);


        Container southContainer = new Container();
        southContainer.setLayout(new FlowLayout());
        InputButton.addActionListener(new InputButtonListener ());
        southContainer.add(InputButton);
        StepButton.addActionListener(new StepListener());
        southContainer.add(StepButton);
        Quick.addActionListener(new QuickListener());
        southContainer.add(Quick);


        container.add(northContainer, BorderLayout.NORTH);
        container.add(westContainer, BorderLayout.WEST);
        container.add(centerContainer, BorderLayout.CENTER);
        container.add(eastContainer, BorderLayout.EAST);
        container.add(southContainer, BorderLayout.SOUTH);
        frame.setVisible(true);
    }
    // тут классы "реагенты" кнопочек
    class InputButtonListener implements ActionListener { // тоже можно проверить
        public void actionPerformed (ActionEvent e) {
            if(visual != null)
                return;

            if(textIn.getText().length() == 0) {
                JOptionPane.showMessageDialog(null, "Your matrix is empty", "Message", JOptionPane.PLAIN_MES-
SAGE);
            } else {

                try {
                    graph = new Graph(textIn.getText());
```

```
            } catch (Exception exception) {
                System.err.println("Failed to create a graph");
            }
            int[][] matr = graph.getMatrix();

            System.out.println(container.getSize().height);
            System.out.println(container.getSize().width);


            delVertexText.setText("Example: 3");
            addEdgeText.setText("Example: 1->2=3");
            changeEdgeText.setText("Example: 1->2=4");
            delEdgeText.setText("Example: 1->2");


            visual = new Vizualizator();
            visual.initMatrix(matr, matr[0].length);
            visual.functionVisual(centerContainer.getHeight(), centerContainer.getWidth());
            int h = centerContainer.getSize().height;
            int w = centerContainer.getSize().width;
            System.out.println(h);
            System.out.println(w);
            centerContainer.add(visual);
            container.revalidate();
            //centerContainer.setVisible(false);
            centerContainer.setVisible(true);
            /*container.add(visual);
            container.setVisible(false);
            container.setVisible(true);*/

        }
    }
}

class RandomActionListener implements ActionListener { // тоже можно проверить
    public void actionPerformed (ActionEvent e) {



        if(permission){

            delVertexText.setText("Example: 3");
            addEdgeText.setText("Example: 1->2=3");
            changeEdgeText.setText("Example: 1->2=4");
            delEdgeText.setText("Example: 1->2");

            int vertexes = (int)spinnerV.getValue();
            int edges = (int)spinnerE.getValue();
            if(vertexes <= 0) {
                JOptionPane.showMessageDialog(null, "Wrong vertexes input", "Message", JOptionPane.PLAIN_MES-
SAGE);
            }
            else if(edges <= 0 || edges >= vertexes*(vertexes-1)) {
                JOptionPane.showMessageDialog(null, "Wrong edges input", "Message", JOptionPane.PLAIN_MES-
SAGE);
            }
            else {
                int[][] matr = new int[vertexes][vertexes];
                String str = "";

                int[] randomArray = new int[edges];
                for(int i = 0; i < edges; i++){
```

```
                Random rand = new Random();
                randomArray[i] = rand.nextInt(20);
                if(randomArray[i] == 0){
                    i--;
                }
            }

            for(int i = 0; i < edges; i++){
                Random rand = new Random();
                // Random rand_k = new Random();
                int k = rand.nextInt(vertexes);
                int j = rand.nextInt(vertexes);
                if(matr[k][j] > 0){
                    i--;
                }
                else {
                    matr[k][j] = randomArray[i];
                }

            }

            str += "\n";
            str += "\n";

            for(int i = 0; i < vertexes; i++) {
                for (int j = 0; j < vertexes; j++) {
                    str += matr[i][j] + " ";
                }
                str += "\n";
            }

