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

ОТЧЕТ

по учебной практике

Тема: Разработка визуализатора алгоритма Краскала на Java

Студент гр. 8383 _____ Колмыков В.Д.

Студент гр. 8383 _____ Степанов В.Д.

Студент гр. 8383 _____ Шишкин И.В.

Руководитель _____ Жангиров Т.Р.

Санкт-Петербург

2020

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Колмыков В.Д. группы 8383

Студент Степанов В.Д. группы 8383

Студент Шишкин И.В. группы 8383

Тема практики: Разработка визуализатора алгоритма Краскала на Java

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: Краскала.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 10.07.2020

Дата защиты отчета: 10.07.2020

Студент _____ Колмыков В.Д.

Студент _____ Степанов В.Д.

Студент _____ Шишкин И.В.

Руководитель _____ Жангиров Т.Р.

АННОТАЦИЯ

В данной работе будет реализован визуализатор алгоритма Краскала на языке программирования Java при помощи библиотеки Java Swing. При этом, приложение будет удовлетворять следующим требованиям: будет реализован графический интерфейс, пошаговая визуализация алгоритма, приложение будет ясным и удобным. Работа выполняется в бригаде. Контроль версий программы осуществляется при помощи системы контроля версий Git. Промежуточные версии программы были согласованы с преподавателем.

SUMMARY

In this work, we will implement a visualizer of the Kruskal algorithm in the Java programming language using the Java Swing library. At the same time, the application will satisfy the following requirements: a graphical interface will be implemented, step-by-step visualization of the algorithm, the application will be clear and convenient. The work is carried out in the team. Version control of the program is carried out using the version control system Git. Intermediate versions of the program were agreed with the teacher.

СОДЕРЖАНИЕ

	Введение	6
1.	Требования к программе	7
1.1.	Исходные требования к программе	7
1.1.1.	Требования к вводу исходных данных	7
1.1.1.	Требования к визуализации	7
1.2.	Уточнение требований после сдачи 1-ой версии	7
2.	План разработки и распределение ролей в бригаде	8
2.1.	План разработки	8
2.2.	Распределение ролей в бригаде	8
3.	Особенности реализации	9
3.1.	Структуры данных	9
3.2.	Основные методы	13
4.	Тестирование	15
4.1	Тестирование интерфейса программы	15
4.2	Тестирование алгоритма	21
	Заключение	27
	Список использованных источников	28

Приложение А. Main.java 29
Приложение Б. Window.java 30
Приложение В. Node.java 39
Приложение Г. AlgorithmLoggingWindow.java 40
Приложение Д. AddEdgeWindow.java 41
Приложение Е. Algorithm.java 44
Приложение Ж. AlgorithmManager.java 45
Приложение З. DSF.java 47
Приложение И. Edge.java 49
Приложение К. KruskalAlgorithm.java 50
Приложение Л. AddEdgeException.java 55
Приложение М. IncoherentGraphException.java 56
Приложение Н. PrevBeforeException.java 57
Приложение О. Circle.java 58
Приложение П. GraphicsPanel.java 60
Приложение Р. Line.java 63
Приложение С. Point.java 65
Приложение Т. CreateEdgeActionListener.java 66
Приложение У. CreateFirstExampleActionListener.java 67
Приложение Ф. CreateNodeActionListener.java 68
Приложение Х. CreateSecondExampleActionListener.java 70
Приложение Ц. CreateThirdExampleActionListener.java 71
Приложение Ч. RestartActionListener.java 72
Приложение Ш. ResultActionListener.java 73
Приложение Щ. RunActionListener.java 74
Приложение Ы. EdgeColor.java 75

ВВЕДЕНИЕ

Цель данной практики заключается в реализации визуализатора алгоритма Краскала. Алгоритм строит минимальное остовное дерево взвешенного связанного неориентированного графа. Алгоритм имеет широкое применение, например, он применяется в проектировании сетей, сегментации изображений.

Программа будет написана на языке программирования Java и визуализирована при помощи библиотеки Java Swing.

Задача данной практики заключается в приобретении навыков работы в команде над проектом, составлении плана разработки, проектировании приложения.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1. Требования к вводу исходных данных

Входные данные могут вноситься при работе в самом приложении или из файла.

1.1.2. Требования к визуализации

Граф и работа алгоритма должны быть визуализированы в программе. Визуализация алгоритма должна быть пошаговой, шаги не должны быть крупными.

