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

ОТЧЁТ

по учебной практике

Тема: Визуализация работы алгоритм Краскала

Студент гр. 9383 Хотяков Е.П.

Студентка гр. 9383 Чебесова И.Д.

Студентка гр. 9383 _____ Лапина А.А.

Преподаватель _____ Фиалковский М.С.

Санкт-Петербург

2021

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Хотяков Е. П. группы 9383

Студентка Чебесова И.Д. группы 9383

Студент Лапина А.А. группы 9383

Тема практики: Визуализация работы алгоритм Краскала

Задание на практику:

Командная итеративная разработка визуализации алгоритма на Java с графическим интерфейсом.

Алгоритм: Краскала.

Сроки прохождения практики: 01.07.2021 – 11.07.2021

Дата сдачи отчета: 06.07.2021

Дата защиты отчета: 06.07.2021

Студент гр. 9383		Хотяков Е.П.
Студентка гр. 9383		Чебесова И.Д.
Студентка гр. 9383	_____	Лапина А.А.
Преподаватель	_____	Фиалковский М.С.

АННОТАЦИЯ

Основной целью работы является получение навыков программирования на языке Java и освоение парадигмы объектно-ориентированного программирования. Цель достигается командной работой путём разработки программы с графическим интерфейсом, использующей указанные в задании алгоритмы. В этой работе необходимо реализовать алгоритм, находящий минимальное остовное дерево. Пользователь задает количество ребер и вершин в графе, а затем и сами ребра с весом. Для проверки корректности программы используются JUnit тесты.

SUMMARY

The main goal of the work is to acquire programming skills in the Java language and master the paradigm of object-oriented programming. The goal is achieved by teamwork by developing a program with a graphical interface using the algorithms specified in the task. In this work, it is necessary to implement an algorithm that finds the minimum spanning tree. The user sets the number of edges and vertices in the graph, and then the edges themselves with weight. JUnit tests are used to check the correctness of the program.

СОДЕРЖАНИЕ

	Введение	4
1.	Требования к программе	5
1.1.	Описание задачи	7
1.2.	Интерфейс	7
1.3.	Входные данные	10
1.4.	Выходные данные	10
2	План разработки и распределение ролей в бригаде	11
2.1.	План разработки	11
2.2.	Распределение ролей в бригаде	11
3.	Функции и структуры программы	12
3.1.	*class* EnterData	12
3.2.	*class* Kruscal	12
3.3.	*class* AlgorithmWindow	12
3.4.	*class DataWindow*	12
3.5.	*class* Graph	13
3.6.	*Pair<T,U>*	13
3.7.	*MessageError*	13
3.8.	*TableConversion()*	13
3.9.	*Архитектура*	14
4.	Тестирование	15
4.1.	Ручное тестирование	15
4.2.	Unit-тестирование	17
	Заключение	19
	Приложение А. Разработанный программный код	20

ВВЕДЕНИЕ

Задача практики состоит в разработке приложения, позволяющего находить минимальное остовное дерево. Программа визуализирует пошаговое нахождение остовного дерева на графе. Для его поиска используется алгоритм Краскала.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

Спецификация проекта «Визуализация работы алгоритм Краскала»

1.1. Описание задачи

Пользователь вводит два числа - количество вершин V и количество ребер N. Затем он вводит сами ребра в следующем формате: start finish weight — то есть стартовая вершина, конечная и вес ребра. После ввода данных сортируем ребра и вполняем поиск минимального остовного дерева с помощью алгоритма Краскала, который визуализируется с помощью библиотеки swing.

1.2. Интерфейс

Интерфейс программы состоит из:

• Окно для ввода данных пользователем с кнопками начала запуска алгоритма («Start»), повторного ввода данных («Restart») и меню («Options»).

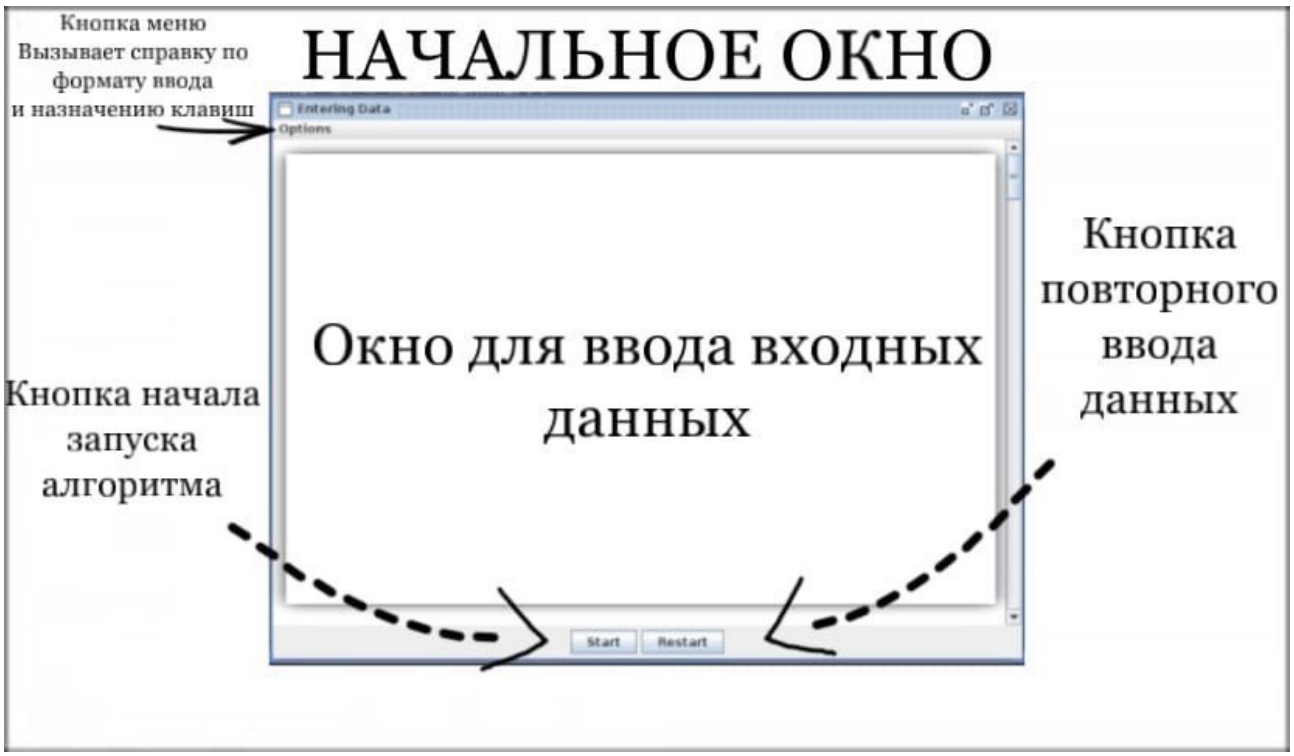

Рисунок 1 — Начальное окно

• Кнопка «Start» отвечает за начало работы алгоритма, закрытие начального окна и открытие нового. После ее нажатия данные также будут

проанализированы, и, в случае возникновения ошибки, будет появляться окно с соответствующим сообщением.

• Кнопка «Restart» служит для сброса входных данных и выводит сообщение пользователю с просьбой ввести новые данные.

• Кнопка импорт из файла.

• Кнопка «стрелка в кружочке» возвращает алгоритм в начало, печатая соответствующее сообщение в текстовое поле окна (Этот текст нужен для демонстрации кнопок и потом будет удален, вместо чего будет меняться гарф).

• Кнопка «шаг назад» делает шаг назад алгоритма и выводит сообщение.

• Кнопка «шаг вперед» выводит сообщение и делает шаг вперед у алгоритма.

• Кнопка «> >» сразу прогоняет алгоритм и выводит итоговый результат.

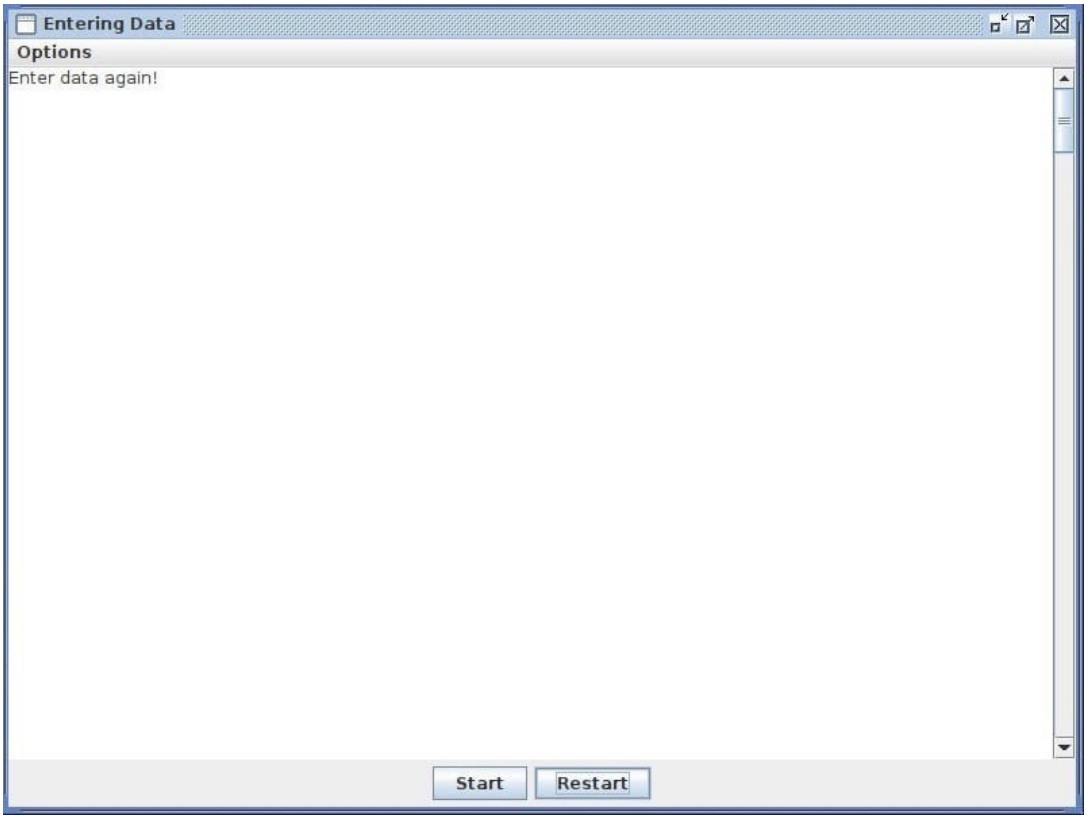

Рисунок 2 — Демонстрация работы программы после нажатия кнопки «Restart»

• Меню с пунктом «Help!», после вызова которого появляется новое окно с информацией по формату ввода и назначении кнопок.