            /*for(int i = 0; i < vertexes; i++) {
                for(int j = 0; j < vertexes; j++) {
                    Random rand = new Random();
                    matr[i][j] = (edges > 0 ? rand.nextInt(20) : 0);
                    edges--;
                    str+=matr[i][j] + " ";
                }
                str+= "\n";
            }*/
            textIn.setText(str);
            try {
                graph = new Graph(str);
            } catch (Exception exception) {
                System.err.println("Failed to create a graph");
            }

            visual = new Vizualizator();
            visual.initMatrix(matr, matr[0].length);
            visual.functionVisual(centerContainer.getHeight(), centerContainer.getWidth());

            centerContainer.add(visual);
            centerContainer.setVisible(true);

        }


    }
  }
}

class StepListener implements ActionListener { // проверить и имплементировать графику
```

```
    public void actionPerformed(ActionEvent e) {

        //graph.print();
        permission = false;
        graph.FWStep();
       // graph.print();

        visual.updateResultMatrix(graph.getMatrix(), graph.getMatrix().length);

    }
}

class QuickListener implements ActionListener { // проверить и имплементировать графику
    public void actionPerformed(ActionEvent e) {

        permission = false;
        graph.FloydWarshall();
        visual.updateResultMatrix(graph.getMatrix(), graph.getMatrix().length);


       /* str += "\n";
        str += "\n";

        for(int i = 0; i < vertexes; i++) {
            for (int j = 0; j < vertexes; j++) {
                str += matr[i][j] + " ";
            }
            str += "\n";
        }*/

    }
}

class addVertexListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        //исключения и проверка корректности ввода
        //формат ввода - число(номер вершины)
        //добавляем вершину в матрицу и увеличиваем n
        if(permission){
            graph.updateMatrix();
            visual.addVert();
        }

    }
}
class addEdgeListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        //исключения и проверка корректности ввода
        //формат ввода: v1->v2=число

        if(permission){

            String v = addEdgeText.getText().toString();
            int splitIndex1 = v.indexOf("->",0);
            int splitIndex2 = v.indexOf("=", 0);

            String first = v.substring(0, splitIndex1);
            String second = v.substring(splitIndex1 + 2, splitIndex2);
            String third = v.substring(splitIndex2 + 1);

            int v1 = Integer.parseInt(first);
            int v2 = Integer.parseInt(second);
```

```
            int edge = Integer.parseInt(third);

            delEdgeText.setText("");

            graph.addEdge(v1, v2, edge);
            try {
                visual.addEdge(v1, v2, edge);
            } catch (IOException ioException) {
                System.err.println("Failed to add an edge");
            }
        }



    }
}
class changeButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {

        //формат ввода: v1->v2=3
        if(permission){

            String v = addEdgeText.getText().toString();
            int splitIndex1 = v.indexOf("->",0);
            int splitIndex2 = v.indexOf("=", 0);

            String first = v.substring(0, splitIndex1);
            String second = v.substring(splitIndex1 + 2, splitIndex2);
            String third = v.substring(splitIndex2 + 1);

            int v1 = Integer.parseInt(first);
            int v2 = Integer.parseInt(second);
            int edge = Integer.parseInt(third);

            delEdgeText.setText("");

            graph.changeEdge(v1, v2, edge);
            visual.changeEdge(v1, v2, edge);

        }

    }
}

class delVertexListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {

        //исключения и проверка корректности ввода
        //формат ввода: нет ввода
        if(permission){
            Scanner in = new Scanner(delVertexText.getText());
            int v = in.nextInt();
            graph.deleteVert(v);
            try {
                visual.removeVert(v);
            } catch (IOException ioException) {
                System.err.println("Failed to remove the vertex.");
            }

            delVertexText.setText("");
        }
```

```
    }
  }

  class delEdgeListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {

      //исключения и проверка корректности ввода
       //формат ввода: v1->v2

      if(permission){
        String v = delEdgeText.getText().toString();
        int splitIndex = v.indexOf("->",0);

        String first = v.substring(0, splitIndex);
        String second = v.substring(splitIndex + 2);

        int v1 = Integer.parseInt(first);
        int v2 = Integer.parseInt(second);

        delEdgeText.setText("");

        graph.deleteEdge(v1, v2);
        try {
          visual.removeEdge(v1, v2);
        } catch (IOException ioException) {
          System.err.println("Failed to remove an edge");
        }
      }