1.2. Уточнение требований после сдачи 1-ой версии

Использовать паттерн «посредник» для связи класса окна с классом графической панели и класса окна с классом алгоритма.

Реализовать окно с логированием промежуточных данных.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

29.06 – Распределение ролей в бригаде и создание плана разработки.

30.06 и 01.07 – Выполнение вводного задания.

02.07 – Создание прототипа GUI.

03.07 – Создание UML-диаграмм (классов и вариантов использования).

04.07 – Согласование интерфейса и плана разработки с преподавателем.

05.07 – Начало создания интерфейса при помощи Java Swing.

06.07 – Создание возможности ввода графа, реализация примеров графа.

07.07 – Согласование первой итерации с преподавателем.

08.07 – Реализация алгоритма Краскала.

09.07 – Реализация визуализации алгоритма.

10.07 – Сдача второй (финальной) итерации преподавателю.

2.2. Распределение ролей в бригаде

Колмыков Владислав Денисович – ответственный за визуализацию алгоритма.

Степанов Владислав Денисович – ответственный за интерфейс.

Шишкин Иван Викторович – ответственный за алгоритмизацию.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

```
public class Node {
    private int number;
    private Map<Integer, Edge> edgesMap;

    public Node(int number);
    public int getNumber();
    public void addEdge(int number, Edge edge);
    public Map<Integer, Edge> getEdgesMap();
}
```

Класс Node реализует вершины графа.

Поля: number – номер вершины в графе; edgesMap – карта (номер вершины – ребро от текущей вершины к вершине с номером, указанным в карте).

Методы: конструктор принимает на вход номер вершины и записывает его в поле number. Методы getNumber и getEdgesMap возвращают соответствующие значения number и edgesMap. Метод addEdge на вход принимает номер вершины, к которому будет идти ребро, и само ребро.

```
public class Edge implements Comparable<Edge> {
    private int from;
    private int to;
    private int capacity;
    Color color = Color.BLACK;

    public Edge(int from, int to, int capacity);
    @Override
    public int compareTo(Edge o);
    public int getCapacity ();
    public int getFrom ();
```

```
    public int getTo ();
    public Color getColor();
    public void setColor(Color color);
}
```

Класс Edge реализует класс ребра.

Поля: from – номер вершины, из которой следует ребро; to – номер вершины, в которое идет ребро; capacity – вес ребра; color – цвет ребра.

Методы: в конструктор принимаются значения from, to, capacity, и записываются в поля. Компаратор compareTo – нужен для сортировки ребер в порядке возрастания их веса. Гетеры getCapacity, getFrom, getTo, getColor возвращают соответствующие значения.

```
public class DSF {
    int[] set;
    int[] rnk;
    int size = 0;
    ArrayList<ArrayList<Integer>> prevSet;
    ArrayList<ArrayList<Integer>> prevRnk;

    DSF(int size);
    int set(int x);
    boolean union(int u, int v);
    void unUnionForPrev();
}
```

Класс DSF реализует класс системы непересекающихся множеств.

Поля: массив set – хранит для каждой вершины номер ее множества; массив rnk – хранит для каждой вершины ее ранг; size – размер этих массивов, т.е. количество вершин в графе; списки prevSet и prevRnk нужны для отката алгоритма на один шаг, чтобы разъединить вершины в графе.

Методы: в конструктор подается количество вершин size, выделяется память для массивов set и rnk, а также циклом значения в массиве set меняются: первой вершине соответствует первое множество, второй вершине

соответствует второе множество и т.д. Метод set возвращает множество, которому принадлежит x. Метод union производит слияние вершин, если они лежат в разных множествах и возвращает true, если же вершины лежат в одинаковых множествах, то возвращает false (это означает, что при слиянии этих вершин в графе будут циклы). Заключительный метод unUnionForPrev производит отмену для предыдущего слияния, чтобы у алгоритма была возможность откатиться назад. Для этого метода и нужны списки prevSet и prevRnk.

```
public class KruskalAlgorithm extends Algorithm {
    private int nodesNumber = 0;
    private Node[] nodes = new Node[0];
    private ArrayList<Edge> edges = new ArrayList<>();
    private DSF dsf;
    private Edge currentEdge;
    private int currentEdgeIndex;
    private int minMSTWeight = 0;

    public                    KruskalAlgorithm(AlgorithmManager
algorithmManager);
    void setNumOfNodes(int num);
    void addEdge(int number1, int number2, int capacity);
    void run();
    boolean step();
    void chooseEdge();
    void handleEdge();
    int getIntermediateResult ();
    void getResult();
    void restart();
    boolean isValidGraph();
    void prev();
    void rechooseEdgeForPrev();
    rehandleEdgeForPrev();
```

```
}
```

Класс KruskalAlgorithm реализует класс алгоритма Краскала.

