

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

ОТЧЕТ
по учебной практике
ТЕМА: ВИЗУАЛИЗАЦИЯ АЛГОРИТМА ФЛОЙДА-УОРШЕЛЛА

Студент гр. 9304	_____	Ковалёв П.Д.
Студент гр. 9304	_____	Борисовский В.Ю.
Студент гр. 9304	_____	Прокофьев М.Д.
Руководитель	_____	Фиалковский М.С.

Санкт-Петербург

2021

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Ковалёв П.Д. группы 9304

Студент Борисовский В.Ю. группы 9304

Студент Прокофьев М. Д. группы 9304

Тема практики: визуализация алгоритма Флойда-Уоршелла

Задание на практику:

Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.

Алгоритм: алгоритм Флойда-Уоршелла.

Сроки прохождения практики: 01.07.2021 – 14.07.2021

Дата сдачи отчета: 10.07.2021

Дата защиты отчета: 12.07.2021

Студент

Ковалёв П.Д.

Студент

Борисовский В.Ю.

Студент

Прокофьев М.Д.

Руководитель

Фиалковский М.С.

АННОТАЦИЯ

Цель практики – научиться работать в команде и улучшить умение писать код в объектно-ориентированном стиле на языке программирования Java. Изучить основы языка программирования Java, а также средства разработки приложений с графическим интерфейсом на данном языке. В рамках практики выполняется мини-проект в команде, суть которого – реализация визуализатора графового алгоритма средствами языка Java. В процессе работы предстоит разработать прототип интерфейса, реализовать сам алгоритм, а также при помощи средств тестирования отладить разработанную программу. Нашей командой был выбран алгоритм Флойда-Уоршелла.

SUMMARY

The goal of the practice is to learn how to work in a team and improve the ability to write code in an object-oriented style in the Java programming language. Learn the basics of the Java programming language, as well as tools for developing applications with a graphical interface in this language. As part of the practice, a mini-project is carried out in a team, the essence of which is the implementation of a graph algorithm visualizer using the Java language. In the process of work, it is necessary to develop a prototype of the interface, implement the algorithm itself, and also use testing tools to debug the developed program. Our team chose the Floyd-Warshall algorithm.

СОДЕРЖАНИЕ

	Введение	5
1.	Требования к программе	6
1.1.	Исходные требования к программе	6
1.1.1	Требования к визуализации	7
1.1.2	Требования к входным данным	9
1.1.3	Требования к архитектуре	10
1.2.	Уточнение требований после сдачи прототипа	11
2.	План разработки и распределение ролей в бригаде	13
2.1.	План разработки	13
2.2.	Распределение ролей в бригаде	13
3.	Особенности реализации	14
3.1.	Структуры данных	14
3.2.	Основные методы	15
3.2.1	Основные методы класса Main	15
3.2.2	Основные методы класса FileOpening	16
3.2.3	Основные методы класса MouseAdapterFabric	17
3.2.4	Основные методы класса MatrixAdapterFabric	18
3.2.5	Основные методы класса TableAdapterFabric	19
3.2.6	Основные методы класса VertexEnumerator	20
4.	Тестирование	21
4.1	Тестирование графического интерфейса	21
4.2	Тестирование итогового проекта	22
	Заключение	23
	Список использованных источников	24
	Приложение А. Исходный код – только в электронном виде	25

ВВЕДЕНИЕ

Основная цель практики – реализовать визуализатор алгоритма Флойда-Уоршелла. Алгоритм предназначен для нахождения кратчайших путей между вершинами во взвешенном графе. Для реализации проекта, необходимо реализовать графический интерфейс, сам алгоритм и объединить данные наработки.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные требования к программе

Приложение будет иметь графический интерфейс, через который будет возможно вводить данные (вершины и рёбра графа), так же будет реализована поддержка ввода данных через файл. Программа будет иметь несколько режимов работы: многошаговый режим, суть которого – пошаговая демонстрация работы алгоритма и одношаговый – программа вычислит кратчайшие пути в переданном ей графе и перечислит стоимости переходов от одних вершин к другим в области “Матрица смежности” в окне программы.

1.1.1 Требования к визуализации

Окно программы будет выглядеть следующим образом:

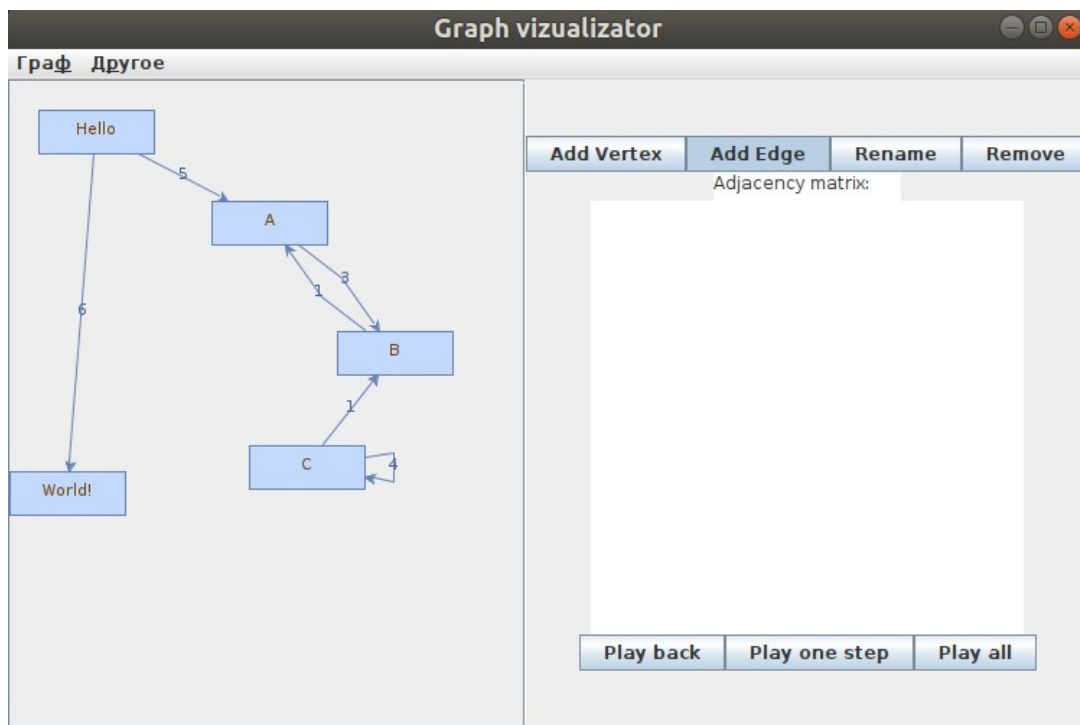


Рисунок 1 – Окно программы

Из рисунка 1 видно, что программа будет реализовывать следующий функционал:

Добавление/удаление вершин и рёбер. После нажатия соответствующих кнопок, клик левой кнопки мыши будет создавать или удалять вершину графа, а также добавлять ребро между вершинами. При добавлении ребра будет всплывать окошко, в котором нужно будет ввести вес ребра. Это видно на рисунке 2.

Переименование. Данная кнопка позволяет переименовать вершину, нажатием на нее. При нажатии появляется окошко, в котором пользователю будет предложено ввести новое название вершины.

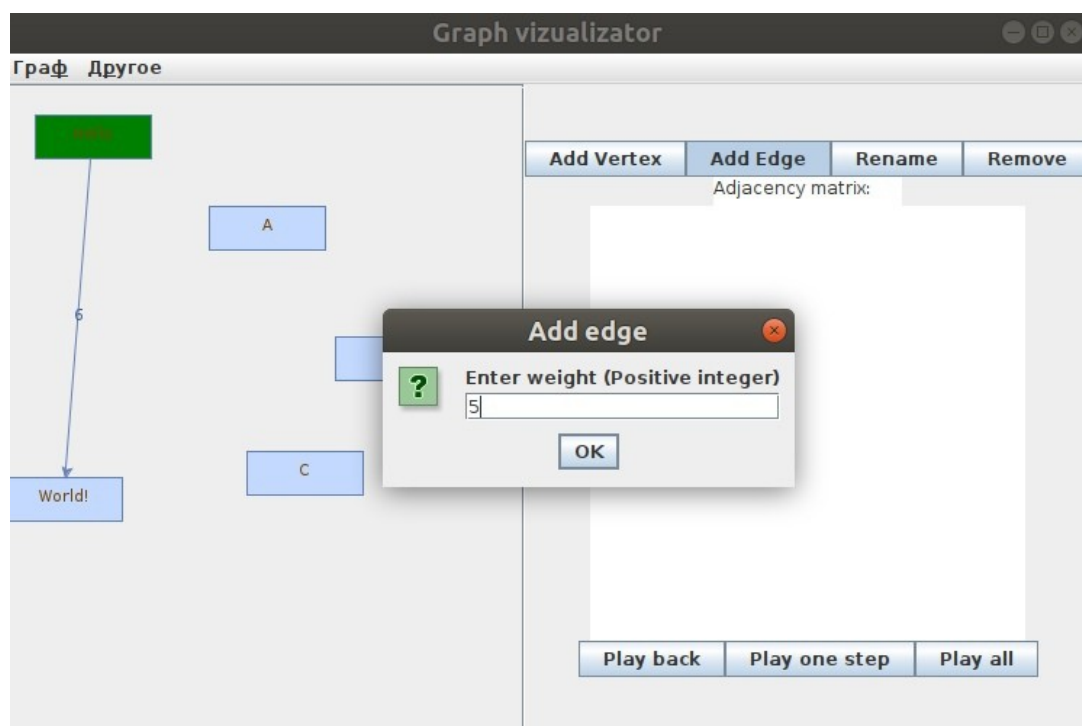


Рисунок 2 – Добавление ребра

При нажатии на “Граф” доступны две опции: загрузить граф из файла – появится окошко, в котором можно будет перемещаться по файловой системе для нахождения файла с графом и создать новый граф (подразумевается, что данное действие очистит рабочую область).