    }
  }

}
```

Файл Vizualizator.java

```
package GUI;

import com.mxgraph.layout.mxCircleLayout;
import com.mxgraph.layout.mxParallelEdgeLayout;
import com.mxgraph.model.mxCell;
import com.mxgraph.swing.mxGraphComponent;
import com.mxgraph.util.mxEvent;

import com.mxgraph.util.mxEventSource;
import com.mxgraph.view.mxGraph;

import com.mxgraph.view.mxEdgeStyle;
//import com.mxgraph.view.mxStylesheet;


import com.mxgraph.util.mxConstants;



import javax.swing.*;
//import java.io.PrintWriter;
import java.awt.event.*;
import java.io.IOException;
```

```
import java.util.*;



public class Vizualizator extends JPanel{
   private static int n = 0;
   private static int curCount = 0;
   private int height;
   private int width;
   int[][] matrix;//заданная изначально матрица
   int[][] resultMatrix;//матрица достижимости
   int[] vertName;


   private mxGraph graph;
   private mxGraph stepGraph;
   private mxGraphComponent graphComponent;//модель графа
   private mxGraphComponent stepGraphComponent;//модель графа достижимости на текущем шаге
   private Object parent;
   private Object points[];
   //private HashMap <Object, HashMap<Object, Object>> edges;
   private MouseAdapter mouseAdapter;
  // private mxEventSource.mxIEventListener listener;
   private MouseMotionListener eventListener;

   public void updateResultMatrix(int[][] matrix, int curN){
      //this.setSize();

      if(curN != n)
         return;
      resultMatrix = java.util.Arrays.copyOf(resultMatrix, n);

      for(int i = 0; i < n; i++) {
         resultMatrix[i] = java.util.Arrays.copyOf(matrix[i], n);
      }
   }

   private void upDateVertCount(boolean flag){
      if(flag) {
         n = n + 1;
         points = java.util.Arrays.copyOf(points, n);
         vertName = java.util.Arrays.copyOf(vertName, n);
         matrix = java.util.Arrays.copyOf(matrix, n);
         resultMatrix = java.util.Arrays.copyOf(resultMatrix, n);

         for(int i = 0; i < n-1; i++) {
            matrix[i] = java.util.Arrays.copyOf(matrix[i], n);
            resultMatrix[i] = java.util.Arrays.copyOf(resultMatrix[i], n);
         }
         matrix[n-1] = new int[n];
         resultMatrix[n-1] = new int[n];

         for(int i = 0; i < n; i++){
            matrix[n-1][i] = 0;
            //check rs matr
            resultMatrix[n-1][i] = 99999999;
         }
      }

   }

   public void initMatrix(int[][] adjMatrix, int newN){
```

```
    n = newN;
    curCount = n;
    vertName = new int[n];
    matrix = new int[n][n];
    resultMatrix = new int[n][n];
    points = new Object[n];//объекты вершмн


    matrix = java.util.Arrays.copyOf(adjMatrix, n);
    resultMatrix = java.util.Arrays.copyOf(adjMatrix, n);
    for(int i = 0; i < n; i++){
        matrix[i] = java.util.Arrays.copyOf(adjMatrix[i], n);
        resultMatrix[i] = java.util.Arrays.copyOf(adjMatrix[i], n);
        vertName[i] = 0;
    }

}

private void returnGraphModel(){

    this.remove(this.getComponents()[0]);
    this.setVisible(false);
    this.add(graphComponent);
    this.setVisible(true);
    this.revalidate();

}
public void functionVisual(int h, int w) {

    graph = new mxGraph();
    height = h;
    width = w;
    this.setSize(h, w);