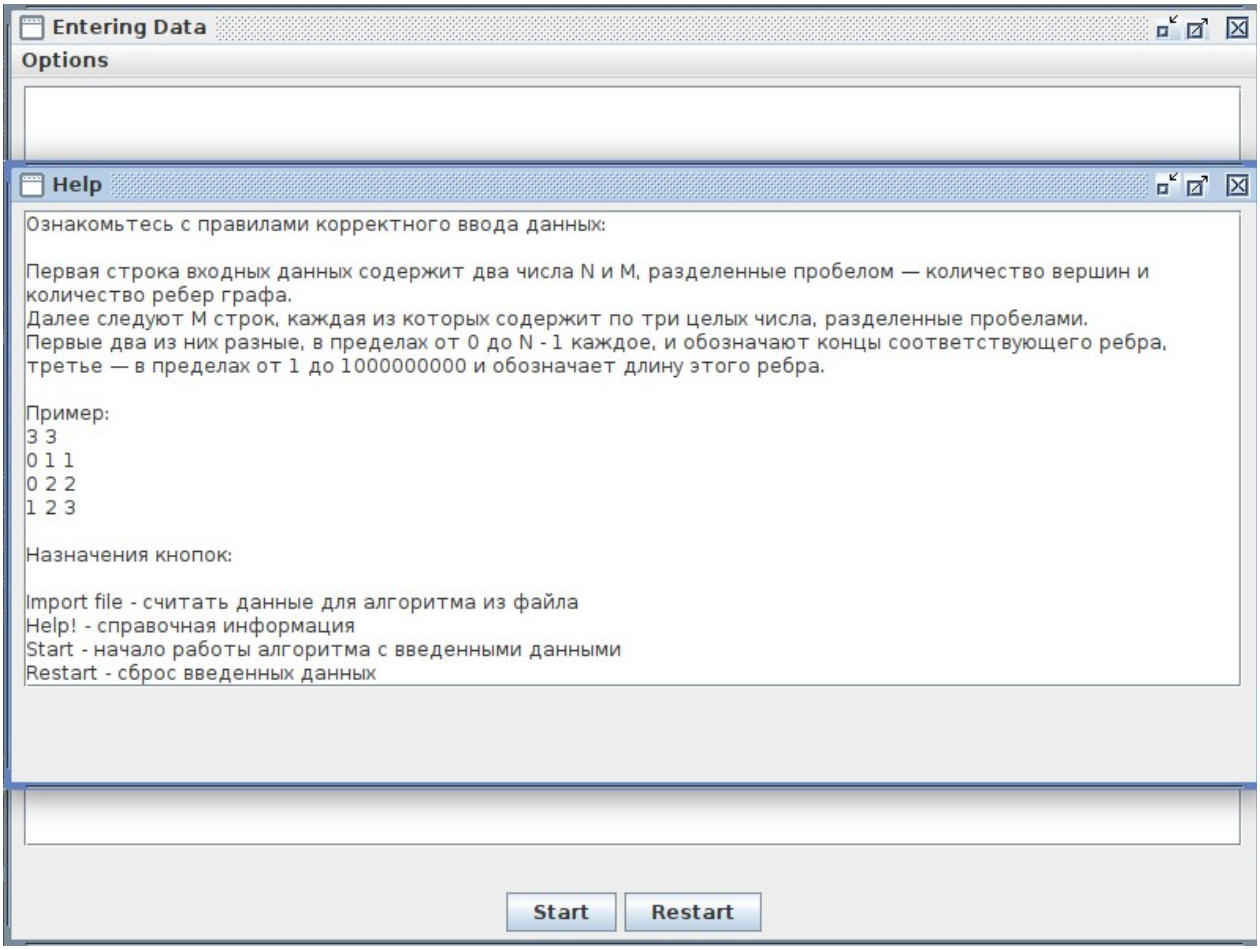

Рисунок 3 — Окно «Help!» с информацией о вводе данных

• Основное окно работы алгоритма - Algorithm Frame. Оно содержит четыре кнопки отвечающие за возврат к началу работы алгоритма, шаг назад, шаг вперёд, и переход к результату работы. На экране будет начерчен граф и массив рёбер. На каждом шаге будут изменятся цвета рёбер для наглядной демонстрации рассматриваемого шага в окне. Также в окне Logging Frame выводится информация о работе алгоритма Краскала.

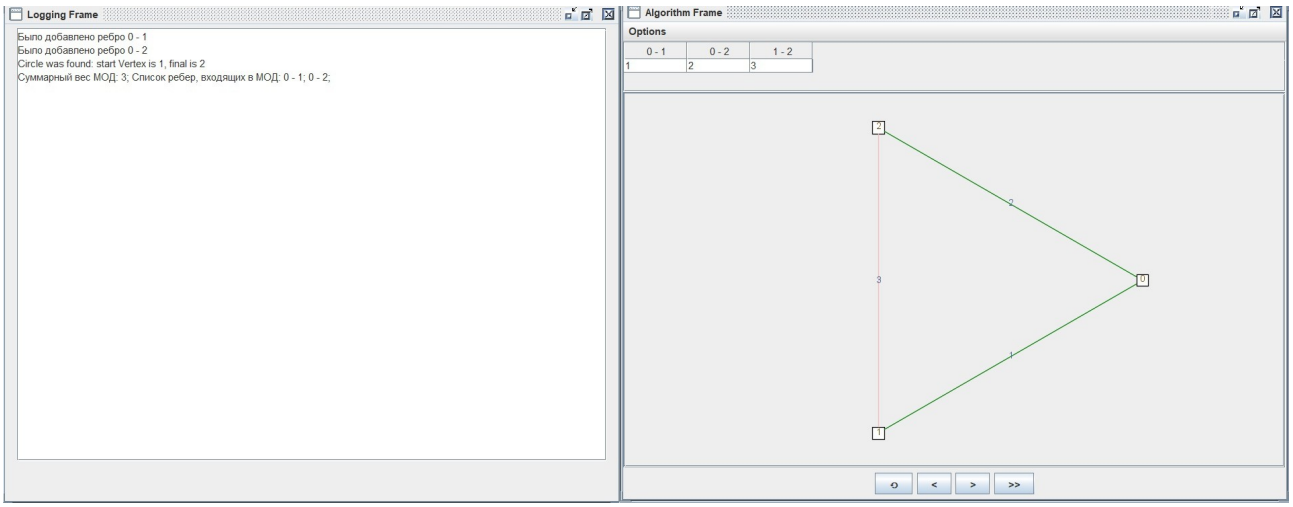

Рисунок 4 — Основное окно работы алгоритма

1.3. Входные данные

Первая строка входных данных должна содержать два числа N и M, разделенные пробелом — количество вершин и количество ребер графа. Далее следуют M строк, каждая из которых содержит по три целых числа, разделенные пробелами. Первые два из них разные, в пределах от 0 до N - 1 каждое, и обозначают концы соответствующего ребра, третье — в пределах от 1 до 1000000000 и обозначает длину этого ребра.

Пример:

3 3

0 1 1

0 2 2

1 2 3

1.4. Выходные данные

Выводится граф и массив рёбер. На каждом шаге изменяются цвета рёбер для наглядной демонстрации рассматриваемого шага. В итоге найдено минимальное остовное дерево и выписаны: вес МОД и ребра, входящие в МОД.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

1. К 4 июля начать работу над пользовательским интерфейсом,

обсудить применения алгоритмов для решения поставленной

задачи.

2. К 5 июля подготовить прототип пользовательского интерфейса и

спецификацию проекта.

3. К 6 июля реализовать основные классы и методы для реализации алгоритма.

4. К 7 июля подготовить вывод окон в приложении с помощью swing.

5. К 9 июля реализовать интерфейс: вывод графа, добавление дуг, графическая демонстрация работы алгоритма.

6. К 11 июля организовать пробное соединение интерфейса и логики.

7. К 12 июля закончить работу по логике интерфейсу программы.

8. К 12 июля протетировать программу, написать Unit-тесты.

8. К 13 июля завершить разработку проекта.

2.2. Распределение ролей в бригаде

• Чебесова Ирина – фронтенд;

• Лапина Анастасия – тестирование, документация;

• Хотяков Евгений– алгоритмист, лидер.

3. ФУНКЦИИ И СТРУКТУРЫ ПРОГРАММЫ

3.1. *class* EnterData*:*

*Х*ранит в себе список всех введенных ребер, а также количество этих ребер и количество вершин в графе.

Методы:

• dataSort() - метод для сортировки списка ребер. Ребра сортируются по весу в порядке возрастания.

• readDataFromText — считывает данные с консоли возвращает true, если считывание прошло успешно и false, если была выявлена ошибка (это было сделано, чтобы в интерфейсе обрабатывалась ошибка).