При нажатии на “Другое”, можно будет просмотреть вкладки “справка” и “о программе”.

Сделать шаг. Данная опция позволит шаг за шагом увидеть работу алгоритма.

Сделать шаг назад. Данная опция дает возможность откатиться на один шаг назад в визуализации алгоритма.

Запустить полностью. Данная опция позволит запустить алгоритм на графе без демонстрации пошаговой обработки графа.

На рисунке 1 также видно поле “Adjacency matrix”, которое представляет собой матрицу кратчайших путей, которая будет изменяться в процессе работы алгоритма.

Также, слева видна рабочая область, в которой будет находиться граф.

1.1.2 Требования к входным данным

Входные данные можно задать двумя способами: вручную создать граф, добавив вершины и рёбра в окне программы или загрузить текстовый файл, содержащий матрицу смежности графа, при помощи кнопки “Loading”. Если пользователь хочет загрузить текстовый файл, то он должен поместить туда данные следующим образом: на первой строке находится количество строк матрицы смежности, на второй – количество столбцов, после чего располагается сама матрица смежности графа. В случае неверных данных в текстовом файле, программа получит исключение об ошибке и уведомит об этом пользователя, после чего завершится.

Исходя из алгоритма Флойда-Уоршелла, требования к входным данным следующие: веса рёбер следует подавать неотрицательными (алгоритм не предназначен для работы с отрицательными рёбрами) и целыми, т.к. при работе с вещественными весами, высокая точность вычислений не гарантируется.

1.1.3 Требования к архитектуре

Проект выполняется средствами языка программирования Java. Для визуализации используется библиотека Swing. Данная библиотека была выбрана потому, что предоставляет большой набор связанных с ней библиотек, которые в свою очередь позволяют визуализировать графы. В частности, в данном проекте используется библиотека jgraphx (<https://github.com/vlsi/jgraphx-publish>).

В реализации алгоритма Флойда-Уоршелла граф хранится в матрице смежности и все действия производятся с ней. Тип данных элементов матрицы – целые числа. Результатом работы алгоритма является набор кратчайших путей между вершинами.

1.2. Уточнение требований после попытки сдачи спецификации

После сдачи спецификации был добавлен следующий функционал: кнопка “Play back”, которая позволяет откатиться на один шаг назад в пошаговом выполнении алгоритма, вкладки “Граф” и “Другое”. Первая вкладка позволяет очистить рабочую область и загрузить граф из файла, а вторая - просмотреть вкладки “справка” и “о программе” соответственно.

Также была добавлена кнопка Algo Mode, которая позволяет производить алгоритмические действия с графом, но при этом запрещает изменять граф. Реализованный функционал можно видеть на рисунке 3.

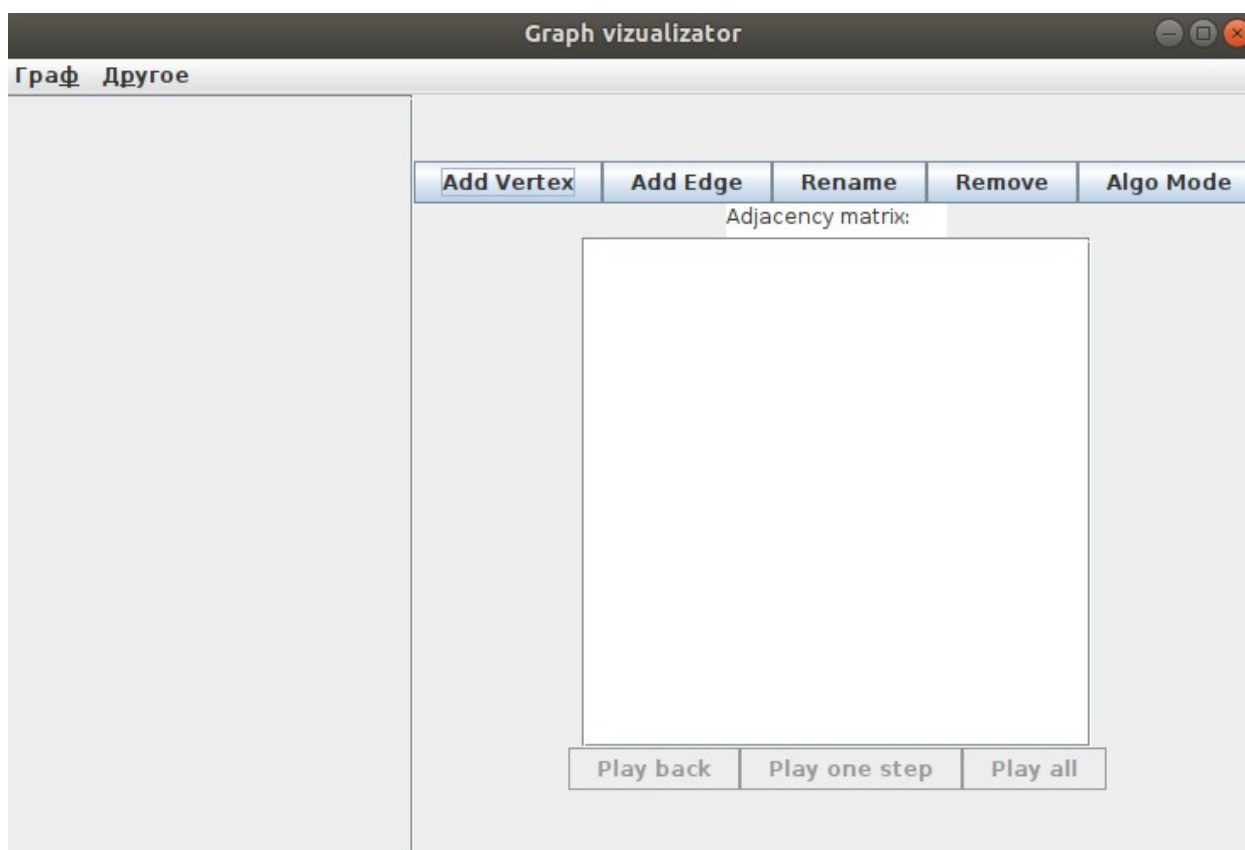


Рисунок 3 - Кнопка Algo Mode

На рисунке 4 можно видеть пример работы программы.

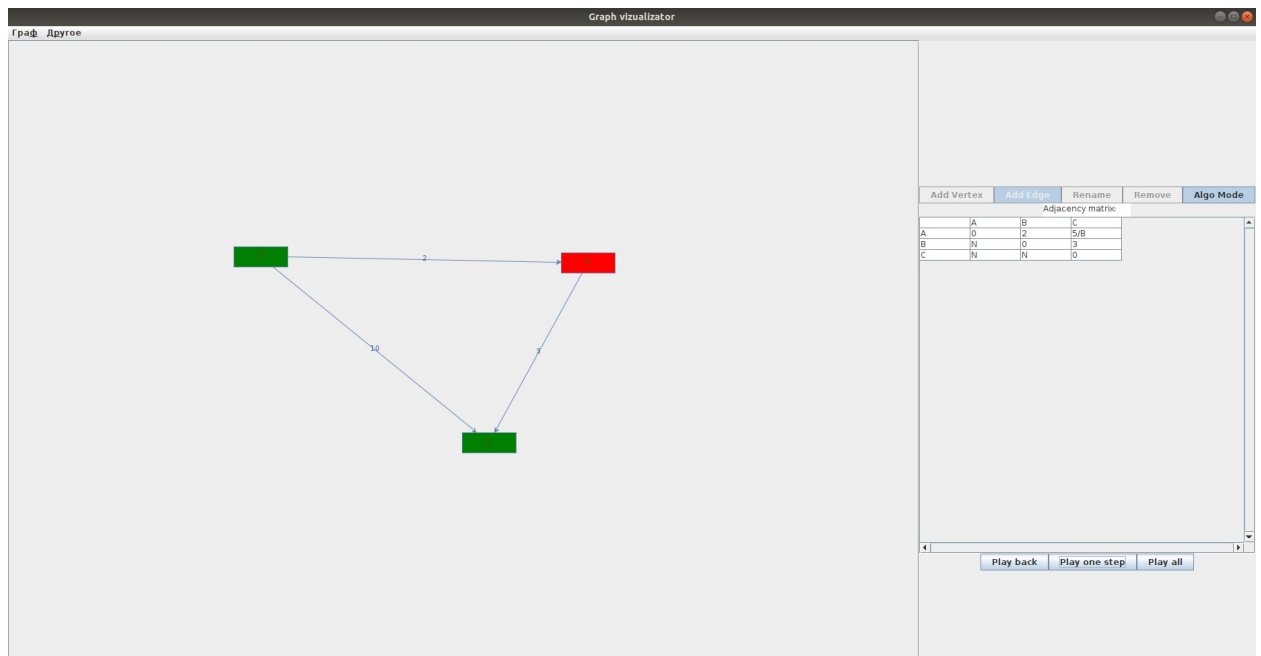


Рисунок 4 — Пример работы программы

Из рисунка 4 видно, что визуализатор успешно выполняет свою задачу: подсвечивает опорную вершину красным цветом, а вершины, между которыми улучшает путь — зеленым. Также он заполняет матрицу смежности в правой части окна приложения.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

До 01.07.2021 – Распределение по бригадам и выбор темы мини-проекта

До 07.07.2021 – Сдача вводного задания

До 07.07.2021 – Согласование спецификации. Создание прототипа графического интерфейса.