   // boolean permission = false;

   if(mouseAdapter == null)
       mouseAdapter = new MouseAdapter() {
           @Override
           public void mouseClicked(MouseEvent mouseEvent) {
               //permission = true;
               mxCell cell = null;
               if(graphComponent != null) {
                   cell = (mxCell) graphComponent.getCellAt(mouseEvent.getX(), mouseEvent.getY());
               }
               if(cell != null)
                   displayStepResult((int)cell.getValue());
               else{
                   if(stepGraphComponent != null){
                       returnGraphModel();
                   }
               }
               super.mouseClicked(mouseEvent);
           }
           @Override
           public void mouseDragged(MouseEvent e) {
               //System.out.println("?????//");
               //graphComponent.getToolkit().
               //graphComponent.getCom
               mxCell cell = null;
```

```
            cell = (mxCell) graphComponent.getCellAt(e.getX(), e.getY());
            if(cell != null) {

               if(cell.isVertex() && cell.isConnectable()){
                  System.out.println("*****");
                  super.mouseDragged(e);
               }else{
                  //mxGraph.prototype.resetEdgesOnMove:
                  /*graph.getModel();
                  graph.startEditingAtCell(change.child);
                  grap.start*/

                  /*if(cell.isConnectable())
                     return;*/
                 // graphComponent.getCursor().
                  //super.mouseReleased();
                  // super.mouseReleased(e);
               }
               //return;
               //

            }else {
               return;
               //System.out.println("?????//");
            }

            //super.mouseDragged(e);
         }

      };
/*if(eventListener == null)
   eventListener = new EventListener() {
      @Override
      public String toString() {
         return super.toString();
      }
   }*/
/* eventListener = new MouseMotionAdapter() {
    @Override
    public void mouseDragged(MouseEvent mouseEvent) {
       System.out.println("$$$$$");
       mxCell cell = null;
       cell = (mxCell) graphComponent.getCellAt(mouseEvent.getX(), mouseEvent.getY());
       if(cell != null)
          super.mouseDragged(mouseEvent);
    }
 };*/



 parent = graph.getDefaultParent();
 graph.getModel().beginUpdate();


 double phi0 = 0;
 double phi = 2 * Math.PI / n;
 int r = 250; // радиус окружности

 for (int i = 0; i < points.length; i++) {
    //points[i] - вершина
       if(vertName[i] == -1)//вершина удалена
```

```
            continue;
          points[i] = graph.insertVertex(parent, null, i + 1, 300 + r * Math.cos(phi0), 300 + r * Math.sin(phi0), 18,
18, "shape=ellipse");
          //stepGraph.insertVertex()
          phi0 += phi;
          vertName[i] = 1;

    }

      for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
          if(matrix[i][j] > 0) {
            //var style = graph.getStylesheet().getDefaultEdgeStyle();

            //Object o = graph.createEdge(parent, null, matrix[i][j], points[i], points[j],"edgeStyle=myEdgeStyle");
            //graph.createEdge()
            Object o = graph.insertEdge(parent, null, matrix[i][j], points[i], points[j]);
            //graph.getModel().is
            //o.
            //graph.getModel().setStyle(, "edgeStyle=myEdgeStyle");
            //style.put("strokeColor", standartColor);
            //graph.insertEdge(parent, null, matrix[i][j], points[i], points[j]);
          }
        }
      }

    graph.getModel().endUpdate();

    mxParallelEdgeLayout layout = new mxParallelEdgeLayout(graph);
    layout.execute(graph.getDefaultParent());

    //graph.getModel().addListener(mxEvent.CHANGE, listener);


    graphComponent = new mxGraphComponent(graph);


    graphComponent.getGraphControl().addMouseListener(mouseAdapter);


    //graphComponent.getGraphControl().removeMouseMotionListener(graphComponent.getMouseMotionListen-
ers()[0]);
    graphComponent.getGraphControl().addMouseMotionListener(mouseAdapter);


    this.add(graphComponent);
    this.revalidate();
  }

  public void addVert(){

    curCount++;
    upDateVertCount(true);

    graph.getModel().beginUpdate();
    points[n-1] = graph.insertVertex(parent, null, n, 300, 300, 18, 18, "shape=ellipse");
    vertName[n - 1] = 1;
    graph.getModel().endUpdate();


  }
```

```
  public void removeVert(int vertID) throws IOException {//название вершины (от 1 и ...)