Поля: nodesNumber – количество вершин в графе; nodes – массив вершин; edges – список ребер; dsf – переменная типа DSF, используется для доступа к методам класса DSF; currentEdge – текущее ребро; currentEdgeIndex – индекс текущего ребра в списке ребер edges; minMSTWeight – минимальный вес остовного дерева.

Методы: В конструктор подается переменная типа AlgorithmManager, которая записывается в поле класса. Метод setNumOfNodes на вход получает количество вершин num, записывает это значение в поле nodesNumber, и выделяет память для полей nodes и edges. Метод addEdge создает ребро из вершины number1 в вершину number2 с весом capacity. Метод run производит запуск алгоритма Краскала (придает переменным нужные значения, производит сортировку ребер). Метод step делает шаг алгоритма, возвращает false, если алгоритм закончил свою работу (индекс текущего ребра становится равен размеру списка edges) и true в любом другом случае. Методы chooseEdge и handleEdge – 2 возможных шага, используются в методе step. step вызывает chooseEdge, если текущее ребро не выбрано, метод chooseEdge берет следующее ребро из списка edges и присваивает ему желтый цвет. step вызывает handleEdge, если текущее ребро уже было выбрано методом chooseEdge. В handleEdge вызывается метод union класса DSF, которому подаются 2 вершины текущего ребра. Если слияние произошло, то цвет текущего ребра меняется на зеленый, если же нет – меняется на красный. Гетер getIntermediateResult возвращает текущий вес остовного дерева. Функция getResult выдает сразу результат алгоритма без пошагового режима. Метод restart производит запуск алгоритм заново. Метод isValid производит проверку связности графа. Функция prev делает шаг алгоритма назад, она вызывает еще 2 метода rechooseEdgeForPrev и rehandleEdgeForPrev. Метод rechooseEdgeForPrev перекрашивает желтое ребро в черное. Метод rehandleEdgeForPrev

перекрашивает зеленое или красное ребро в желтое, а также вызывает метод класса DSF unUnionForPrev.

3.2. Основные методы

3.2.1. Конструктор Window()

В этом методе создается весь интерфейс. Размер окна – 1600x1000, открывается по середине экрана. Создается меню-бар menuBar, в котором 3 пункта меню: createMenu – кнопка create graph, createExampleMenu – кнопка examples, createAlgorithmMenu – кнопка Algorithm, graphicsPanel – панель, на которой происходит рисование и визуализация алгоритма.

3.2.2. Метод Window.runAlgorithm()

В этом методе происходит запуск алгоритма через посредника AlgorithmManager, который, в свою очередь, вызывает метод KruskalAlgorithm.run().

3.2.3. Метод void Window.addEdge(int number1, int number2, int capacity)

В этом методе происходит добавление ребра через посредника AlgorithmManager. В нем происходит обработка возможных ошибок при вводе данных.

3.2.4. Метод Window.setNumberOfNodes(int num)

Через посредника AlgorithmManager создает граф с num вершинами.

3.2.5. Метод GraphGraphicsManager.drawGraph(Node[] nodes)

Вызывает необходимые методы у graphicsPanel класса GraphicsPanel, чтобы нарисовать граф, вершины которого находятся в массиве nodes.

3.2.6. Метод GraphGraphicsManager.drawNode(int number, int quantity)

Вызывает необходимые методы у graphicsPanel класса GraphicsPanel для рисования вершины.

3.2.7. Метод GraphGraphicsManager.drawEdge(int number1, int number2, int value, int quantity, Color color)

Вызывает необходимые методы у graphicsPanel класса GraphicsPanel, чтобы нарисовать ребро из вершины number1 в number2 с весом value, цветом color, quantity – количество вершин, необходимо для определения местоположения ребра.