До 10.07.2021 – Сдача второго этапа

До 14.07.2021 – Сдача финальной версии мини-проекта

2.2. Распределение ролей в бригаде

Ковалёв Павел – разработка алгоритма и структуры данных, хранящей граф, ведение документации, тестирование.

Борисовский Виктор – разработка и дизайн графического интерфейса

Прокофьев Михаил – лидер, объединение алгоритма и GUI, координация работы алгоритмиста и разработчика интерфейса.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

В качестве структуры данных используется вектор векторов, который представляет из себя матрицу смежности графа. Данная структура данных используется в алгоритме для вычисления кратчайших путей в графе.

3.2. Основные методы

3.2.1 Основные методы класса **Main**

Данный класс отвечает за графическую составляющую приложения.

private JMenu createMenuItems() — метод создает меню в левом верхнем углу приложения

private void deleteGraph(mxGraph g) — метод очищает объект класса *mxGraph*

void setAllVertexDefault(mxGraph g) — метод устанавливает стиль вершин по умолчанию

public Main() - метод осуществляющий создание графического интерфейса

3.2.2 Основные методы класса FileOpening

Данный класс отвечает за загрузку данных из файла.

void ReadingData() — метод считывает матрицу смежности из файла

int getVertexNum() — метод возвращает число вершин

Vector<Vector<Integer>> getMatrixAdj() — метод возвращает матрицу смежности

3.2.3 Основные методы класса **MouseAdapterFabric**

Данный класс отвечает за взаимодействие с интерфейсом программы.

MouseAdapter *getAddEdgeMouseAdapter()* — метод добавляет ребро по клику мыши

void *getAddEdgeMouseAdapterFromFile(Vector<Vector<Integer>> my_vec)* — метод добавляет ребро для графа из файла

MouseAdapter *getAddVertexMouseAdapter()* — метод добавляет вершину по клику мыши

void *getAddVertexMouseAdapterFromFile(VertexEnumerator vertexEnumerator)* — метод добавляет вершину для графа из файла

MouseAdapter *getRenameVertexMouseAdapter()* — метод осуществляет переименование вершины

MouseAdapter *getDeleteCellMouseAdapter()* — метод осуществляет удаление объекта класса *Cell*. Используется при удалении вершин и ребер в графе

MouseAdapter *getMoveVertexMouseAdapter()* — метод позволяет перемещать вершины графа

boolean *connectionVertex(mxCell cell1, mxCell cell2)* — метод возвращает *true*, если вершины *cell1* и *cell2* соединены и *false* в противном случае

static void *getRunStep()* — метод осуществляет выполнение шага вперед в алгоритме

static boolean *getRunUndo()* — метод осуществляет выполнение шага назад в алгоритме

static void *getRunAll()* — метод осуществляет безшаговое выполнение алгоритма

static void *getDeleteAll()* — метод осуществляет удаление графа

static void *getDeleteAllSlashText()* — метод осуществляет очистку матрицы смежности

3.2.4 Основные методы класса **MatrixAdapterFabric**

Данный класс отвечает за работу с матрицами в программе.

static void addVertexMatrix(Vector<Vector<Integer>> my_vec) — метод добавляет элемент в матрицу смежности

static void addVertexMatrixSlashText(Vector<Vector<String>> my_vec) — метод добавляет в матрицу пометку опорной вершины

static void addEdgeMatrix(Vector<Vector<Integer>> my_vec, int i, int j, int value) — метод добавляет ребро в матрицу

static void deleteVertexMatrix(Vector<Vector<Integer>> my_vec, int number) — метод удаляет вершину из матрицы

static void deleteVertexMatrixSlashText(Vector<Vector<String>> my_vec, int number) — метод удаляет пометку опорной вершины из матрицы

static void deleteEdgeMatrix(Vector<Vector<Integer>> my_vec, int i, int j) — метод удаляет ребро из матрицы

static void deleteVertexMatrixAll(Vector<Vector<Integer>> my_vec) — метод удаляет все вершины из матрицы

static void deleteVertexMatrixAllSlashText(Vector<Vector<String>> my_vec) — метод удаляет весь текст пометок из матрицы

static void cleanVertexMatrixSlashText(Vector<Vector<String>> my_vec) — метод очищает матрицу пометок

static void copyMatrix(Vector<Vector<Integer>> vec1, Vector<Vector<Integer>> vec2) — метод осуществляет копирование в матрицу vec1 матрицы vec2. Метод ожидает, что vec1 придется инициализировать.

static void changeMatrix(Vector<Vector<Integer>> vec1, Vector<Vector<Integer>> vec2) — метод осуществляет замену элементов в матрице vec1 на элементы матрицы vec2

3.2.5 Основные методы класса TableAdapterFabric

Данный класс отвечает за текстовое поле в интерфейсе программы.

static Vector<String> getVecNames(mxGraph my_g) — метод возвращает вектор с названиями вершин

static int getMaxLengthElementString(Vector<String> my_str_vec, int bool_int) — метод возвращает максимальную длину элементов строки с вершинами

static int getMaxLengthElement(Vector<Vector<Integer>> my_vec) - метод возвращает максимальную длину элементов строки с вершинами

static int getMaxLengthElementFloyd(Vector<Vector<Integer>> my_vec, Vector<Vector<String>> my_vecSlashText) — метод возвращает максимальную длину элемента из матрицы смежности

static void addTableUpdate(Vector<Vector<Integer>> my_vec, JTextArea my_jta, mxGraph my_g) — метод обновляет заголовки таблицы, в которой отображается матрица смежности

static void addTableUpdateFloyd(Vector<Vector<Integer>> my_vec, Vector<Vector<String>> my_vecSlashText, JTextArea my_jta, mxGraph my_g) — метод обновляет таблицу, в которой отображается матрица смежности

static void addTableElement(JTextArea my_jta, String my_str, int l) — метод добавляет элемент в таблицу

static void resetElementTableUpdateFloyd(Vector<Vector<String>> my_vec, String vertexName, int i, int j) — метод сбрасывает элементы таблицы

3.2.6 Основные методы класса **VertexEnumerator**

Данный класс отвечает за нумерацию вершин графа.

String getNextValue() — метод возвращает следующее название вершины

static void Restart() — метод сбрасывает счетчик названий вершин

4. ТЕСТИРОВАНИЕ

4.1. Тестирование алгоритма

Для тестирования алгоритма был написан класс *ReleaseFloydTester*, который содержит юнит-тесты методов класса *ReleaseFloyd*. Тестировались следующие методы:

fillMatrix(Vector <Vector<Integer>>)

run_step(Vector<Vector<Integer>> , mxGraph)

run_all(Vector <Vector<Integer>>)

На *fillMatrix()* был написан один тест, на *run_step()* и *run_all()* по 4 теста. В каждом из них методу подавался граф в виде матрицы смежности, после проверялась сумма элементов матрицы смежности со значением, которое должно было получиться.

4.2. Тестирование итогового проекта

Интерфейс программы тестировался вручную, для этого в визуализаторе прогонялись несколько раз различные графы, для которых было необходимо вычислить кратчайшие пути между вершинами. Попутно проверялся весь функционал программы, например загрузка графа из файла и очистка рабочей области.

ЗАКЛЮЧЕНИЕ

В ходе выполнения проекта была реализована программа, представляющая собой визуализатор алгоритма Флойда-Уоршелла. Для реализации интерфейса использовалась библиотека Swing, а для работы с визуализацией графов — библиотека jgraphx. На момент сдачи практики, все поставленные задачи были выполнены успешно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. <http://java-online.ru/libs-swing.xhtml>
2. <https://stepik.org/course/187/promo>
3. <https://habr.com/ru/post/77382/>
4. Java 8 Полное руководство. Герберт Шилдт.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл Main.java:

```
import com.mxgraph.model.mxCell;
import com.mxgraph.swing.mxGraphComponent;
import com.mxgraph.swing.mxGraphOutline;
import com.mxgraph.view.mxGraph;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Main extends JFrame
{

    private JToggleButton jButtonAddVertex = new
JToggleButton("Add Vertex");
    private JToggleButton jButtonAddEdge = new
JToggleButton("Add Edge");
    private JToggleButton jButtonRename = new
JToggleButton("Rename");
    private JToggleButton jButtonRemove = new
JToggleButton("Remove");
    private JToggleButton jButtonAlgoMode = new
JToggleButton("Algo Mode");
    private JButton jButtonStepUndo = new JButton("Play back");
    private JButton jButtonStep = new JButton("Play one step");
    private JButton jButtonAll = new JButton("Play all");
    static JFrame jFrame;
    private final String[][] menuFile =
        {{"Граф"      , "Ф", "", ""},
```

```

        {"Новый граф" , "O", "O", ""},
        {"Открыть граф из файла", "C", "S", ""}};
private final String[][] menuEdit =
    {"Другое" , "P", "", ""},
    {"Справка" , "B", "X", ""},
    {"Об авторах", "K", "C", ""}};
private final double widthVertex = 80;
private final double heightVertex = 30;
public static JTextArea jFieldTable;
private static final long serialVersionUID = -
2707712944901661771L;
/*=====*/
private JButton btnFileFilter = null;
private JFileChooser fileChooser = null;
private final String[][] FILTERS = {"docx", "Файлы Word
(*.docx)"},
    {"pdf" , "Adobe Reader(*.pdf)"};
/*=====*/
private JMenu createMenuItems(final String[][] items,
mxGraph g, MouseAdapterFabric mouseAdapterFabric)
{
    // Создание выпадающего меню
    JMenu menu = new JMenu(items[0][0]);
    menu.setMnemonic(items[0][1].charAt(0));
    for (int i = 1; i < items.length; i++) {
        // пункт меню "Открыть"
        JMenuItem item = new JMenuItem(items[i][0]);
        if (item.getText() == "Новый граф"){
            item.addMouseListener(new MouseAdapter() {
                @Override
                public void mousePressed(MouseEvent e) {
                    MouseAdapterFabric.deleteAll();

MouseAdapterFabric.deleteAllSlashText();
                    deleteGraph(g);
                }
            });
        }
    }
}