    if(vertID > 0 && vertID < n + 1){
      curCount--;
      //upDateVertCount(false);
      graph.getModel().beginUpdate();

      Object pointsForRemove[] = new Object[1];

      pointsForRemove[0] = points[vertID-1];

      graph.removeCells(pointsForRemove);

      graph.getModel().endUpdate();

      for(int i = 0; i < n; i++){
        matrix[i][vertID-1]=0;
        matrix[vertID-1][i]=0;

      }

      vertName[vertID-1] = -1;

    }
    else{
      throw new IOException("This vertex does not exist");      }

  }

  public void addEdge(int v1, int v2, int edge) throws IOException {


    if(v1 > 0 && v2 > 0 && v1 < n + 1 && v2 < n + 1 && vertName[v1-1] == 1 && vertName[v2-1] ==
1){//условия существования вершин

      graph.getModel().beginUpdate();
      matrix[v1-1][v2-1] = edge;

      graph.getModel().beginUpdate();

      this.remove(graphComponent);

      functionVisual(height, width);


      graph.getModel().endUpdate();

    }
    else{
      throw new IOException("This edge does not exist");
    }


  }

  public void changeEdge(int v1, int v2, int newEdge){
    try{
    if(v1 > 0 && v2 > 0 && v1 < n + 1 && v2 < n + 1 && vertName[v1-1] == 1 && vertName[v2-1] ==
1){//условия существования вершин
      graph.getModel().beginUpdate();
      removeEdge(v1, v2);
      addEdge(v1, v2, newEdge);
```

```
            graph.getModel().endUpdate();
        }}
        catch(IOException e){
            System.err.println("This edge can't be change");
        }


    }

    public void removeEdge(int v1, int v2) throws IOException {

        if(v1 > 0 && v2 > 0 && v1 <= n && v2 <= n && vertName[v1-1] == 1 && vertName[v2-1] == 1){//условия
существования вершин

            System.out.println("next step");
            matrix[v1-1][v2-1] = 0;

            graph.getModel().beginUpdate();
            this.remove(graphComponent);
            functionVisual(height, width);
            //m = 0;
            graph.getModel().endUpdate();

        }
        else{
            throw new IOException("This edge does not exist");
        }

    }

    private void displayStepResult(int vert){//матрица достижимсти на текущем шаге

        if(vert <= 0){return;}//никакая вершина не выбрана, но можно нарисовать все ребра

        this.remove(this.getComponents()[0]);
        this.setVisible(false);
        this.setVisible(true);


        stepGraph = new mxGraph();
        stepGraph.getModel().beginUpdate();
        Object[] stepPoints = new Object[n];


        double phi0 = 0;
        double phi = 2 * Math.PI / n;
        int r = 250; // радиус окружности

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

            if(vertName[i] > 0){
                stepPoints[i] = stepGraph.insertVertex(stepGraph.getDefaultParent(), null, i+1, 300 + r * Math.cos(phi0),
300 + r * Math.sin(phi0), 18, 18,"shape=ellipse" );
                phi0 += phi;
            }

        }
        stepGraph.getModel().endUpdate();


          stepGraph.getModel().beginUpdate();
            for(int i = 0; i < n; i++){
```

```
            int edge = resultMatrix[vert-1][i];

            if(edge > 0) {

               var style = stepGraph.getStylesheet().getDefaultEdgeStyle();
               style.put("strokeColor", "#000000");
               style.put("fontColor", "#000000");

               stepGraph.getModel().setStyle(stepGraph.insertEdge(stepGraph.getDefaultParent(), null, edge,
stepPoints[vert-1], stepPoints[i]), "edgeStyle=myEdgeStyle");

            }
         }

      stepGraph.getModel().endUpdate();
      stepGraphComponent = new mxGraphComponent(stepGraph);

      stepGraphComponent.getGraphControl().addMouseListener(mouseAdapter);


      this.add(stepGraphComponent);
      this.revalidate();