• readDataFromFile - принимает имя файла и возвращает строку, содержащую весь текст из этого файла.

• getData() - возвращает информацию о ребрах графа.

• getVertexNum() - возвращает количество вершин.

• getEdgesNum() - возвращает количество ребер.

3.2. *class* Kruscal*:*

Реализовывает алгоритм. Он хранит в себе:

Поля:

• tree_id - массив, хранящий для каждой вершины номер множества, к которому вершина принадлежит. (Изначально все вершины принадлежат разным множествам);

• result - массив, в который добавляются ребра, входящие в МОД;

• data - экземпляр класса EnterData, хранит в себе введенные пользователем данные;

• cost - суммарный вес МОД;

• currentStep - индекс, указывающий, на каком шаге алгоритма мы находимся;

• saveStack - стэк, который используется для сохранения данных при возврате на шаг назад;

• circleFlag - массив, который для каждого этапа хранит информацию о том, был ли найден цикл на данном шаге (true) или нет (false);

Методы:

В конструкторе инициализируются начальные данные и сортируется массив с ребрами по возрастанию по весу этих ребер.

• union - метода для объединения двух множеств вместе.

• checkConnected - используется после прохождения по всем ребрам графа. Возвращает true, если не удалось получить МОД (граф оказался несвязным) и false, если МОД получен.

• isEnd - проверяет, находится ли алгоритм на конечном этапе.

• nextStep - метод реализует единичную итерацию алгоритма Краскала.

• toFinal - метод прогоняет алгоритм от текущей итерации до конца алгоритма.

• prevStep - метод реализует единичный шаг назад по алгоритму.

• toStart - метод возвращает алгоритм на начальный этап.

• printResult - возвращает строку, в которой записаны все ребра итогового МОД.

• getCost() - возвращает значение cost — суммарный вес МОД.

• getVertexNum() - возвращает количество вершин в графе.

3.3. *class* AlgorithmWindow*:*

Окно для графического вывода работы алгоритма.

Метод:

• openWindow() - выводит окно работы алгоритма, окно «Help» по работе с приложением после ввода данных.

3.4. *class DataWindow:*

Открывает окно для ввода, отрисовывает необходимые кнопки.

Метод:

• openWindow() - открывает окно для ввода, отрисовывает кнопку «Options», «Import» для загрузки файла, окно «Help!» с информацией по вводе данных.

3.5. *class* Graph*:*

Класс, реализующий рисовку и обработку графа.

Метод:

- changeColor- изменяет цвет ребра графа.
- getGraphComponent() - возвращает компоненты графа.

3.6. *MessageError:*

Выводит новое окно с сообщением об ошибке.

Метод:

- throwMessageError- принимает нужное сообщение и выводит его в новом окне.

3.7. *Pair<T,U>:*

Класс для удобного хранения пар.

Поля:

- first- для хранения 1 экземпляра пары.
- second- для хранения 1 экземпляра пары.

3.8. *TableConversion():*

Нужен для вывода массива ребер на экран. Он переводит список ребер из состояния listOfEdges в два вектора для вывода массива.

Метод:

- convert — преобразовывает массив ребер в таблицу.
- getColumns - возвращает количество столбцов.
- getWeights() - возвращает веса.

3.9 Архитектура:

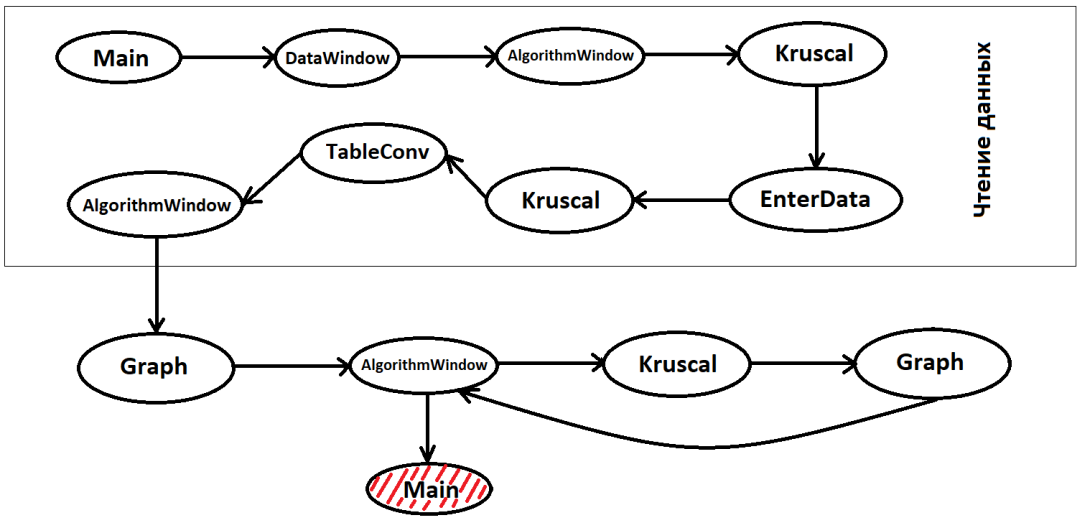

Рисунок 5 - Архитектура работы приложения

4. ТЕСТИРОВАНИЕ

4.1. Ручное тестирование

Пример 1

Входные данные:

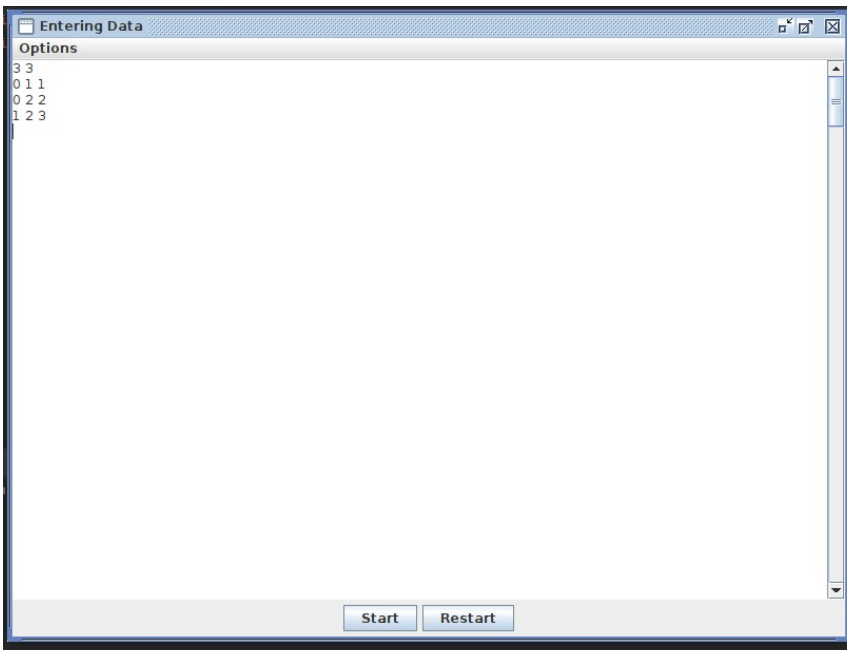

Рисунок 6 – Входные данные для примера 1

Выходные данные:

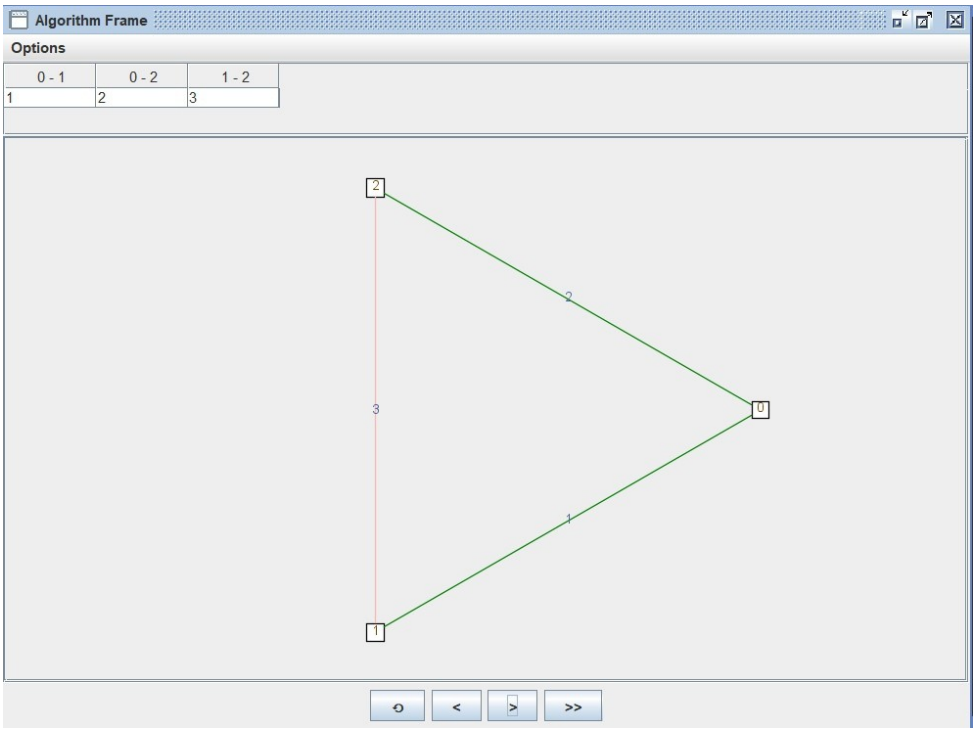

Рисунок 7 – Выходные данные для примера 1

Пример 2

Входные данные:

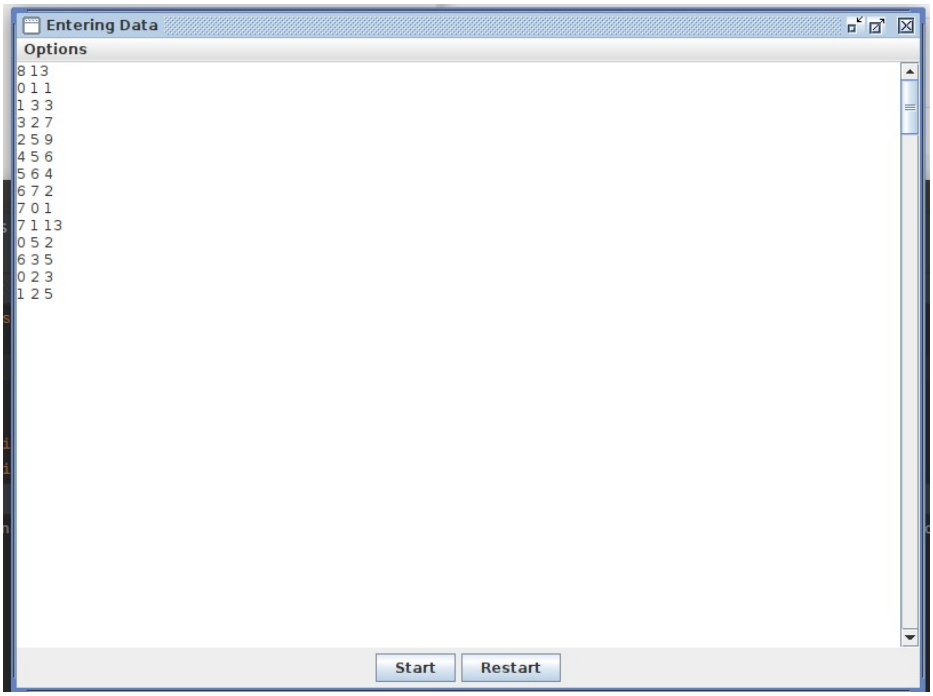

Рисунок 8 – Входные данные для примера 1

Выходные данные:

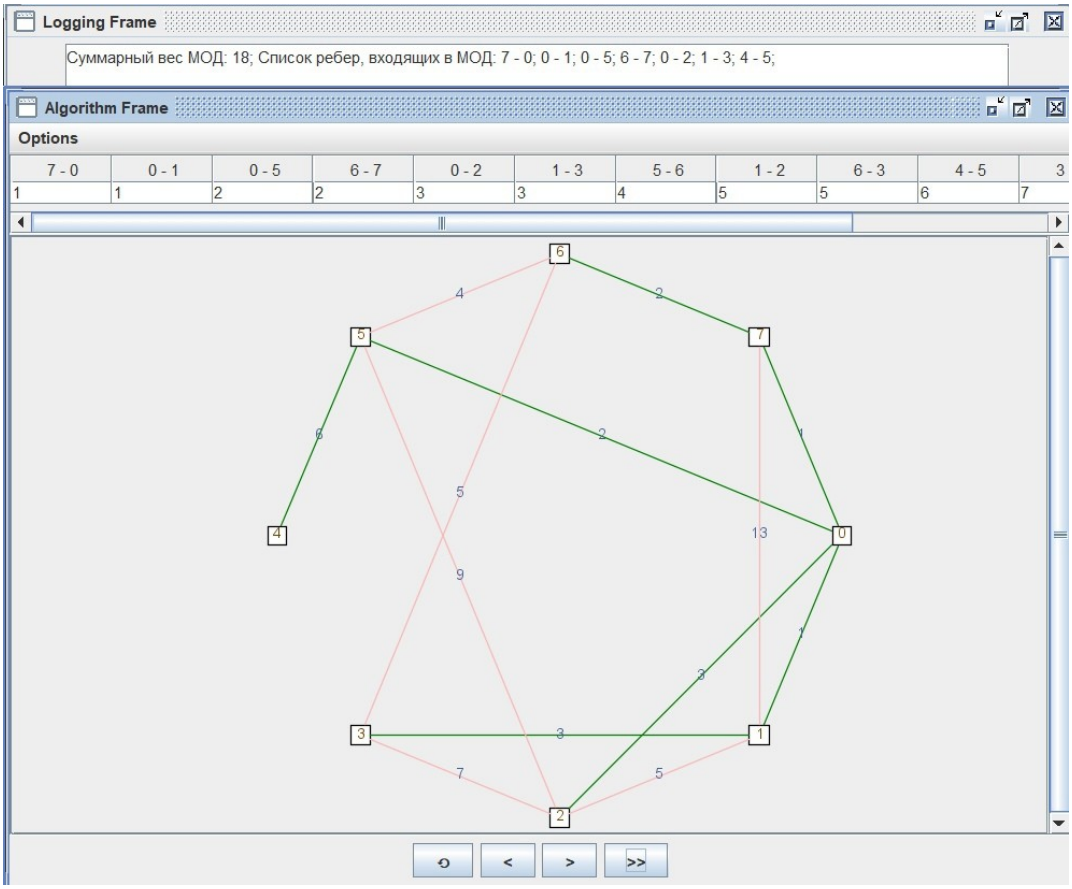

Рисунок 9 – Выходные данные для примера 1

4.2. Unit-тестирование

Для класса **EnterData** применено Unit-тестирование с использованием библиотеки Junit 4. Создан экземпляр класса, проверенна корректность ввода данных. Реализовано 2 теста – входные данные считываются корректно.

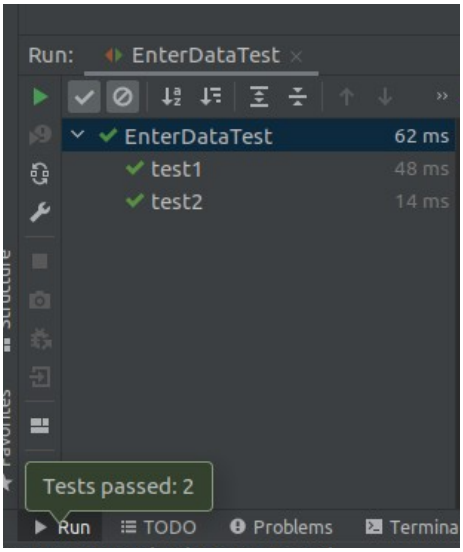

Рисунок 10 – Демострация успешно пройденных тестов для класса EnterData

Для класса **Kruscal** применено Unit-тестирование с использованием библиотеки Junit 4.

1) Созданы методы test1_check_toFinal() и test2_check_toFinal() проверяющие корректность работы метода toFinal(). Изначально было проверено значение поля cost – суммарный вес МОД (должен равняться 0), затем применен метод toFinal() и проверен суммарный вес МОД – новое значение поля cost. Также проверено возвращаемое значение тестируемой функции. Удачно пройдено 2 теста из 2.

2) test1_check_checkConnected() и test2_check_checkConnected() проверяют работу функции checkConnected() с графом, у которого можно найти МОД и у которого МОД найти нельзя. Пройдено 2 теста из 2.

3) Методы test1_check_isEnd(), test2_check_isEnd(), test3_check_isEnd() - тестируют метод isEnd(), проверяющий окончание работы алгоритма. 1 и 2 тесты это делают с помощью одношаговой функции, протестируемой ранее — oFinal(), в 3 тест с помощью многошагового применения nextStep().

Сравнивается возвращаемое значение isEnd() до работы алгоритма (false) и после его выполнения (true). Удачно пройдены все тесты.

4) test1_check_toStart() и test2_check_toStart() проверяют метод toStart(), который возвращает к началу наботы алгоритмя и выводит сообщение "Вы на начале.". Пройдено 2 теста из 2.

5) Методы test1_check_nextStep() и test2_check_nextStep() проверяют корректность работы метода nextStep() путем сравнения возвращаемого значения тестируемого метода. Оба теста пройдены успешно.

6) test1_check_prevStep() и test2_check_prevStep() тестируют prevStep() следующим образом: в начале должно возвращаться сообщение о том, что выполнение шага назад невозможно, так как мы находимся в самом начале, затем выполнение шага вперед и сравнение значения, после запуск алгоритма до конца с помощью toFinal() и возврат назад, проверка возвращаемого значения. Тесты удачно пройдены.

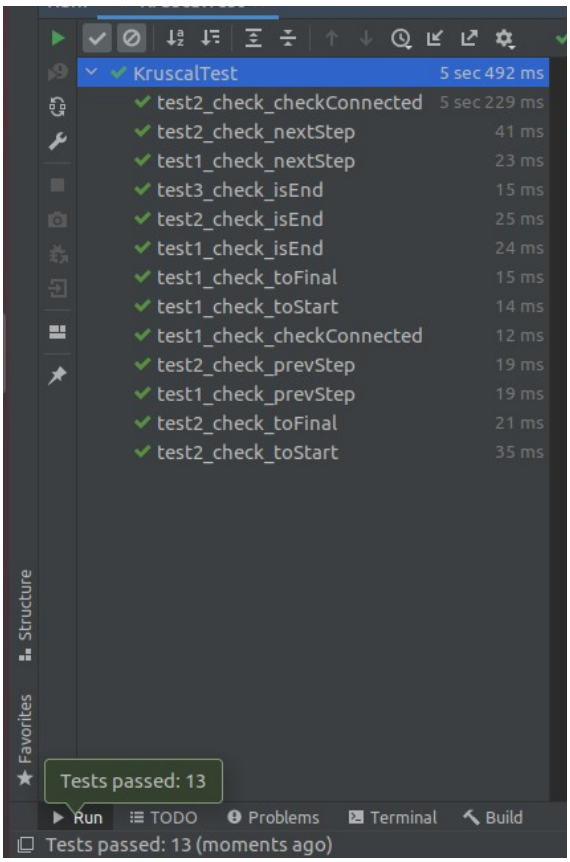

Рисунок 11 – Демострация успешно пройденных тестов для класса Kruscal

ЗАКЛЮЧЕНИЕ

В соответствии с требованиями технического задания была реализована программа для нахождения минимального остовного дерева графа, задаваемого пользователем. Разработан пользовательский интерфейс. Он позволяет считывать введенные данные, анализировать и демонстрировать работу алгоритма Краскала.