```

```

        }
    });
} else

    if (item.getText() == "Открыть граф из файла"){
        item.addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                fileChooser = new JFileChooser();
                fileChooser.setDialogTitle("Выбор
директории");

                // Определение режима - только каталог

fileChooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTOR
IES);

                int result =
fileChooser.showOpenDialog(Main.this);

                // Если директория выбрана, покажем ее в
сообщении

                if (result ==
JFileChooser.APPROVE_OPTION) {

                    jButtonAlgoMode.setSelected(false);
                    jButtonAddVertex.setEnabled(true);
                    jButtonRename.setEnabled(true);
                    jButtonRemove.setEnabled(true);
                    jButtonAddEdge.setEnabled(true);

                    while
(MouseAdapterFabric.getRunUndo());

                    jButtonStepUndo.setEnabled(false);
                    jButtonStep.setEnabled(false);
                    jButtonAll.setEnabled(false);

```

```

        FileOpening fopening = new
FileOpening(fileChooser.getSelectedFile());
        fopening.ReadingData();
        if(!fopening.getCheck()) {

MouseAdapterFabric.deleteAll();

MouseAdapterFabric.deleteAllSlashText();
        deleteGraph(g);

        int a = fopening.getVertexNum();
        VertexEnumerator
vertexEnumerator = new VertexEnumerator();
        for (int i = 0; i < a; i++) {

mouseAdapterFabric.addVertexMouseAdapterFromFile(vertexEnumer
ator);

        }

mouseAdapterFabric.addEdgeMouseAdapterFromFile(fopening.getMa
trixAdj());

        }
        else {

JOptionPane.showMessageDialog(null, "File is incorrecct",
"Promt", JOptionPane.INFORMATION_MESSAGE);

        }

    }

});
}

if (item.getText() == "Справка"){
    item.addMouseListener(new MouseAdapter() {

```

```

        @Override
        public void mousePressed(MouseEvent e) {
            JOptionPane.showMessageDialog(null,
"Данная программа представляет собой визуализатор\n" +
            "алгоритма Флойда-Уоршелла. Граф
можно ввести двумя путями:" +
            "\nвручную добавив вершины и
рёбра на рабочей области,\n" +
            "с учетом того, что веса ребер
представляют собой целые неотрицательные числа." +
            "\nВторой вариант – записать
матрицу смежности в файл и загрузить его." +
            "\nВ данном случае чтобы
показать отсутствие ребра между вершиной i и j,\nследует ввести
число -1.", "Reference", JOptionPane.INFORMATION_MESSAGE);
        }
    });
}

if (item.getText() == "Об авторах"){
    item.addMouseListener(new MouseAdapter() {
        @Override
        public void mousePressed(MouseEvent e) {
            JOptionPane.showMessageDialog(null,
"Разработчики: Михаил Прокофьев, Борисовский Виктор, Ковалёв
Павел", "Authors", JOptionPane.INFORMATION_MESSAGE);

        }
    });
}

item.setMnemonic(items[i][1].charAt(0)); // русская
буква

// установим клавишу быстрого доступа (латинская
буква)

```

```

        item.setAccelerator(KeyStroke.getKeyStroke(items[i]
[2].charAt(0),
                    KeyEvent.CTRL_MASK));
        if (items[i][3].length() > 0)
            item.setIcon(new ImageIcon(items[i][3]));
        menu.add(item);
    }
    return menu;
}

```

```

private void deleteGraph(mxGraph g){
    mxCell cell;
    for (Object c : g.getChildCells(g.getDefaultParent())){
        cell = (mxCell) c;
        cell.removeFromParent();
        cell.removeFromTerminal(true);
        g.refresh();
    }
}

```

```

private void setAllVertexDefault(mxGraph g){
    mxCell cell;
    for (Object c :
g.getChildVertices(g.getDefaultParent())){
        cell = (mxCell) c;
        cell.setStyle("defaultStyle");
    }
    g.refresh();
}

```

```

public Main()
{
    super("Graph vizualizator");
    mxGraph graph = new mxGraph();
}

```

```

        mxGraphComponent graphComponent = new
mxGraphComponent(graph);

        mxGraphOutline mxGraphOutline = new
mxGraphOutline(graphComponent);

        MouseAdapterFabric mouseAdapterFabric = new
MouseAdapterFabric(graph, mxGraphOutline);


        JMenuBar menuBar = new JMenuBar();
        // Создание меню "Файл"
        menuBar.add(createMenuItems(menuFile, graph,
mouseAdapterFabric));
        // Создание меню "Редактирование"
        menuBar.add(createMenuItems(menuEdit, graph,
mouseAdapterFabric));
        setJMenuBar(menuBar);


        graphComponent.setEnabled(false);


        MouseAdapter moveVertexMouseAdapter =
mouseAdapterFabric.getMoveVertexMouseAdapter();

graphComponent.getGraphControl().addMouseListener(moveVertexMous
eAdapter);

graphComponent.getGraphControl().addMouseMotionListener(moveVert
exMouseAdapter);


        JPanel panelCenter = new JPanel();
        JPanel panelRightRight = new JPanel();
        JPanel panelRightRightDown = new JPanel();
        panelRightRight.setMinimumSize(new Dimension(700, 30));


        BoxLayout boxlayoutVerticalAll = new
BoxLayout(panelCenter, BoxLayout.Y_AXIS);

```

```

        BoxLayout boxlayoutAll = new BoxLayout(getContentPane(),
BoxLayout.X_AXIS);

        BoxLayout boxlayoutRightRight = new
BoxLayout(panelRightRight, BoxLayout.X_AXIS);

        BoxLayout boxlayoutRightRightDown = new
BoxLayout(panelRightRightDown, BoxLayout.X_AXIS);

        panelCenter.setLayout(boxlayoutVerticalAll);
        panelRightRight.setLayout(boxlayoutRightRight);
        panelRightRightDown.setLayout(boxlayoutRightRightDown);
        getContentPane().setLayout(boxlayoutAll);

        // JButton jButtonNewGraph = new JButton("New_Graph");

        jButtonStepUndo.setEnabled(false);
        jButtonStep.setEnabled(false);
        jButtonAll.setEnabled(false);

        JTextArea jTextMatrixSmeshznosti = new
JTextArea("Adjacency matrix:");

        jTextMatrixSmeshznosti.setMaximumSize(new Dimension(130,
20));

        jFieldTable = new JTextArea();
        jFieldTable.setMaximumSize(new Dimension(300, 300));
        jFieldTable.setEditable(false);

        panelRightRight.add(jButtonAddVertex);
        panelRightRight.add(jButtonAddEdge);
        panelRightRight.add(jButtonRename);
        panelRightRight.add(jButtonRemove);

```



```

panelRightRight.add(jButtonAlgoMode);

panelRightRightDown.add(jButtonStepUndo);
panelRightRightDown.add(jButtonStep);
panelRightRightDown.add(jButtonAll);

panelCenter.add(panelRightRight);
panelCenter.add(jTextMatrixSmeshznosti);
panelCenter.add(jFieldTable);
panelCenter.add(panelRightRightDown);

MouseListener addVertexMouseListener =
mouseAdapterFabric.getAddVertexMouseListener();
    JButtonAddVertex.addItemListener(new ItemListener() {
        @Override
        public void itemStateChanged(ItemEvent itemEvent) {

if (itemEvent.getStateChange() == ItemEvent.SELECTED) {
            JButtonRemove.setSelected(false);
            JButtonAddEdge.setSelected(false);
            JButtonRename.setSelected(false);

graphComponent.getGraphControl().addMouseListener(addVertexMouse
Adapter);

                } else
if (itemEvent.getStateChange() == ItemEvent.DESELECTED) {

graphComponent.getGraphControl().removeMouseListener(addVertexMo
useAdapter);

                }
            }
        });

MouseListener deleteCellMouseListener =
mouseAdapterFabric.getDeleteCellMouseListener();