   }


   public void displayResult(int[][] matr) {//рисует граф по матрице достижимости

      //this.remove(graphComponent);


      graph = new mxGraph();
      parent = graph.getDefaultParent();
      graph.getModel().beginUpdate();
      double phi0 = 0;
      double phi = 2 * Math.PI / n;
      int r = 250; // радиус окружности


      //отображаем все вершины
      for (int i = 0; i < points.length; i++) {
         //points[i] - вершина
         if (vertName[i] == -1)//вершина удалена
            continue;
         points[i] = graph.insertVertex(parent, null, i + 1, 300 + r * Math.cos(phi0), 300 + r * Math.sin(phi0), 18, 18,
"shape=ellipse");
         phi0 += phi;
      }


      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
            if (matrix[i][j] > 0) {
               HashMap<Object, Object> valH = new HashMap<Object, Object>();
               //вес ребра между вершинами - длина кратчайшего пути между ними
               //var edgeStyle = graph.getStylesheet().getDefaultEdgeStyle();
              // edgeStyle.put(mxConstants.STYLE_EDGE, mxEdgeStyle.EntityRelation);

               valH.put(points[j], graph.insertEdge(parent, null, matr[i][j], points[i], points[j]));//, mxCon-
stants.STYLE_EDGE));
               //edges.put(points[i], valH);
```

```
            }
          }
        }

        graph.getModel().endUpdate();

       // mxParallelEdgeLayout layoutParallel = new mxParallelEdgeLayout(graph);
       // mxCircleLayout layoutCircle = new mxCircleLayout(graph);
        //layoutParallel.execute(graph.getDefaultParent());
       //  layoutCircle.execute(graph.getDefaultParent());

        //graphComponent - наша текущая модель
        this.remove(graphComponent);

        stepGraphComponent = new mxGraphComponent(graph);
        this.add(stepGraphComponent);

        //graphComponent = new mxGraphComponent(graph);
        //this.add(graphComponent);

        this.revalidate();
    }

}
```

Файл VizualizatorTest.java

```
package GUI;

import org.junit.Assert;
import org.junit.jupiter.api.*;

import java.io.IOException;

import static org.junit.jupiter.api.Assertions.*;

class VizualizatorTest {

    private Vizualizator viz;

    @Test
    void addVert() throws IOException {
        viz=new Vizualizator();
        int matrix[][]={{0,1,2},{3,0,4},{5,6,0}};
        viz.initMatrix(matrix,3);
        viz.functionVisual(20,30);
        viz.addVert();
        viz.addEdge(3,4,1);
        assertTrue(viz.matrix[2][3]==1);
    }
    @Test
    void addNonEdge_() throws IOException {
        try{
            viz=new Vizualizator();
            int matrix[][]={{0,1,2},{3,0,4},{5,6,0}};
            viz.initMatrix(matrix,3);
            viz.functionVisual(20,30);
            viz.addVert();
            viz.addVert();
            viz.removeVert(4);
            viz.removeVert(5);
            viz.addEdge(4,5,5);
```

```
    }
    catch(IOException e){
        Assert.assertEquals("This edge does not exist",e.getMessage());
    }
}
@Test
void removeVert() throws IOException {
    viz=new Vizualizator();
    int matrix[][]={{0,1,2},{3,0,4},{5,6,0}};
    viz.initMatrix(matrix,3);
    viz.functionVisual(20,30);
    viz.removeVert(1);
    assertTrue(viz.matrix[0][1]==0);
}

@Test
void addEdge() throws IOException {
    viz=new Vizualizator();
    int matrix[][]={{0,1,2},{3,0,4},{5,6,0}};
    viz.initMatrix(matrix,3);
    viz.functionVisual(20,30);
    viz.addVert();
    viz.addEdge(3,4,5);
    assertTrue(viz.matrix[2][3]==5);
}

@Test
void addNonEdge(){
    try{
        viz=new Vizualizator();
        int matrix[][]={{0,1,2},{3,0,4},{5,6,0}};
        viz.initMatrix(matrix,3);
        viz.functionVisual(20,30);
        viz.addVert();
        viz.removeVert(4);
        viz.addEdge(3,4,5);
    }
    catch(IOException e){
        Assert.assertEquals("This edge does not exist",e.getMessage());
    }
}