В результаты работы были освоены методы работы в команде, распределены обязанности. Также команда была ознакомлена языком программирования Java, в частности с фреймворком Swing для создания графического интефейса, библиотекой JUnit 4 для модульного тестирования.

ПРИЛОЖЕНИЕ А

РАЗРАБОТАННЫЙ КОД

Название файла: AlgorithmWindow

```
import com.mxgraph.swing.mxGraphComponent;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;

public class AlgorithmWindow extends JFrame{

    public void openWindow(String s){
        Kruscal algorithm = new Kruscal(s);
        if(!algorithm.getReadFlag()) {
            return;
        }

                Graph graph = new Graph(algorithm.getVertexNum(),
algorithm.getData());
        mxGraphComponent graphComponent = graph.getGraphComponent();

        JFrame mainFrame = new JFrame("Algorithm Frame");
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setSize(800, 600);

        JFrame logFrame = new JFrame("Logging Frame");
        logFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        logFrame.setSize(800, 600);

        JMenu menu = new JMenu("Options");
        JMenuBar menuBar = new JMenuBar();
                JMenuItem menuHelp = new JMenuItem(new
AbstractAction("Help!") {
            public void actionPerformed(ActionEvent e) {
                JFrame helpFrame = new JFrame("Help");
```

```
                    JPanel helpPanel = new JPanel();
                    JTextArea textHelp = new JTextArea(10,40);
                    JScrollPane paneHelp = new JScrollPane(textHelp);

                    textHelp.append("\nНазначения кнопок:\n\n" +
                            "Help! - справочная информация\n" +
                            "⟲ - перейти на начало работы алгоритма\n" +
                            "< - шаг алгоритма назад\n" +
                            "> - шаг алгоритма вперед\n" +
                            ">> - перейти в конец работы алгоритма");

                    helpPanel.add(paneHelp);
                    textHelp.setEditable(false);

                    helpFrame.setSize(500, 200);
                    helpFrame.getContentPane().add(helpPanel);
                    helpFrame.setLocationRelativeTo(null);
                    helpFrame.setVisible(true);
                }
            });
            menu.add(menuHelp);
            menuBar.add(menu);
            mainFrame.setJMenuBar(menuBar);

            JPanel dataPanel = new JPanel();
            JTextArea logArea = new JTextArea(32,70);
            JScrollPane pane = new JScrollPane(logArea);

                    JTable  table  =  new  JTable(algorithm.getWeights(),
algorithm.getColumns()){
                private static final long serialVersionUID = 1L;

                public boolean isCellEditable(int row, int column) {
                    return false;
                };
            };

            Box contents = new Box(BoxLayout.Y_AXIS);
            contents.setPreferredSize(new Dimension(10, 60));
```

```
                                                    contents.add(new
JScrollPane(table,JScrollPane.VERTICAL_SCROLLBAR_NEVER,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED));
                table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
                table.getTableHeader().setReorderingAllowed(false);


                JPanel buttonPanel = new JPanel();
                JButton buttonAgain = new JButton("⟲");
                buttonPanel.add(buttonAgain);

                buttonAgain.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent e) {
                        logArea.append(algorithm.toStart(graph) + "\n");
                    }
                });

                JButton buttonPrev = new JButton("<");
                buttonPanel.add(buttonPrev);
                buttonPrev.addActionListener(new ActionListener() {

                    @Override
                    public void actionPerformed(ActionEvent e) {
                        logArea.append(algorithm.prevStep(graph) + "\n");
                    }
                });

                JButton buttonNext = new JButton(">");
                buttonPanel.add(buttonNext);
                buttonNext.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent e) {
                        if(!algorithm.isEnd()) {
                                logArea.append(algorithm.nextStep(graph) + "\
n");
                        }
                        else{
                            if(!algorithm.checkConnected()) {
```

```
                                logArea.append(algorithm.printResult(graph)
+ "\n");
                            }
                            else{
                                 logArea.append("Не удалось получить МОД из
заданного графа!" + "\n");
                            }
                        }
                    }
                });

                JButton buttonFinal = new JButton(">>");
                buttonPanel.add(buttonFinal);

                buttonFinal.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent e) {
                        algorithm.toFinal(graph);
                        if(!algorithm.checkConnected()) {
                             logArea.append(algorithm.printResult(graph) + "\
n");
                        }
                        else{
                                 logArea.append("Не удалось получить МОД из
заданного графа!" + "\n");
                        }
                    }
                });

                logArea.setEditable(false);
                dataPanel.add(pane);
                     mainFrame.getContentPane().add(BorderLayout.CENTER,  new
JScrollPane(graphComponent));
                        mainFrame.getContentPane().add(BorderLayout.NORTH,
contents);
                        mainFrame.getContentPane().add(BorderLayout.SOUTH,
buttonPanel);
                logFrame.getContentPane().add(dataPanel);
                mainFrame.setLocationRelativeTo(null);
```

```
            logFrame.setVisible(true);
            mainFrame.setVisible(true);
        }
    }
```

Название файла: DataWindow

```
    import java.awt.*;
    import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;
    import java.io.File;
    import java.io.FileNotFoundException;

    import javax.swing.*;

    public class DataWindow extends JFrame {

        public void openWindow() {
            JFrame.setDefaultLookAndFeelDecorated(true);
            JFrame frame = new JFrame("Entering Data");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(800,600);
            frame.setLocationRelativeTo(null);

            JMenu menu = new JMenu("Options");
            JMenuBar menuBar = new JMenuBar();
                        JMenuItem helpMenu = new JMenuItem(new
AbstractAction("Help!") {
                public void actionPerformed(ActionEvent e) {
                    JFrame helpFrame = new JFrame("Help");

                    JPanel helpPanel = new JPanel();
                    JTextArea textHelp = new JTextArea(20,70);
                    JScrollPane paneHelp = new JScrollPane(textHelp);
                            textHelp.append("Ознакомьтесь с правилами
корректного ввода данных:"+
                            "\n\n" +
                              "Первая строка входных данных содержит два
числа N и M, разделенные пробелом — количество вершин и " +
                            "\n" +
```

```
                                "количество ребер графа. " +
                                "\n" +
                                   "Далее следуют M строк, каждая из которых
содержит по три целых числа, разделенные пробелами. " +
                                "\n" +
                                  "Первые два из них разные, в пределах от 0
до N - 1 каждое, и обозначают концы соответствующего ребра, " +
                                "\n" +
                                   "третье — в пределах от 1 до 1000000000 и
обозначает длину этого ребра.\n" +
                                "\n" +
                                "Пример:\n" +
                                "3 3\n" +
                                "0 1 1\n" +
                                "0 2 2\n" +
                                "1 2 3" +
                                "\n\nНазначения кнопок:\n\n" +
                                 "Import file - считать данные для алгоритма
из файла\n" +
                                "Help! - справочная информация\n" +
                                        "Start - начало работы алгоритма с
введенными данными\n" +
                                "Restart - сброс введенных данных");
                        helpPanel.add(paneHelp);
                        textHelp.setEditable(false);

                        helpFrame.setSize(800, 400);
                        helpFrame.getContentPane().add(helpPanel);
                        helpFrame.setLocationRelativeTo(null);
                        helpFrame.setVisible(true);
                    }
                });
                                JMenuItem fileMenu = new JMenuItem(new
AbstractAction("Import file"){
                    public void actionPerformed(ActionEvent e) {
                        JFileChooser fileOpen = new JFileChooser();
                        int ret = fileOpen.showDialog(null, "Открыть файл");
                        if (ret == JFileChooser.APPROVE_OPTION) {
                            File file = fileOpen.getSelectedFile();
```

```
                        String data = "";
                        try {
                            data = EnterData.readDataFromFile(file);
                                    } catch (FileNotFoundException
fileNotFoundException) {
                            fileNotFoundException.printStackTrace();
                        }
                        frame.setVisible(false);
                        AlgorithmWindow window = new AlgorithmWindow();
                        window.openWindow(data);
                    }
                }
            });

            menu.add(fileMenu);
            menu.add(helpMenu);
            menuBar.add(menu);
            frame.setJMenuBar(menuBar);

            JPanel buttonPanel = new JPanel();
            JPanel textPanel = new JPanel();

            JTextArea textArea = new JTextArea(32,70);
            JScrollPane pane = new JScrollPane(textArea);
            textPanel.add(pane);

            JButton button1 = new JButton("Start");
            buttonPanel.add(button1);

            button1.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    String s = textArea.getText();
                    frame.setVisible(false);
                    AlgorithmWindow window = new AlgorithmWindow();
                    window.openWindow(s);
                }
            });
```

```
            JButton button2 = new JButton("Restart");
            buttonPanel.add(button2);

            button2.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    textArea.setText("Enter data again!");
                }
            });

            frame.getContentPane().add(textPanel);
            frame.getContentPane().add(BorderLayout.SOUTH, buttonPanel);
            frame.setPreferredSize(new Dimension(800, 600));
            frame.setVisible(true);
        }
    }
```