3.2.8. Метод GraphGraphicsManager.redrawGraph()

Вызывает необходимые методы у graphicsPanel класса GraphicsPanel, чтобы перерисовать кешированный граф.

3.2.9. Конструктор AlgorithmLoggingWindow(Window window)

Производит логирование программы.

3.2.10. Метод Window.log(String str)

Вызывает метод AlgorithmLoggingWindow.printCondition, чтобы логировать действия.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование интерфейса программы

На рис. 1 указано начальное состояние программы при ее запуске. На верхней панели расположены 3 кнопки: create graph, examples и algorithm. При нажатии на create graph можно осуществить 3 действия: create new graph – создать новый граф (рис. 2), add edge – добавить ребро в созданный граф (рис. 4) и from file – ввод графа из файла (рис. 3).

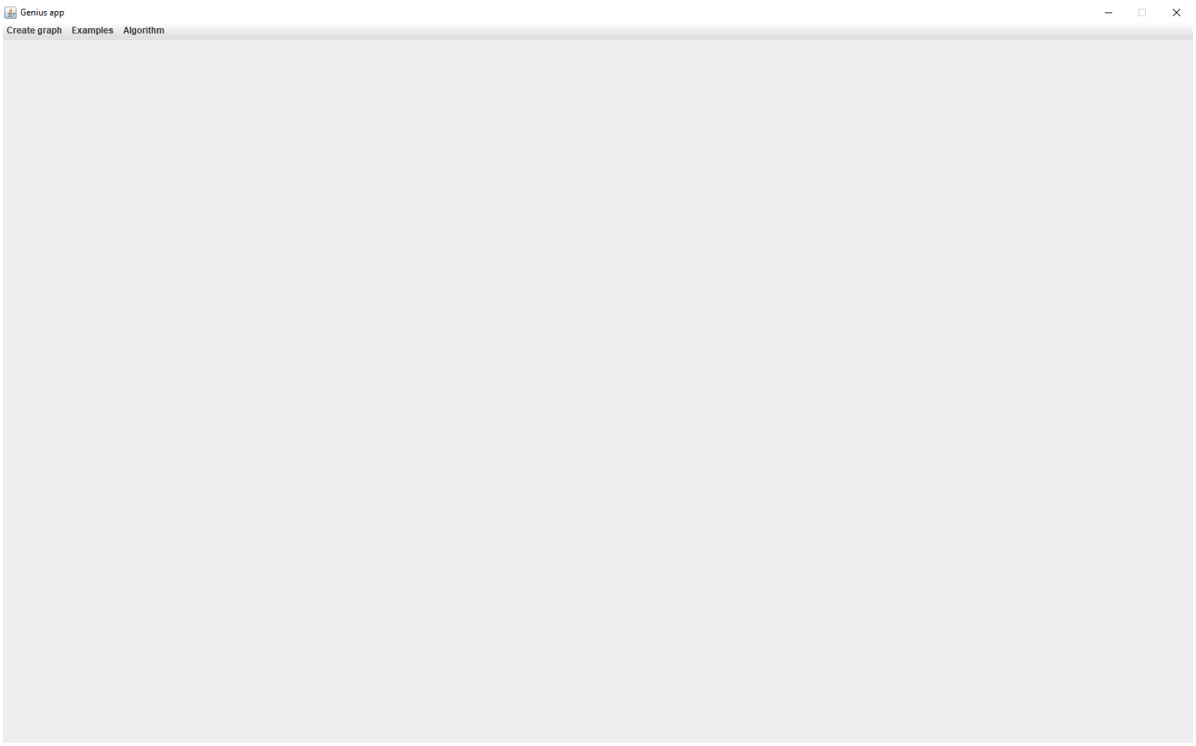

Рисунок 1 – начальное состояние программы

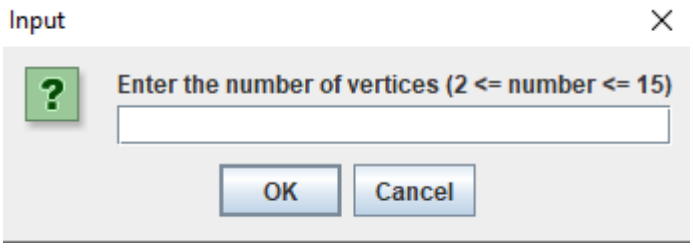

Рисунок 2 – Окно create new graph

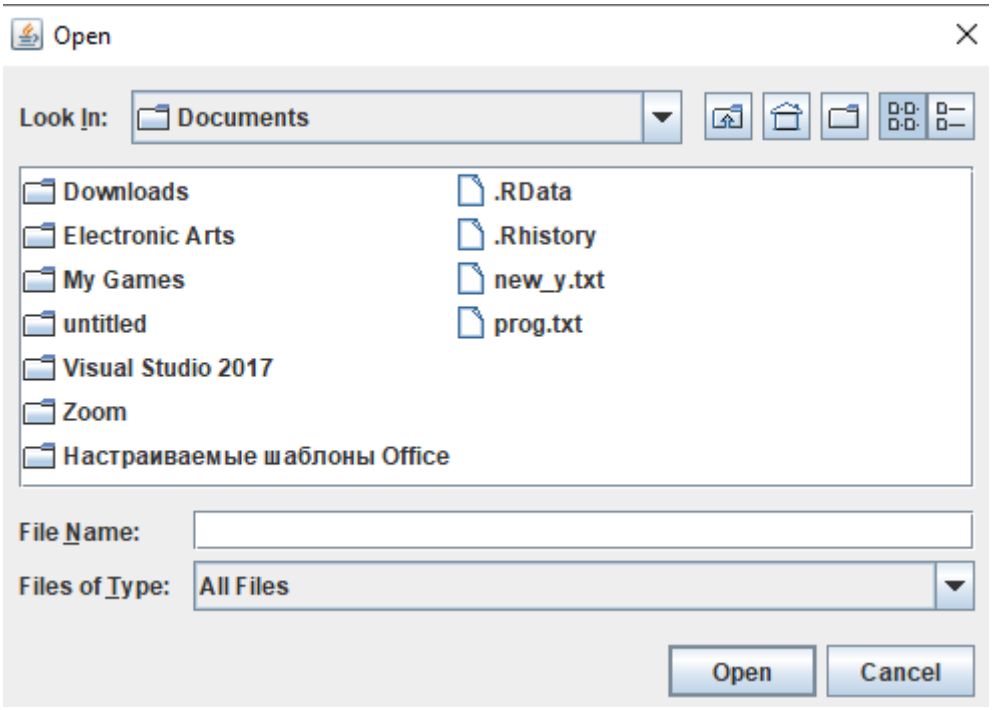

Рисунок 3 – Окно ввода графа from file

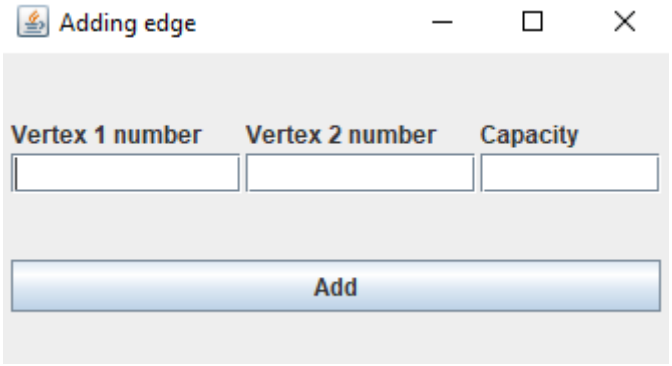

Рисунок 4 – Окно add edge

На рис. 5 указан пример, что будет, если создать граф из 3 вершин и добавить ребра 0-1 с весом 1, 0-2 с весом 2 и 1-2 с весом 5.