```

```

        jButtonRemove.addItemListener(new ItemListener() {
            @Override
            public void itemStateChanged(ItemEvent itemEvent) {
                if
(itemEvent.getStateChange()==ItemEvent.SELECTED){
                    jButtonAddVertex.setSelected(false);
                    jButtonAddEdge.setSelected(false);
                    jButtonRename.setSelected(false);

graphComponent.getGraphControl().addMouseListener(deleteCellMouse
eAdapter);

                } else
if (itemEvent.getStateChange()==ItemEvent.DESELECTED) {

graphComponent.getGraphControl().removeMouseListener(deleteCellM
ouseAdapter);

                }
            }
        });

        jButtonAddEdge.addItemListener(new ItemListener() {
            MouseAdapter addEdgeMouseAdapter;
            @Override
            public void itemStateChanged(ItemEvent itemEvent) {
                if
(itemEvent.getStateChange()==ItemEvent.SELECTED){
                    jButtonAddVertex.setSelected(false);
                    jButtonRemove.setSelected(false);
                    jButtonRename.setSelected(false);
                    addEdgeMouseAdapter =
mouseAdapterFactory.getAddEdgeMouseAdapter();

graphComponent.getGraphControl().addMouseListener(addEdgeMouseAd
apter);

```

```

        } else
if (itemEvent.getStateChange() == ItemEvent.DESELECTED) {
        setAllVertexDefault(graph);

graphComponent.getGraphControl().removeMouseListener(addEdgeMouseAdapter);
    }
}
});

        MouseAdapter renameVertexMouseAdapter =
mouseAdapterFabric.getRenameVertexMouseAdapter();
        JButtonRename.addItemListener(new ItemListener() {
            @Override
            public void itemStateChanged(ItemEvent itemEvent) {
                if
(itemEvent.getStateChange() == ItemEvent.SELECTED) {
                    JButtonAddVertex.setSelected(false);
                    JButtonRemove.setSelected(false);
                    JButtonAddEdge.setSelected(false);

graphComponent.getGraphControl().addMouseListener(renameVertexMouseAdapter);
                } else
if (itemEvent.getStateChange() == ItemEvent.DESELECTED) {

graphComponent.getGraphControl().removeMouseListener(renameVertexMouseAdapter);
                }
            }
        });

        JButtonStep.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

```

```

        MouseAdapterFabric.getRunStep();
    }
});

jButtonAll.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        MouseAdapterFabric.getRunAll();
        JOptionPane.showMessageDialog(null, "Algorithm
is finish, check output matrix", "Promt",
JOptionPane.INFORMATION_MESSAGE);
    }
});

jButtonStepUndo.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

/*-----*/
        boolean state = MouseAdapterFabric.getRunUndo();
        if (!state){
            JOptionPane.showMessageDialog(null, "You
have reached the first step of the algorithm", "Promt",
JOptionPane.INFORMATION_MESSAGE);
        }

/*-----*/
    }
});
/*-----*/
*/

jButtonAlgoMode.addItemListener(new ItemListener() {
    @Override
    public void itemStateChanged(ItemEvent itemEvent) {

```

```

        if
(itemEvent.getStateChange()==ItemEvent.SELECTED){
            jButtonAddVertex.setEnabled(false);
            jButtonRename.setEnabled(false);
            jButtonRemove.setEnabled(false);
            jButtonAddEdge.setEnabled(false);

            jButtonStepUndo.setEnabled(true);
            jButtonStep.setEnabled(true);
            jButtonAll.setEnabled(true);
        } else if
(itemEvent.getStateChange()==ItemEvent.DESELECTED){
            jButtonAddVertex.setEnabled(true);
            jButtonRename.setEnabled(true);
            jButtonRemove.setEnabled(true);
            jButtonAddEdge.setEnabled(true);

            while (MouseAdapterFabric.getRunUndo());
            jButtonStepUndo.setEnabled(false);
            jButtonStep.setEnabled(false);
            jButtonAll.setEnabled(false);
        }
    }
});

```

```

/*-----
-----*/

```

```

        getContentPane().add(graphComponent);
        getContentPane().add(panelCenter);
    }

```

```

public static void main(String[] args)
{
    JFrame = getFrame();
}

```

```

static JFrame getFrame(){
    Main frame = new Main();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Toolkit toolkit = Toolkit.getDefaultToolkit();
    Dimension dimension = toolkit.getScreenSize();
    frame.setBounds(dimension.width / 2 - 375,
dimension.height / 2 - 250, 750, 500);
    frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
    //frame.setUndecorated(true);
    frame.setVisible(true);
    return frame;
}

}

```

Файл FileOpening.java:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.Vector;

public class FileOpening {
    private File f = null;
    private boolean incorrect = false;
    private Vector<Vector<Integer>> matrixAdj = new
Vector<Vector<Integer>>();

    private int vertexNum = 0;

    public FileOpening(File df){
        f = df;
    }

    public void ReadingData(){
        try(FileReader reader = new FileReader(f))
        {
            Scanner scanner = new Scanner(f);
            try {
                int n = scanner.nextInt();
                vertexNum = n;
                for (int i = 0; i < n; i++) {
                    matrixAdj.addElement(new Vector<Integer>());
                    for (int j = 0; j < n; j++) {
                        int scanTmp = scanner.nextInt();
                        if(scanTmp<0)
                            if(scanTmp != -1)
                                incorrect = true;
                    }
                }
            }
        }
    }
}
```

```

        else
            scanTmp = 100000000;
            matrixAdj.get(i).addElement(scanTmp);
        }
    }
}

catch(InputMismatchException e){
    incorrect = true;
}

}

catch(IOException ex){
    System.out.println(ex.getMessage());
}

}

public int getVertexNum(){
    return vertexNum;
}

public boolean getCheck() {
    return incorrect;
}

public Vector<Vector<Integer>> getMatrixAdj(){
    return matrixAdj;
}

}

```


Файл MatrixAdapterFabric.java:

```
import java.util.Vector;

public class MatrixAdapterFabric {

    public static void addVertexMatrix(Vector<Vector<Integer>>
my_vec) {
        if(my_vec.size() == 0) {
            my_vec.addElement(new Vector<Integer>());
            my_vec.get(0).addElement(new Integer(0));
        }else {
            my_vec.addElement(new Vector<Integer>());
            int i = 0;
            for(; i < my_vec.size()-1; i++) {
                my_vec.get(i).addElement(new Integer(-1));
            }
            for(int j = 0; j < my_vec.size(); j++) {
                if(i!=j)
                    my_vec.get(i).addElement(new Integer(-1));
                else
                    my_vec.get(i).addElement(new Integer(0));
            }
        }
    }

    public static void
addVertexMatrixSlashText(Vector<Vector<String>> my_vec) {
        if (my_vec.size() == 0) {
            my_vec.addElement(new Vector<String>());
            my_vec.get(0).addElement(new String(""));
        } else {
            my_vec.addElement(new Vector<String>());
            int i = 0;
            for (; i < my_vec.size() - 1; i++) {
                my_vec.get(i).addElement(new String(""));
            }
        }
    }
}
```

```

        }
        for (int j = 0; j < my_vec.size(); j++) {
            my_vec.get(i).addElement(new String(""));
        }
    }
}

    public static void addEdgeMatrix(Vector<Vector<Integer>>
my_vec, int i, int j, int value) {
        my_vec.get(i).set(j, value);
    }

    public static void
deleteVertexMatrix(Vector<Vector<Integer>> my_vec, int number) {
        int i = 0;
        for(; i < number; i++) {
            my_vec.get(i).remove(number);
        }
        my_vec.remove(number);
        int loc_size = my_vec.size() - 1;
        for(; i < loc_size; i++) {
            my_vec.get(i).remove(number);
        }
        if(my_vec.size()!=0)
            my_vec.get(my_vec.size() - 1).set(my_vec.size() -
1,0);
    }

    public static void
deleteVertexMatrixSlashText(Vector<Vector<String>> my_vec, int
number) {
        int i = 0;
        for(; i < number; i++) {

```

```

        my_vec.get(i).remove(number);
    }
    my_vec.remove(number);
    int loc_size = my_vec.size() - 1;
    for(; i < loc_size; i++) {
        my_vec.get(i).remove(number);
    }
    if(my_vec.size() != 0)
        my_vec.get(my_vec.size() - 1).set(my_vec.size() -
1, "");
    }

    public static void deleteEdgeMatrix(Vector<Vector<Integer>>
my_vec, int i, int j) {
        my_vec.get(i).set(j, 100000000);
    }

    public static void
deleteVertexMatrixAll(Vector<Vector<Integer>> my_vec) {
        int n = my_vec.size();
        for(int i = 0; i < n; i++){
            my_vec.remove(0);
        }
    }

    public static void
deleteVertexMatrixAllSlashText(Vector<Vector<String>> my_vec) {
        int n = my_vec.size();
        for(int i = 0; i < n; i++){
            my_vec.remove(0);
        }
    }

    public static void
cleanVertexMatrixSlashText(Vector<Vector<String>> my_vec) {

```

```

        for(int i = 0; i < my_vec.size(); i++) {
            for(int j = 0; j < my_vec.size(); j++) {
                my_vec.get(i).set(j, "");
            }
        }
    }

    public static void copyMatrix(Vector <Vector<Integer>> vec1,
    Vector <Vector<Integer>> vec2){
        for(int i = 0; i < vec2.size(); i++){
            vec1.addElement(new Vector<Integer>());
            for(int j = 0; j < vec2.size(); j++){
                vec1.get(i).addElement(vec2.get(i).get(j));
            }
        }
    }

    public static void changeMatrix(Vector <Vector<Integer>>
    vec1, Vector <Vector<Integer>> vec2){
        for(int i = 0; i < vec2.size(); i++) {
            for (int j = 0; j < vec2.size(); j++) {
                vec1.get(i).set(j, vec2.get(i).get(j));
            }
        }
    }
}

```

Файл MouseAdapterFabric.java:

```
import com.mxgraph.layout.mxCircleLayout;
import com.mxgraph.layout.mxParallelEdgeLayout;
import com.mxgraph.model.mxCell;
import com.mxgraph.model.mxGeometry;
import com.mxgraph.model.mxICell;
import com.mxgraph.swing.mxGraphOutline;
import com.mxgraph.view.mxGraph;

import javax.swing.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.Scanner;
import java.util.Vector;

public class MouseAdapterFabric {
    private static Integer indexVertex;
    private static Integer indexEdge;
    private final double widthVertex = 80;
    private final double heightVertex = 30;
    private static mxGraph graph;
    private mxGraphOutline graphOutline;
    private static Vector<Vector<Integer>> adj;
    private static Vector<Vector<String>> adjSlashText;

    MouseAdapterFabric(mxGraph g, mxGraphOutline gOutline){
        graph = g;
        graphOutline = gOutline;
        adj = new Vector<Vector<Integer>>();
        adjSlashText = new Vector<Vector<String>>();
        indexVertex = -1;
        indexEdge = -1;
    }
}
```

```

public MouseAdapter getAddEdgeMouseAdapter(){
    MouseAdapter addEdgeMouseAdapter = new MouseAdapter() {
        mxCell firstCell = null;
        mxCell secondCell = null;
        int firstCellId = 0;
        int secondCellId = 0;

        private void createEdge(JTextField textWeight, int
weight){
            graph.getModel().beginUpdate(); // начали
ОБНОВЛЯТЬ
            try
            {
                firstCell.setStyle("defaultStyle;");
                indexEdge++;
                MatrixAdapterFabric.addEdgeMatrix(adj,
firstCellId, secondCellId, weight);

TableAdapterFabric.addTableUpdate(adj,Main.jFieldTable, graph);
                graph.insertEdge(graph.getDefaultParent(),
indexEdge.toString()+"e", textWeight.getText(), firstCell,
secondCell);

                new
mxParallelEdgeLayout(graph).execute(graph.getDefaultParent());
                firstCell = null;
                secondCell = null;
                graph.refresh();
            }
            finally
            {
                graph.getModel().endUpdate(); // закончили
            }
        }
        private void chooseFirstVertex(MouseEvent e){

```

```

        firstCell = (mxCell)
graphOutline.getGraphComponent().getCellAt(e.getX(), e.getY());

        if (firstCell != null){

            if (firstCell.isVertex()){
                StringBuilder time_string = new
StringBuilder( firstCell.getId());

time_string.deleteCharAt(time_string.length()-1);
                firstCellId =
Integer.parseInt(time_string.toString());
                //System.out.println(firstCellId + ": 1
vert");

                graph.getModel().beginUpdate(); //
начали обновлять

                try
                {

firstCell.setStyle("fillColor=green");

                graph.refresh();

                }
                finally
                {

                    graph.getModel().endUpdate(); //
закончили

                }

            } else {
                firstCell = null;
            }
        }
    }

    public void chooseSecondVertex(MouseEvent e){
        secondCell = (mxCell)
graphOutline.getGraphComponent().getCellAt(e.getX(), e.getY());

```

```

        if (secondCell == null){
            resetSelection();
        } else {
            if (secondCell.isVertex()){
                StringBuilder time_string = new
StringBuilder( secondCell.getId());

time_string.deleteCharAt(time_string.length()-1);
                secondCellId =
Integer.parseInt(time_string.toString());
                //System.out.println(secondCellId + ": 2
vert");

                if (connectionVertex(firstCell,
secondCell)){

                    resetSelection();
                    JOptionPane.showMessageDialog(null,
"These vertices are already connected!", "Invalid selection of
vertices", JOptionPane.ERROR_MESSAGE);
                } else {
                    tryCreateEdge();

                }
            }
        }
    }

private void resetSelection(){
    graph.getModel().beginUpdate(); // начали
ОБНОВЛЯТЬ

    try
    {
        firstCell.setStyle("defaultStyle;");
        firstCell = null;
        graph.refresh();
    }
}

```



```

        finally
        {
            graph.getModel().endUpdate(); // закончили
        }
    }

    public void tryCreateEdge(){
        JTextField edgeWeight = new JTextField();
        final JComponent[] inputs = new JComponent[] {
            new JLabel("Enter weight (Positive
integer)"),
            edgeWeight
        };
        int result = JOptionPane.showConfirmDialog(null,
inputs, "Add edge", JOptionPane.PLAIN_MESSAGE);
        if (result == JOptionPane.OK_OPTION) {
            Scanner sc = new
Scanner(edgeWeight.getText().trim());
            if (!sc.hasNextInt()){
                resetSelection();
                JOptionPane.showMessageDialog(null,
"You entered not an integer!", "Invalid weight input",
JOptionPane.ERROR_MESSAGE);

            } else {
                int weight = sc.nextInt();
                if (sc.hasNext()){
                    resetSelection();
                    JOptionPane.showMessageDialog(null,
"Wrong number of arguments!", "Invalid weight input",
JOptionPane.ERROR_MESSAGE);
                } else if (weight < 0){
                    resetSelection();
                }
            }
        }
    }
}

```

```

        JOptionPane.showMessageDialog(null,
"The number must be positive!", "Invalid weight input",
JOptionPane.ERROR_MESSAGE);
    } else {
        createEdge(edgeWeight, weight);
    }
}
}
}

@Override
public void mouseClicked(MouseEvent e) {
    if (firstCell == null){
        chooseFirstVertex(e);
    } else {
        chooseSecondVertex(e);
    }
}

};
return addEdgeMouseAdapter;
}

```

```

public void
getAddEdgeMouseAdapterFromFile(Vector<Vector<Integer>> my_vec) {
    // System.out.println(my_vec);
    // System.out.println(adj);
    MatrixAdapterFabric.changeMatrix(adj, my_vec);
    mxCell timeCell1, timeCell2;
    for(int i = 0; i < adj.size(); i++) {
        for(int j = 0; j < adj.size(); j++) {
            for (Object c :
graph.getChildVertices(graph.getDefaultParent())) {
                timeCell1 = (mxCell) c;
                for (Object e :
graph.getChildVertices(graph.getDefaultParent())) {

```



```

        VertexEnumerator vertexEnumerator = new
VertexEnumerator();
        MouseAdapter addVertexMouseAdapter = new MouseAdapter()
{
    @Override
    public void mouseClicked(MouseEvent e) {
        graph.getModel().beginUpdate(); // начали
ОБНОВЛЯТЬ
        try
        {
            indexVertex++;
            // System.out.println((indexVertex-1)+ " + 1
= " + indexVertex + "; " + indexVertex + " + v =" + indexVertex
+ "v");

            Object v2 =
graph.insertVertex(graph.getDefaultParent(), null,
vertexEnumerator.getNextValue(), e.getX() - widthVertex / 2,
e.getY() - heightVertex / 2, widthVertex, heightVertex);
            mxCell test = (mxCell) v2;
            test.setId(indexVertex.toString()+"v");
            // System.out.println(test.getId()+ " create
vert");

            MatrixAdapterFabric.addVertexMatrix(adj);

MatrixAdapterFabric.addVertexMatrixSlashText(adjSlashText);

TableAdapterFabric.addTableUpdate(adj,Main.jFieldTable, graph);
        }
        finally
        {
            graph.getModel().endUpdate(); // закончили
        }
    }
};
return addVertexMouseAdapter;

```

```

    }

    public void
getAddVertexMouseAdapterFromFile(VertexEnumerator
vertexEnumerator){
    graph.getModel().beginUpdate(); // начали обновлять
    try {
        indexVertex++;
        // System.out.println((indexVertex-1)+ " + 1 = " +
indexVertex + "; " + indexVertex + " + v =" + indexVertex +
"v");

        Object v2 =
graph.insertVertex(graph.getDefaultParent(), null,
vertexEnumerator.getNextValue(), 0, 0, widthVertex,
heightVertex);

        new
mxCircleLayout(graph).execute(graph.getDefaultParent());

        mxCell test = (mxCell) v2;
        test.setId(indexVertex.toString() + "v");
        //System.out.println(test.getId()+ " create vert");
        MatrixAdapterFabric.addVertexMatrix(adj);

MatrixAdapterFabric.addVertexMatrixSlashText(adjSlashText);

        TableAdapterFabric.addTableUpdate(adj,
Main.jFieldTable, graph);
    }finally{
        graph.getModel().endUpdate(); // закончили
    }
}

    public MouseAdapter getRenameVertexMouseAdapter(){
        MouseAdapter renameVertexMouseAdapter = new
MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {

```

```

        mxCell cell = (mxCell)
graphOutline.getGraphComponent().getCellAt(e.getX(), e.getY());
        if (cell != null){
            if (cell.isVertex()){
                JTextField changeName = new
JTextField(cell.getValue().toString());
                final JComponent[] inputs = new
JComponent[] {
                    new JLabel("Enter new name"),
                    changeName
                };
                int result =
JOptionPane.showConfirmDialog(null, inputs, "Change name",
JOptionPane.PLAIN_MESSAGE);
                if (result == JOptionPane.OK_OPTION) {
                    graph.getModel().beginUpdate(); //
начали обновлять
                    try
                    {
                        graph.getModel().setValue(cell,
changeName.getText());
                        graph.refresh();

JTableAdapterFabric.addTableUpdate(adj, Main.jFieldTable, graph);
                    }
                    finally
                    {
                        graph.getModel().endUpdate(); //
закончили
                    }
                }
            }
        }
    }
};

```