Название файла: EnterData

```
    import java.io.File;
    import java.io.FileNotFoundException;
    import java.util.ArrayList;
    import java.util.Comparator;
    import java.util.Scanner;

    public class EnterData {
        private int vertexNum, edgesNum;
               private   ArrayList<Pair<Pair<Integer,   Integer>,   Long>>
listOfEdges;

        public EnterData(){
            listOfEdges = new ArrayList<>();
        }

            public  static  String  readDataFromFile(File  file)  throws
FileNotFoundException {
            String s = "";
            try {
                Scanner in = new Scanner(file);
                while (in.hasNext())
```

```
                    s += in.nextLine() + "\r\n";
                in.close();
            }
            catch (FileNotFoundException e){
                    MessageError.throwMessageError("Файл не смог
открыться.");
                return null;
            }
            return s;
        }

        public boolean readDataFromText(String data_){
            if(data_ == null)
                return false;

            Scanner scanner = new Scanner(data_);

            try{
                vertexNum = scanner.nextInt();
                edgesNum = scanner.nextInt();
                if(vertexNum < 1 || edgesNum < 1){
                        throw new IllegalArgumentException("Некорректно
введенные данные! Количество ребер и вершин должены быть натуральными
числами");
                }

            }
            catch (java.util.InputMismatchException e){
                    MessageError.throwMessageError("Некорректно введенные
данные! Количество ребер и вершин должны быть числами.");
                return false;
            }
            catch (IllegalArgumentException e){
                MessageError.throwMessageError(e.getMessage());
                return false;
            }

            long tmpInt = 0;
            int tmpFirst = 0;
```

```
            int tmpSecond = 0;
            for (int i = 0; i < edgesNum; i++){
                if(!scanner.hasNextLine()){
                                MessageError.throwMessageError("Некорректно
введенные данные! Введено недостаточно ребер.");
                    return false;
                }
                try{
                    tmpFirst = scanner.nextInt();
                    tmpSecond = scanner.nextInt();
                    tmpInt = scanner.nextLong();
                      if(tmpFirst >= vertexNum || tmpSecond >= vertexNum
|| tmpFirst < 0 || tmpSecond < 0)
                            throw new IllegalArgumentException("Некорректно
введенные данные! Номер введенного ребра не удовлетроряет условию 0 <= x
<= N-1");
                    if(tmpInt < 1)
                            throw new IllegalArgumentException("Некорректно
введенные данные! Вес ребра должен быть положительным");
                }
                catch (java.util.InputMismatchException e){
                                MessageError.throwMessageError("Некорректно
введенные данные! Требуется вводить числа.");
                    return false;

                }
                catch (IllegalArgumentException e){
                    MessageError.throwMessageError(e.getMessage());
                    return false;
                }
                        listOfEdges.add(new  Pair<>(new  Pair<>(tmpFirst,
tmpSecond), tmpInt));
            }
            scanner.close();
            return  true;
        }

        public void dataSort(){
```

```
            listOfEdges.sort(new Comparator<Pair<Pair<Integer, Integer>,
Long>>() {
                @Override
                   public int compare(Pair<Pair<Integer, Integer>, Long>
o1, Pair<Pair<Integer, Integer>, Long> o2) {
                    return o1.second > o2.second ? 1 : -1;
                }
            });
        }

            public final ArrayList<Pair<Pair<Integer, Integer>, Long>>
getData(){
            return listOfEdges;
        }

        public Integer getVertexNum(){
            return vertexNum;
        }
        public Integer getEdgesNum(){
            return edgesNum;
        }
    }
```

Название файла: Graph

```
    import javax.swing.*;

    import com.mxgraph.swing.mxGraphComponent;
    import com.mxgraph.view.mxGraph;

    import java.util.ArrayList;

    public class Graph extends JFrame {

        private mxGraph graph;
        private mxGraphComponent graphComponent;
        private ArrayList<Object> vertexArray;
           public Graph(Integer vertexNum, ArrayList<Pair<Pair<Integer,
Integer>, Long>> data)
        {
```

```
            graph = new mxGraph();
            vertexArray = new ArrayList<>();
            Object parent = graph.getDefaultParent();
            String style = "rounded;strokeColor=black;fillColor=white";
            graph.getModel().beginUpdate();
            double angle = 2 * Math.PI / vertexNum;
            double alpha;
            try
            {
                for (int i = 0; i < vertexNum; i++){
                    int R = 210;
                    double r = Math.sqrt(R);
                    alpha = i * angle;
                    double x = 400 + R * Math.cos(alpha);
                    double y = 215 + R * Math.sin(alpha);
                    String value = Integer.toString(i);
                        Object vertex = graph.insertVertex(parent, null,
value, x, y, r, r, style);
                    vertexArray.add(vertex);
                }
                for (Pair<Pair<Integer, Integer>, Long> edge: data) {
                    if (data.size() <= 40){
                        String value = Long.toString(edge.second);
                                    graph.insertEdge(parent, null, value,
vertexArray.get(edge.first.first),     vertexArray.get(edge.first.second),
"strokeColor=#D1D1D1;endArrow=none");
                    }
                    else{
                                       graph.insertEdge(parent, null, "",
vertexArray.get(edge.first.first),     vertexArray.get(edge.first.second),
"strokeColor=#D1D1D1;endArrow=none");
                    }
                }
            }
            finally
            {
                graph.getModel().endUpdate();
            }
            graphComponent = new mxGraphComponent(graph);
```

```
            graphComponent.setEnabled(false);
            graphComponent.setAutoScroll(true);
        }

         public void changeColor (String color, Integer start, Integer
finish){
            String style = "strokeColor=" + color + ";endArrow=none";
                                                    for(Object      edge:
graph.getEdgesBetween(vertexArray.get(start), vertexArray.get(finish)))
                graph.getModel().setStyle(edge, style);
        }

        public mxGraphComponent getGraphComponent() {
            return graphComponent;
        }
    }
```