@Test
void addSecondEdge(){
    try{
        viz=new Vizualizator();
        int matrix[][]={{0,1,2},{3,0,4},{5,6,0}};
        viz.initMatrix(matrix,3);
        viz.functionVisual(20,30);
        viz.addVert();
        viz.addEdge(3,4,5);
        viz.addEdge(3,4,5);
    }
    catch(IOException e){
        Assert.assertEquals("This edge does not exist",e.getMessage());
    }
}

@Test
void changeEdge() throws IOException {
    viz=new Vizualizator();
    int matrix[][]={{0,1,2},{3,0,4},{5,6,0}};
```

```
    viz.initMatrix(matrix,3);
    viz.functionVisual(20,30);
    viz.addVert();
    viz.addVert();
    viz.addEdge(4,5,5);
    viz.changeEdge(4,5,10);
    assertTrue(viz.matrix[3][4]==10);
}

@Test
void changeNonEdge() throws IOException {
    try {
        viz = new Vizualizator();
        int matrix[][] = {{0, 1, 2}, {3, 0, 4}, {5, 6, 0}};
        viz.initMatrix(matrix, 3);
        viz.functionVisual(20, 30);
        viz.addVert();
        viz.addVert();
        viz.addEdge(3, 4, 5);
        viz.changeEdge(5, 6, 10);
    } catch (IOException e) {
        Assert.assertEquals("This edge does not exist", e.getMessage());
    }
}

    @Test
    void removeEdge() throws IOException {
        viz=new Vizualizator();
        int matrix[][]={{0,1,2},{3,0,4},{5,6,0}};
        viz.initMatrix(matrix,3);
        viz.functionVisual(20,30);
        viz.addVert();
        viz.addVert();
        viz.addEdge(4,5,5);
        viz.removeEdge(3,4);
        System.out.println(viz.matrix[3][4]);
        assertFalse(viz.matrix[3][4]==0);
    }

    @Test
    void removeNonEdge(){
        try{
            viz=new Vizualizator();
            int matrix[][]={{0,1,2},{3,0,4},{5,6,0}};
            viz.initMatrix(matrix,3);
            viz.functionVisual(20,30);
            viz.addVert();
            viz.addEdge(3,4,5);
            viz.removeEdge(3,4);
            viz.removeEdge(3,4);}
        catch(IOException e){
            Assert.assertEquals("This edge does not exist",e.getMessage());
        }
    }


    @Test
    void removeNonVert(){
    try{
        viz = new Vizualizator();
        int matrix[][] = {{0, 1, 2}, {3, 0, 4}, {5, 6, 0}};
        viz.initMatrix(matrix, 3);
```

```
            viz.functionVisual(20,30);
            viz.addVert();
            viz.removeVert(4);
            viz.removeVert(4);}
        catch (IOException e){
            Assert.assertEquals("This vertex does not exist",e.getMessage()); }
        }

    }
```

Файл GraphTest.java

```
package Graph;

import org.junit.jupiter.api.*;

import static org.junit.jupiter.api.Assertions.assertEquals;


class GraphTest {
    private Graph graph,graph1;

    @BeforeEach
    void setUp() throws Exception {
        String str="0 0 3 0\n1 0 5 0\n0 0 0 1\n4 8 0 0";
        String str1="0 10 18 8 0 0\n10 0 16 9 21 0\n0 16 0 0 0 15\n7 9 0 0 0 12\n0 0 0 0 0 23\n0 0 15 0 23 0";
        graph=new Graph(str);
        graph1=new Graph(str1);
    }

    @Test
    void floydWarshall() {
        graph.FloydWarshall();
        assertEquals("0 12 3 4\n1 0 4 5\n5 9 0 1\n4 8 7 0",graph.print());
        graph1.FloydWarshall();
        assertEquals("0 10 18 8 31 20\n10 0 16 9 21 21\n26 16 0 25 37 15\n7 9 25 0 30 12\n64 54 38 63 0 23\n41
31 15 40 23 0",graph1.print());
    }

}
```