```

        return renameVertexMouseAdapter;
    }

    public MouseAdapter getDeleteCellMouseAdapter() {
        MouseAdapter deleteCellMouseAdapter = new MouseAdapter()
        {

            @Override
            public void mouseClicked(MouseEvent e) {
                int cellId = 0;
                mxCell cell = (mxCell)
graphOutline.getGraphComponent().getCellAt(e.getX(), e.getY());
                if (cell != null) {
                    if (cell.isVertex()) {
                        StringBuilder time_string = new
StringBuilder( cell.getId());

time_string.deleteCharAt(time_string.length()-1);
                        cellId =
Integer.parseInt(time_string.toString());
                        graph.getModel().beginUpdate(); //
начали обновлять

                        try
                        {

MatrixAdapterFabric.deleteVertexMatrix(adj, cellId);

MatrixAdapterFabric.deleteVertexMatrixSlashText(adjSlashText, cel
lId);

                        cell.setId("222");
                        cell.removeFromParent();
                        cell.removeFromTerminal(true);

TableAdapterFabric.addTableUpdate(adj, Main.jFieldTable, graph);
                        mxCell timeCell;

```

```

        //System.out.println(cellId +
"::contr");

        for (Object c :
graph.getChildVertices(graph.getDefaultParent())){
            timeCell = (mxCell) c;
            int timeCellId = 1;
            StringBuilder
time_stringTimeCellId = new StringBuilder( timeCell.getId());

time_stringTimeCellId.deleteCharAt(time_stringTimeCellId.length(
)-1);

            timeCellId =
Integer.parseInt(time_stringTimeCellId.toString());
            if(timeCellId>cellId) {

//System.out.println(timeCellId + ">" + cellId);

                Integer renameCellId =
timeCellId - 1;

timeCell.setId(renameCellId.toString() + "v");

            }

//System.out.println(timeCell.getId());

            }
            indexVertex--;
            //System.out.println("delete vert: "
+ cell.getId() +" indexVertex stal: " + indexVertex);
            graph.refresh();
        }
        finally
        {
            graph.getModel().endUpdate(); //
закончили

        }
    }
}

```



```

else {
    graph.getModel().beginUpdate(); //
начали обновлять
    try
    {
        mxCell timeFirstCell = (mxCell)
cell.getSource();
        mxCell timeSecondCell = (mxCell)
cell.getTarget();

        int firstCellId;
        int secondCellId;
        StringBuilder time_string = new
StringBuilder( timeFirstCell.getId());
        StringBuilder time_string2 = new
StringBuilder( timeSecondCell.getId());

time_string.deleteCharAt(time_string.length()-1);

time_string2.deleteCharAt(time_string2.length()-1);
        firstCellId =
Integer.parseInt(time_string.toString());
        secondCellId =
Integer.parseInt(time_string2.toString());

MatrixAdapterFabric.deleteEdgeMatrix(adj, firstCellId,
secondCellId);

        cell.removeFromParent();
        cell.removeFromTerminal(true);

TableAdapterFabric.addTableUpdate(adj, Main.jFieldTable, graph);
        graph.refresh();
    }
    finally
    {

```

```

graph.getModel().endUpdate(); //

закончили

        }
    }

    }

};
return deleteCellMouseAdapter;
}

public MouseAdapter getMoveVertexMouseAdapter() {
    MouseAdapter moveVertexMouseAdapter = new MouseAdapter()
{
    int distanceX, distanceY;
    boolean moves = false;
    double width, height;
    mxCell cell;

    @Override
    public void mouseDragged(MouseEvent e) {
        if (moves){
            mxGeometry geometry = cell.getGeometry();
            if (cell.isVertex()){
                graph.getModel().beginUpdate();
                graph.getModel().setGeometry(cell, new
mxGeometry(geometry.getX() + (e.getX() - distanceX),
geometry.getY() + (e.getY() - distanceY), width, height));
                distanceX = e.getX();
                distanceY = e.getY();
                graph.getModel().endUpdate();
            }
        }
    }
}
}

```

```

        @Override
        public void mousePressed(MouseEvent e) {
            cell = (mxCell)
graphOutline.getGraphComponent().getCellAt(e.getX(), e.getY());
            if (cell != null){
                mxGeometry geometry = cell.getGeometry();
                width = geometry.getWidth();
                height = geometry.getHeight();
                distanceX = e.getX();
                distanceY = e.getY();
                moves = true;
            }
        }

        @Override
        public void mouseReleased(MouseEvent e) {
            distanceX = 0;
            distanceY = 0;
            moves = false;
        }
    };
    return moveVertexMouseAdapter;
}

private boolean connectionVertex(mxCell cell1, mxCell cell2)
{
    boolean check = false;
    if (cell1 == cell2){
        for (int i = 0; i < cell1.getEdgeCount(); i++){
            mxICell source = ((mxCell)
cell1.getEdgeAt(i)).getSource();
            mxICell target = ((mxCell)
cell1.getEdgeAt(i)).getTarget();
            if (cell1 == target && cell1 == source){
                check = true;
            }
        }
    }
}

```

```

        }
    }
    } else {
        for (int i = 0; i < cell1.getEdgeCount(); i++) {
            mxICell target = ((mxCell)
cell1.getEdgeAt(i)).getTarget();
            if (target == cell2) {
                check = true;
            }
        }
    }
    return check;
}

public static void getRunStep() {
    ReleaseFloyd.run_step(adj, adjSlashText, graph);
}

public static boolean getRunUndo() {
    /*-----*/
    //пофиксить выход за границу
    Vector<Vector<Integer>> tmp = ReleaseFloyd.getState();
    if(tmp != null){
        adj = tmp;
        //System.out.println(ReleaseFloyd.getState());

        TableAdapterFabric.addTableUpdate(adj,Main.jFieldTable, graph);
        return true;
    } else{
        return false;
    }

    /*-----*/
}

```

```

public static void getRunAll() {
    ReleaseFloyd.run_all(adj, adjSlashText, graph);
}

public static void getDeleteAll() {
    indexVertex = -1;
    indexEdge = -1;
    VertexEnumerator.Restart();
    MatrixAdapterFabric.deleteVertexMatrixAll(adj);
    TableAdapterFabric.addTableUpdate(adj, Main.jFieldTable,
graph);
}

public static void getDeleteAllSlashText() {
    indexVertex = -1;
    indexEdge = -1;
    VertexEnumerator.Restart();

MatrixAdapterFabric.deleteVertexMatrixAllSlashText(adjSlashText)
;
    TableAdapterFabric.addTableUpdate(adj, Main.jFieldTable,
graph);
}
}

```

Файл ReleaseFloyd.java:

```
import com.mxgraph.model.mxCell;
import com.mxgraph.view.mxGraph;

import javax.swing.*;
import java.awt.*;
import java.util.*;

public class ReleaseFloyd {

    private static int k_step = 0;
    private static int i_step = 0;

    private static ArrayList<Vector<Vector<Integer>>> states =
new ArrayList <Vector<Vector<Integer>>>();

    public static Vector<Vector<Integer>> getState(){
        if(states.size() != 0){
            Vector<Vector<Integer>> st =
states.get(states.size() - 1);
            states.remove(states.size() - 1);
            if(k_step > 0){
                if(i_step > 0){
                    i_step--;
                }else{
                    i_step = st.size() - 1;
                    k_step--;
                }
            }
            return st;
        }
        return null;
    }

    public static void fillMatrix(Vector <Vector<Integer>>
my_vec) {
```

```

int n = my_vec.size();
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        if(my_vec.get(i).get(j) == -1){
            my_vec.get(i).set(j, 100000000);
        }
    }
}
}

```

```

public static void run_step(Vector <Vector<Integer>> my_vec,
Vector<Vector<String>> my_vecSlashText, mxGraph graph) {
    int n = my_vec.get(0).size();
    fillMatrix(my_vec);
    RecolorVertexDefault(graph);
    Vector <Vector<Integer>> tmp = new Vector
<Vector<Integer>>();
    MatrixAdapterFabric.copyMatrix(tmp, my_vec);
    states.add(tmp);
    //System.out.println(states);
    if(k_step<n) {
        if(i_step<n) {
            MakeStep(my_vec, my_vecSlashText, graph, n);
            ++i_step;
        }else {
            i_step = 0;
            ++k_step;
            if(k_step < n) {
                MakeStep(my_vec, my_vecSlashText, graph, n);
            }
            ++i_step;
        }
    }
    if(k_step == n){

```

```

        JOptionPane.showMessageDialog(null, "Algorithm is
finish, check output matrix", "Promt",
JOptionPane.INFORMATION_MESSAGE);
        //System.out.println("End of algo");
    }else {
        //System.out.println("srabotalo rs");
    }
}

    public static void run_all(Vector <Vector<Integer>> my_vec,
Vector<Vector<String>> my_vecSlashText, mxGraph graph) {
        int n = my_vec.get(0).size();
        fillMatrix(my_vec);
        Vector <Vector<Integer>> tmp = new Vector
<Vector<Integer>>();
        MatrixAdapterFabric.copyMatrix(tmp, my_vec);
        states.add(tmp);
        for (int k=0; k<n; ++k) {
            for (int i = 0; i < n; ++i)
                for (int j = 0; j < n; ++j)