Название файла: Kruscal

```
    IMPORT JAVA.UTIL.ARRAYLIST;
    IMPORT JAVA.UTIL.STACK;

    PUBLIC CLASS KRUSCAL {
        PRIVATE ARRAYLIST<INTEGER> TREE_ID;
        PRIVATE ARRAYLIST<PAIR<INTEGER, INTEGER>> RESULT;
        PRIVATE ENTERDATA DATA;
        PRIVATE LONG COST;
        PRIVATE INT CURRENTSTEP;
        PRIVATE BOOLEAN READFLAG;
        PRIVATE STACK<PAIR<INTEGER, INTEGER>> SAVESTACK;
        PRIVATE ARRAYLIST<BOOLEAN> CIRCLEFLAG;

        //TABLE
        PRIVATE OBJECT[] COLUMNSHEADEREDGES;
        PRIVATE OBJECT[][] ARRAYOFWEIGHT;

        PUBLIC KRUSCAL(STRING DATA_){
            COST = 0;
            CURRENTSTEP = 0;
            CIRCLEFLAG = NEW ARRAYLIST<>();
```

```
            RESULT = NEW ARRAYLIST<>();
            SAVESTACK = NEW STACK<>();
            TREE_ID = NEW ARRAYLIST<>();

            DATA = NEW ENTERDATA();
            READFLAG = DATA.READDATAFROMTEXT(DATA_);
            DATA.DATASORT();

            //TABLE
            TABLECONVERSION TABLEDATA = NEW TABLECONVERSION();
            TABLEDATA.CONVERT(DATA.GETDATA());
            COLUMNSHEADEREDGES = TABLEDATA.GETCOLUMNS();
            ARRAYOFWEIGHT = TABLEDATA.GETWEIGHTS();

            INT VERTEXNUM = DATA.GETVERTEXNUM();
            FOR (INT I = 0; I < VERTEXNUM; I++) {
                TREE_ID.ADD(I);
            }
        }

        PUBLIC FINAL BOOLEAN GETREADFLAG(){
            RETURN READFLAG;
        }

        PRIVATE VOID UNION(INT START, INT FINISH){
                        INT  OLD_ID  =  TREE_ID.GET(FINISH),  NEW_ID  =
TREE_ID.GET(START);
            FOR (INT I = 0; I < DATA.GETVERTEXNUM(); ++I)
                IF (TREE_ID.GET(I) == OLD_ID)
                    TREE_ID.SET(I, NEW_ID);
        }


        PUBLIC BOOLEAN CHECKCONNECTED(){
            INT SETINDEX = TREE_ID.GET(0);
            FOR (INT I = 1; I < DATA.GETVERTEXNUM(); I++)
                IF (TREE_ID.GET(I) != SETINDEX) {
                    RETURN TRUE;
                }
```

```
            RETURN FALSE;
        }

        PUBLIC STRING NEXTSTEP(GRAPH GRAPH){
            IF(SAVESTACK.ISEMPTY()) {
                INT START, FINISH;
                LONG WEIGHT;
                            PAIR<PAIR<INTEGER,  INTEGER>,  LONG>  EDGE  =
DATA.GETDATA().GET(CURRENTSTEP);
                START = EDGE.FIRST.FIRST;
                FINISH = EDGE.FIRST.SECOND;
                WEIGHT = EDGE.SECOND;
                IF (TREE_ID.GET(START) != TREE_ID.GET(FINISH)) {
                    COST += WEIGHT;
                    RESULT.ADD(NEW PAIR<>(START, FINISH));
                    UNION(START, FINISH);
                    CURRENTSTEP++;
                    CIRCLEFLAG.ADD(FALSE);
                    GRAPH.CHANGECOLOR("GREEN", START, FINISH);
                      RETURN ("БЫЛО ДОБАВЛЕНО РЕБРО " + START + " - " +
FINISH);
                } ELSE {
                    CIRCLEFLAG.ADD(TRUE);
                    RESULT.ADD(NEW PAIR<>(START, FINISH));
                    CURRENTSTEP++;
                    GRAPH.CHANGECOLOR("RED", START, FINISH);
                     RETURN ("CIRCLE WAS FOUND: START VERTEX IS " + START
+ ", FINAL IS " + FINISH);
                }
            }
            ELSE{
                IF(CIRCLEFLAG.GET(CURRENTSTEP++)){
                        GRAPH.CHANGECOLOR("RED",  SAVESTACK.PEEK().FIRST,
SAVESTACK.PEEK().SECOND);
                        RETURN ("CIRCLE WAS FOUND: START VERTEX IS " +
SAVESTACK.PEEK().FIRST + ", FINAL IS " + SAVESTACK.POP().SECOND);
                }
                ELSE{
```

```
                        GRAPH.CHANGECOLOR("GREEN", SAVESTACK.PEEK().FIRST,
SAVESTACK.PEEK().SECOND);
                                    RETURN ("БЫЛО ДОБАВЛЕНО РЕБРО " +
SAVESTACK.PEEK().FIRST + " - " + SAVESTACK.POP().SECOND);
                }
            }
        }

        PUBLIC STRING PREVSTEP(GRAPH GRAPH){
            IF(CURRENTSTEP != 0 && !CIRCLEFLAG.GET(CURRENTSTEP-1)) {
                CURRENTSTEP--;
                SAVESTACK.PUSH(RESULT.GET(CURRENTSTEP));
                                        GRAPH.CHANGECOLOR("#D1D1D1",
RESULT.GET(CURRENTSTEP).FIRST, RESULT.GET(CURRENTSTEP).SECOND);
                RETURN ("РЕБРО " + RESULT.GET(CURRENTSTEP).FIRST + " - "
+ RESULT.GET(CURRENTSTEP).SECOND + " БЫЛО УДАЛЕНО ИЗ МОД");
            }
                ELSE IF(CURRENTSTEP != 0 && !CIRCLEFLAG.ISEMPTY() &&
CIRCLEFLAG.GET(CURRENTSTEP-1)){
                CURRENTSTEP--;
                SAVESTACK.PUSH(RESULT.GET(CURRENTSTEP));
                                        GRAPH.CHANGECOLOR("#D1D1D1",
RESULT.GET(CURRENTSTEP).FIRST, RESULT.GET(CURRENTSTEP).SECOND);
                RETURN ("НА ПРЕДЫДУЩЕМ ШАГЕ БЫЛ ВЫЯВЛЕН ЦИКЛ");
            }
            ELSE {
                RETURN ("ВОЗВРАТ НА ПРЕДЫДУЩИЙ ШАГ НЕВОЗМОЖЕН - ВЫ НА
ПЕРВОМ ШАГЕ");
            }
        }

        PUBLIC STRING TOFINAL(GRAPH GRAPH){
            ARRAYLIST<PAIR<PAIR<INTEGER, INTEGER>, LONG>> LISTOFEDGES =
DATA.GETDATA();
            FOR(INT I = CURRENTSTEP; I < LISTOFEDGES.SIZE(); I++){
                NEXTSTEP(GRAPH);
            }
            RETURN PRINTRESULT(GRAPH);
        }
```

```
        PUBLIC STRING TOSTART(GRAPH GRAPH){
            WHILE(CURRENTSTEP != 0){
                PREVSTEP(GRAPH);
            }
            RETURN "ВЫ НА НАЧАЛЕ.";
        }


        PUBLIC STRING PRINTRESULT(GRAPH GRAPH){
            STRING RESULT_STRING = NEW STRING();
            IF(!CHECKCONNECTED()) {
                RESULT_STRING += "СУММАРНЫЙ ВЕС МОД: " + COST + "; ";
                RESULT_STRING += "СПИСОК РЕБЕР, ВХОДЯЩИХ В МОД: ";
                FOR(INT I = 0; I < RESULT.SIZE(); I++){
                    IF(!CIRCLEFLAG.GET(I))
                          RESULT_STRING += RESULT.GET(I).FIRST + " - " +
RESULT.GET(I).SECOND + "; ";
                }
            }
            RETURN RESULT_STRING;
        }

        PUBLIC BOOLEAN ISEND(){
            IF(CURRENTSTEP == DATA.GETDATA().SIZE())
                RETURN TRUE;
            RETURN FALSE;
        }

        PUBLIC LONG GETCOST(){RETURN COST;}
        PUBLIC FINAL OBJECT[] GETCOLUMNS(){
            RETURN COLUMNSHEADEREDGES;
        }
        PUBLIC OBJECT[][] GETWEIGHTS(){
            RETURN ARRAYOFWEIGHT;
        }

        PUBLIC INTEGER GETVERTEXNUM(){ RETURN DATA.GETVERTEXNUM(); }
```

```
            PUBLIC FINAL ARRAYLIST<PAIR<PAIR<INTEGER, INTEGER>, LONG>>
GETDATA(){ RETURN DATA.GETDATA(); }
    }
```

Название файла: Main

```
    public class Main {
        public static void main(String arg[]){
            DataWindow window = new DataWindow();
            window.openWindow();
        }
    }
```

Название файла: MessageError

```
    import java.awt.*;
    import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;

    import javax.swing.*;

    public class MessageError extends JFrame{
        public static void throwMessageError(String errorName) {
            JFrame.setDefaultLookAndFeelDecorated(true);
            JFrame message = new JFrame("Error");
             message.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE)
;
            message.setSize(800, 100);
            message.setLocationRelativeTo(null);

            JPanel buttonPanel = new JPanel();
            JButton buttonOK = new JButton("Ok");
            buttonPanel.add(buttonOK);

            buttonOK.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    message.setVisible(false);
                    System.exit(1);
                }
            });
```

```
            JPanel labelPanel = new JPanel();
            JLabel errorLabel = new JLabel(errorName);
            labelPanel.add(errorLabel);

                        message.getContentPane().add(BorderLayout.CENTER,
labelPanel);
                         message.getContentPane().add(BorderLayout.SOUTH,
buttonPanel);
            message.setVisible(true);
        }
    }
```

Название файла: Pair<T,U>

```
    public class Pair<T,U> {
        public T first;
        public U second;

        public Pair(T first, U second) {
            this.first = first;
            this.second = second;
        }
    }
```

Название файла: TableConversion

```
    import java.util.ArrayList;
    import java.util.Arrays;

    public class TableConversion {
        private Object[] columnsHeaderEdges;
        private Object[][] arrayOfWeight;

        public TableConversion() {
            arrayOfWeight = new Object[1][];
        }

            public  void  convert(ArrayList<Pair<Pair<Integer,  Integer>,
Long>> listOfEdges){
            ArrayList<String> edgesNames = new ArrayList<>();
            ArrayList<Long> weight = new ArrayList<>();
```

```
            for(Pair<Pair<Integer, Integer>, Long> edge: listOfEdges){
                        edgesNames.add(edge.first.first  +  "  -  "  +
edge.first.second);
                weight.add(edge.second);
            }
            columnsHeaderEdges = edgesNames.toArray();
            arrayOfWeight[0] = weight.toArray();
        }

        public final Object[] getColumns(){
            return columnsHeaderEdges;
        }

        public Object[][] getWeights(){
            return arrayOfWeight;
        }
    }
```

Название файла: EnterDataTest

```
    import org.junit.Test;
    import java.util.Scanner;

    import static org.junit.Assert.*;

    public class EnterDataTest{
        @Test
        public void test1(){
            String data_ =  "8 13\n" +
                    "0 1 1\n" +
                    "1 3 3\n" +
                    "3 2 7\n" +
                    "2 5 9\n" +
                    "4 5 6\n" +
                    "5 6 4\n" +
                    "6 7 2\n" +
                    "7 0 1\n" +
                    "7 1 13\n" +
                    "0 5 2\n" +
                    "6 3 5\n" +
```

```
                    "0 2 3\n" +
                    "1 2 5";
            EnterData data = new EnterData();
            data.readDataFromText(data_);
            Scanner scanner = new Scanner(data_);
               Integer VertexNum = new Integer(scanner.nextInt());//new
Integer(8);
            Integer EdgesNum = new Integer(scanner.nextInt());
            assertEquals(data.getVertexNum(), VertexNum);
            assertEquals(data.getEdgesNum(), EdgesNum);
        }

        @Test
        public void test2(){
            String data_ =  "4 4\n" +
                    "3 3 1\n" +
                    "1 1 2\n" +
                    "2 1 3\n" +
                    "3 1 4";
            EnterData data = new EnterData();
            data.readDataFromText(data_);
            Scanner scanner = new Scanner(data_);
               Integer VertexNum = new Integer(scanner.nextInt());//new
Integer(8);
            Integer EdgesNum = new Integer(scanner.nextInt());
            assertEquals(data.getVertexNum(), VertexNum);
            assertEquals(data.getEdgesNum(), EdgesNum);
            //assertEquals(1, 1);
        }
    }
```