MakeStepAll(my_vec,my_vecSlashText,graph,i,j,k);
        }
        TableAdapterFabric.addTableUpdateFloyd(my_vec,
my_vecSlashText, Main.jFieldTable, graph);
    }

    private static void MakeStep(Vector<Vector<Integer>> my_vec,
Vector<Vector<String>> my_vecSlashText, mxGraph graph, int n) {
        for(int j_step=0; j_step<n; ++j_step) {
            int tmp1 = my_vec.get(i_step).get(j_step);
            int tmp2 = Math.min(my_vec.get(i_step).get(j_step),
my_vec.get(i_step).get(k_step) +
my_vec.get(k_step).get(j_step));

```



```

        //System.out.println("collision: " + tmp1 + " and "
+ tmp2);

        if(tmp1!=tmp2) {
            my_vec.get(i_step).set(j_step, tmp2);
            my_vecSlashText.get(i_step).set(j_step,
TableAdapterFabric.getVecNames(graph).get(k_step));
            TableAdapterFabric.addTableUpdateFloyd(my_vec,
my_vecSlashText, Main.jFieldTable, graph);
            RecolorVertex(graph, i_step, j_step, k_step);
        }
    }
}

```

```

private static void MakeStepAll(Vector<Vector<Integer>>
my_vec, Vector<Vector<String>> my_vecSlashText, mxGraph
graph ,int i_step, int j_step, int k_step) {
    int tmp1 = my_vec.get(i_step).get(j_step);
    int tmp2 = Math.min(my_vec.get(i_step).get(j_step),
my_vec.get(i_step).get(k_step) +
my_vec.get(k_step).get(j_step));
    if(tmp1!=tmp2) {
        my_vec.get(i_step).set(j_step, tmp2);
        my_vecSlashText.get(i_step).set(j_step,
TableAdapterFabric.getVecNames(graph).get(k_step));
    }
}

```

```

private static void RecolorVertexDefault(mxGraph graph) {
    for (Object c :
graph.getChildVertices(graph.getDefaultParent())) {
        mxCell timeCell;
        timeCell = (mxCell) c;
        timeCell.setStyle("defaultStyle");
    }
    graph.refresh();
}

```

```

    }

    private static void RecolorVertex(mxGraph graph, int i_step,
int j_step, int k_step){

        for (Object c :
graph.getChildVertices(graph.getDefaultParent())) {
            mxCell timeCell;
            int timeCellId;
            timeCell = (mxCell) c;
            StringBuilder time_string = new
StringBuilder( timeCell.getId());
            time_string.deleteCharAt(time_string.length()-1);
            timeCellId =
Integer.parseInt(time_string.toString());
            //System.out.println(timeCellId + ": " + i_step+ ",
" + i_step+ ", " + k_step+ ", ");
            if(i_step == timeCellId) {
                //System.out.println("+");
                timeCell.setStyle("fillColor=green");
            }
            if(j_step == timeCellId) {
                //System.out.println("++");
                timeCell.setStyle("fillColor=green");
            }
            if(k_step == timeCellId) {
                //System.out.println("+++");
                timeCell.setStyle("fillColor=red");
            }
        }

        graph.refresh();
    }

    public static void algo(Vector <Vector<Integer>> my_vec) {

```

```

        int n = my_vec.get(0).size();
        fillMatrix(my_vec);
        Vector <Vector<Integer>> tmp = new Vector
<Vector<Integer>>();
        MatrixAdapterFabric.copyMatrix(tmp, my_vec);
        for (int k=0; k<n; ++k) {
            for (int i = 0; i < n; ++i)
                for (int j = 0; j < n; ++j)
                    my_vec.get(i).set(j,
Math.min(my_vec.get(i).get(j), my_vec.get(i).get(k) +
my_vec.get(k).get(j)));
        }
    }
}

```

Файл TableAdapterFabric.java:

```
import com.mxgraph.model.mxCell;
import com.mxgraph.view.mxGraph;

import java.util.Vector;
import javax.swing.*.*;

public class TableAdapterFabric {
    public static Vector<String> getVecNames(mxGraph my_g) {
        Vector<String> out_vec = new Vector<String>();
        mxCell timeCell;
        for (Object c :
my_g.getChildVertices(my_g.getDefaultParent())) {
            timeCell = (mxCell) c;
            String time_string = (String)
my_g.getModel().getValue(timeCell);
            //System.out.println(time_string);
            out_vec.addElement(time_string);
        }
        // System.out.println(out_vec);
        return out_vec;
    }

    //Для названий вершин

    public static int getMaxLengthElementString(Vector<String>
my_str_vec, int bool_int) {
        int out_int = bool_int;
        for(int i = 0; i < my_str_vec.size(); i++) {
            if(my_str_vec.get(i).length()>bool_int) {
                out_int = my_str_vec.get(i).length();
            }
        }
    }
}
```

```

        return out_int;
    }

//Для элементов

    public static int
getMaxLengthElement(Vector<Vector<Integer>> my_vec) {
    Integer max = 0;
    for(int i = 0; i < my_vec.size(); i++) {
        for (int j = 0; j < my_vec.size(); j++) {
            if((my_vec.get(i).get(j) <
1000000000)&&(my_vec.get(i).get(j) > max))
                max = my_vec.get(i).get(j);
        }
    }
    String time_string = max.toString();
    return time_string.length();
}

    public static int
getMaxLengthElementFloyd(Vector<Vector<Integer>> my_vec,
Vector<Vector<String>> my_vecSlashText) {
    Integer max = 0;
    for(int i = 0; i < my_vec.size(); i++) {
        for (int j = 0; j < my_vec.size(); j++) {
            int my_vec_l =
my_vec.get(i).get(j).toString().length();
            int my_vecSlashText_l =
my_vecSlashText.get(i).get(j).length();
            int tmp = my_vec_l + my_vecSlashText_l + 1;
            if(tmp > max)
                max = tmp;
        }
    }
    String time_string = max.toString();

```

```

        return time_string.length();
    }

//Для обновления таблицы

    public static void addTableUpdate(Vector<Vector<Integer>>
my_vec, JTextArea my_jta, mxGraph my_g) {
        my_jta.setText(null);
        Vector<String> str_vec = getVecNames(my_g);
        int l_loc;

        l_loc = getMaxLengthElement(my_vec);
        l_loc = getMaxLengthElementString(str_vec, l_loc);

        addTableElement(my_jta, "", l_loc);
        for(int i = 0; i < my_vec.size(); i++) {
            addTableElement(my_jta, str_vec.get(i), l_loc);
        }
        my_jta.append("\n");
        for(int i = 0; i < my_vec.size(); i++) {
            addTableElement(my_jta, str_vec.get(i), l_loc);
            for(int j = 0; j < my_vec.size(); j++) {
                if((my_vec.get(i).get(j) == 100000000)||
(my_vec.get(i).get(j) == -1))
                    addTableElement(my_jta, "N", l_loc);
                else
                    addTableElement(my_jta,
my_vec.get(i).get(j).toString(), l_loc);
                //my_jta.append(my_vec.get(i).get(j).toString()
+ " ");
            }
            my_jta.append("\n");
        }
    }
}

```

```

        public static void
addTableUpdateFloyd(Vector<Vector<Integer>> my_vec,
Vector<Vector<String>> my_vecSlashText, JTextArea my_jta,
mxGraph my_g) {
    my_jta.setText(null);
    Vector<String> str_vec = getVecNames(my_g);
    int l_loc;

    l_loc = getMaxLengthElementFloyd(my_vec,
my_vecSlashText);
    l_loc = getMaxLengthElementString(str_vec, l_loc);

    addTableElement(my_jta, "", l_loc);
    for(int i = 0; i < my_vec.size(); i++) {
        addTableElement(my_jta, str_vec.get(i), l_loc);
    }
    my_jta.append("\n");
    for(int i = 0; i < my_vec.size(); i++) {
        addTableElement(my_jta, str_vec.get(i), l_loc);
        for(int j = 0; j < my_vec.size(); j++) {
            if((my_vec.get(i).get(j) == 100000000)||
(my_vec.get(i).get(j) == -1))
                addTableElement(my_jta, "N", l_loc);
            else {
                if (my_vecSlashText.get(i).get(j).length() >
0) {
                    addTableElement(my_jta,
my_vec.get(i).get(j).toString() + "/" +
my_vecSlashText.get(i).get(j), l_loc);
                }
                else {
                    addTableElement(my_jta,
my_vec.get(i).get(j).toString(), l_loc);
                }
            }
        }
    }
}

```

```

        }
        my_jta.append("\n");
    }
}

//Для печати

    public static void addTableElement(JTextArea my_jta, String
my_str, int l) {
        int insert_l = my_str.length();
        my_jta.append(my_str);
        for(; insert_l<l; insert_l++) {
            my_jta.append(" ");
        }
        my_jta.append(" ");
    }

//Для Флойда

    public static void
resetElementTableUpdateFloyd(Vector<Vector<String>> my_vec,
String vertexName, int i, int j) {
        my_vec.get(i).set(j, vertexName);
    }

}

```


Файл VertexEnumerator.java:

```
public class VertexEnumerator {
    private static Character prefix;
    private static String currentString;
    private static Character alphabetPosition;

    VertexEnumerator() {
        alphabetPosition = 65;
        prefix = 65;
        currentString = "";
    }

    public String getNextValue() {
        String value;
        if (alphabetPosition < 91) {
            value = currentString + alphabetPosition.toString();
            alphabetPosition++;
            return value;
        } else {
            alphabetPosition = 65;
            currentString += prefix;
            prefix++;
            value = currentString + alphabetPosition.toString();
            alphabetPosition++;
            return value;
        }
    }

    public static void Restart() {
        alphabetPosition = 65;
        prefix = 65;
        currentString = "";
    }
}
```