Название файла: KruscalTest

```
    import org.junit.Test;

    import static org.junit.Assert.*;

    public class KruscalTest {
        @Test
        public void test1_check_toFinal(){
```

```
            String data_ =  "8 13\n" +
                    "0 1 1\n" +
                    "1 3 3\n" +
                    "3 2 7\n" +
                    "2 5 9\n" +
                    "4 5 6\n" +
                    "5 6 4\n" +
                    "6 7 2\n" +
                    "7 0 1\n" +
                    "7 1 13\n" +
                    "0 5 2\n" +
                    "6 3 5\n" +
                    "0 2 3\n" +
                    "1 2 5";
            Kruscal kruscal = new Kruscal(data_);
            assertEquals(kruscal.getCost(),0);
                     Graph  graph  =  new  Graph(kruscal.getVertexNum(),
kruscal.getData());
            kruscal.toFinal(graph);
             assertEquals(kruscal.toFinal(graph), "Суммарный вес МОД: 18;
Список ребер, входящих в МОД: 7 - 0; 0 - 1; 0 - 5; 6 - 7; 0 - 2; 1 - 3; 4
- 5; ");
            assertEquals(kruscal.getCost(), 18);
            //assertEquals(kruscal.toFinal(graph), "18");


        }

        @Test
        public void test2_check_toFinal(){
            String data_ =  "3 3\n" +
                    "0 1 1\n" +
                    "0 2 2\n" +
                    "1 2 3";

            Kruscal kruscal = new Kruscal(data_);
            assertEquals(kruscal.getCost(),0);
                     Graph  graph  =  new  Graph(kruscal.getVertexNum(),
kruscal.getData());
```

```
            kruscal.toFinal(graph);
             assertEquals(kruscal.toFinal(graph), "Суммарный вес МОД: 3;
Список ребер, входящих в МОД: 0 - 1; 0 - 2; ");
            assertEquals(kruscal.getCost(), 3);
        }

        @Test
        public void test1_check_checkConnected(){
            //проверка с графом, у которого можно найти МОД
            String data_ =  "3 3\n" +
                    "0 1 1\n" +
                    "0 2 2\n" +
                    "1 2 3";
            Kruscal kruscal = new Kruscal(data_);
                     Graph  graph  =  new  Graph(kruscal.getVertexNum(),
kruscal.getData());
            kruscal.toFinal(graph);
            assertEquals(kruscal.checkConnected(),false);
        }

        @Test
        public void test2_check_checkConnected(){
            //проверка с графом, у которого нельзя найти МОД
            String data_ =  "3 1\n" +
                    "1 2 4";
            Kruscal kruscal = new Kruscal(data_);
                     Graph  graph  =  new  Graph(kruscal.getVertexNum(),
kruscal.getData());
            kruscal.toFinal(graph);
            assertEquals(kruscal.checkConnected(),true);
        }

        @Test
        //одношаговая - "автоматическая" проверка с помощью toFinal()
        public void test1_check_isEnd(){
            String data_ =  "3 3\n" +
                    "0 1 1\n" +
                    "0 2 2\n" +
                    "1 2 3";
```

```
            Kruscal kruscal = new Kruscal(data_);
                assertEquals(kruscal.isEnd(), false);   //до прохождения
алгоритма - "не конец"
                    Graph  graph  =  new  Graph(kruscal.getVertexNum(),
kruscal.getData());
            kruscal.toFinal(graph);
              assertEquals(kruscal.isEnd(), true);    //после прохождения
алгоритма - "конец"
        }

        @Test
        public void test2_check_isEnd(){
            String data_ =   "8 13\n" +
                    "0 1 1\n" +
                    "1 3 3\n" +
                    "3 2 7\n" +
                    "2 5 9\n" +
                    "4 5 6\n" +
                    "5 6 4\n" +
                    "6 7 2\n" +
                    "7 0 1\n" +
                    "7 1 13\n" +
                    "0 5 2\n" +
                    "6 3 5\n" +
                    "0 2 3\n" +
                    "1 2 5";
            Kruscal kruscal = new Kruscal(data_);
                assertEquals(kruscal.isEnd(), false);   //до прохождения
алгоритма - "не конец"
                    Graph  graph  =  new  Graph(kruscal.getVertexNum(),
kruscal.getData());
            kruscal.toFinal(graph);
              assertEquals(kruscal.isEnd(), true);    //после прохождения
алгоритма - "конец"
        }


        @Test
        //многошаговая проверка с помощью nextStep()
```

```
    public void test3_check_isEnd(){
        String data_ =  "3 3\n" +
                "0 1 1\n" +
                "0 2 2\n" +
                "1 2 3";
        Kruscal kruscal = new Kruscal(data_);
                 Graph  graph  =  new  Graph(kruscal.getVertexNum(),
kruscal.getData());
            assertEquals(kruscal.isEnd(),  false);   //до прохождения
алгоритма - "не конец"
        kruscal.nextStep(graph);
        assertEquals(kruscal.isEnd(), false);
        kruscal.nextStep(graph);
        assertEquals(kruscal.isEnd(), false);
        kruscal.nextStep(graph);
          assertEquals(kruscal.isEnd(), true);    //после прохождения
алгоритма - "конец"
    }

    @Test
    public void test1_check_toStart(){
        String data_ =  "3 3\n" +
                "0 1 1\n" +
                "0 2 2\n" +
                "1 2 3";
        Kruscal kruscal = new Kruscal(data_);
                 Graph  graph  =  new  Graph(kruscal.getVertexNum(),
kruscal.getData());
        kruscal.toFinal(graph);
        assertEquals(kruscal.toStart(graph), "Вы на начале.");
    }

    @Test
    public void test2_check_toStart(){
        String data_ =   "8 13\n" +
                "0 1 1\n" +
                "1 3 3\n" +
                "3 2 7\n" +
                "2 5 9\n" +
```

```
                    "4 5 6\n" +
                    "5 6 4\n" +
                    "6 7 2\n" +
                    "7 0 1\n" +
                    "7 1 13\n" +
                    "0 5 2\n" +
                    "6 3 5\n" +
                    "0 2 3\n" +
                    "1 2 5";
            Kruscal kruscal = new Kruscal(data_);
                     Graph  graph  =  new  Graph(kruscal.getVertexNum(),
kruscal.getData());
            kruscal.toFinal(graph);
            assertEquals(kruscal.toStart(graph), "Вы на начале.");
        }

        @Test
        public void test1_check_nextStep(){
            String data_ =  "3 3\n" +
                    "0 1 1\n" +
                    "0 2 2\n" +
                    "1 2 3";
            Kruscal kruscal = new Kruscal(data_);
                     Graph  graph  =  new  Graph(kruscal.getVertexNum(),
kruscal.getData());
             assertEquals(kruscal.nextStep(graph), "Было добавлено ребро
0 - 1");
             assertEquals(kruscal.nextStep(graph), "Было добавлено ребро
0 - 2");
               assertEquals(kruscal.nextStep(graph), "Circle was found:
start Vertex is 1, final is 2");
        }

        @Test
        public void test2_check_nextStep(){
            String data_ =   "8 13\n" +
                    "0 1 1\n" +
                    "1 3 3\n" +
                    "3 2 7\n" +
```

```
                    "2 5 9\n" +
                    "4 5 6\n" +
                    "5 6 4\n" +
                    "6 7 2\n" +
                    "7 0 1\n" +
                    "7 1 13\n" +
                    "0 5 2\n" +
                    "6 3 5\n" +
                    "0 2 3\n" +
                    "1 2 5";
            Kruscal kruscal = new Kruscal(data_);
                     Graph  graph  =  new  Graph(kruscal.getVertexNum(),
kruscal.getData());
            assertEquals(kruscal.nextStep(graph), "Было добавлено ребро
7 - 0");
            assertEquals(kruscal.nextStep(graph), "Было добавлено ребро
0 - 1");
            assertEquals(kruscal.nextStep(graph), "Было добавлено ребро
0 - 5");
            assertEquals(kruscal.nextStep(graph), "Было добавлено ребро
6 - 7");
            assertEquals(kruscal.nextStep(graph), "Было добавлено ребро
0 - 2");
            assertEquals(kruscal.nextStep(graph), "Было добавлено ребро
1 - 3");
              assertEquals(kruscal.nextStep(graph), "Circle was found:
start Vertex is 5, final is 6");
              assertEquals(kruscal.nextStep(graph), "Circle was found:
start Vertex is 1, final is 2");
              assertEquals(kruscal.nextStep(graph), "Circle was found:
start Vertex is 6, final is 3");
            assertEquals(kruscal.nextStep(graph), "Было добавлено ребро
4 - 5");
              assertEquals(kruscal.nextStep(graph), "Circle was found:
start Vertex is 3, final is 2");
              assertEquals(kruscal.nextStep(graph), "Circle was found:
start Vertex is 2, final is 5");
              assertEquals(kruscal.nextStep(graph), "Circle was found:
start Vertex is 7, final is 1");
```

```
    }

    @Test
    public void test1_check_prevStep(){
        String data_ =  "3 3\n" +
                "0 1 1\n" +
                "0 2 2\n" +
                "1 2 3";
        Kruscal kruscal = new Kruscal(data_);
                 Graph  graph  =  new  Graph(kruscal.getVertexNum(),
kruscal.getData());
             assertEquals(kruscal.prevStep(graph), "Возврат на предыдущий
шаг невозможен - Вы на первом шаге");
        //делаем шаг вперед
        kruscal.nextStep(graph);
            assertEquals(kruscal.prevStep(graph), "Ребро 0 - 1 было
удалено из МОД");
        //проходимся до конца алгоритма
        kruscal.toFinal(graph);
           assertEquals(kruscal.prevStep(graph), "На предыдущем шаге
был выявлен цикл");
    }

    @Test
    public void test2_check_prevStep(){
        String data_ =   "8 13\n" +
                "0 1 1\n" +
                "1 3 3\n" +
                "3 2 7\n" +
                "2 5 9\n" +
                "4 5 6\n" +
                "5 6 4\n" +
                "6 7 2\n" +
                "7 0 1\n" +
                "7 1 13\n" +
                "0 5 2\n" +
                "6 3 5\n" +
                "0 2 3\n" +
                "1 2 5";
```

```
            Kruscal kruscal = new Kruscal(data_);
                     Graph  graph  =  new  Graph(kruscal.getVertexNum(),
kruscal.getData());
             assertEquals(kruscal.prevStep(graph), "Возврат на предыдущий
шаг невозможен - Вы на первом шаге");
            //делаем шаг вперед
            kruscal.nextStep(graph);
                assertEquals(kruscal.prevStep(graph), "Ребро 7 - 0 было
удалено из МОД");
            //проходимся до конца алгоритма
            kruscal.toFinal(graph);
              assertEquals(kruscal.prevStep(graph), "На предыдущем шаге
был выявлен цикл");
         }
     }
```