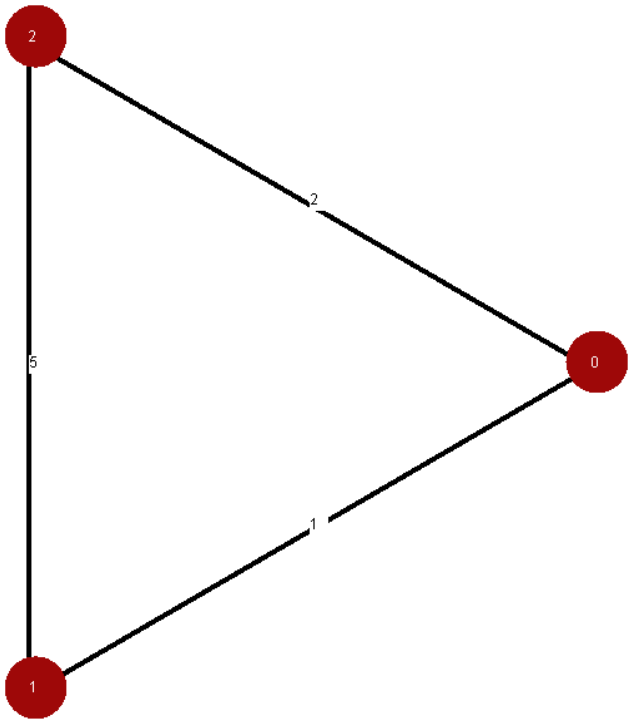

Рисунок 5 – Визуализация графа

После нажатия кнопки Algorithm-Run (рис. 6) происходит запуск алгоритма, все кнопки на верхней панели блокируются, становится видимой панель с права с кнопками step – переход на следующий шаг, prev – переход на предыдущий шаг, result – вывод результата, restart – перезапуск.

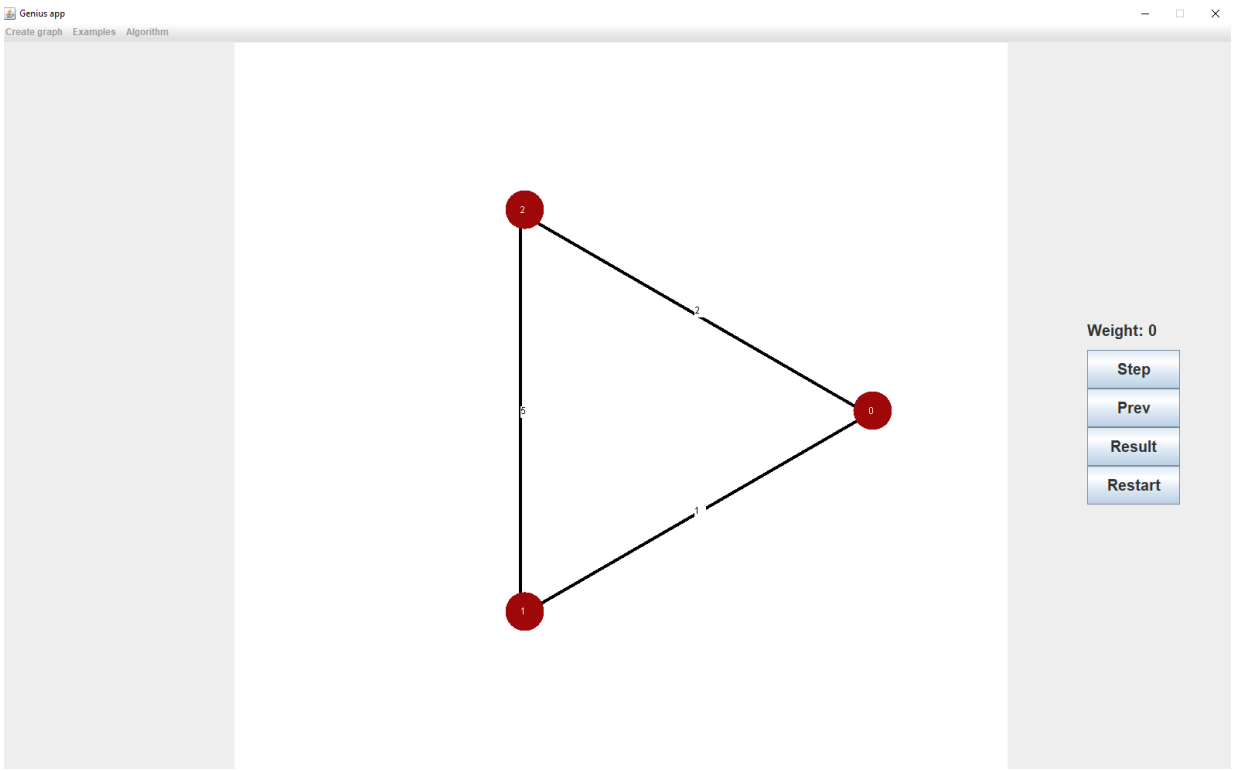

Рисунок 6 – После нажатия кнопки Algorithm-Run

На рис. 7 показано состояние программы после нажатия кнопки result. Кнопки на верхней панели вновь можно нажать, а кнопки справа блокируются.

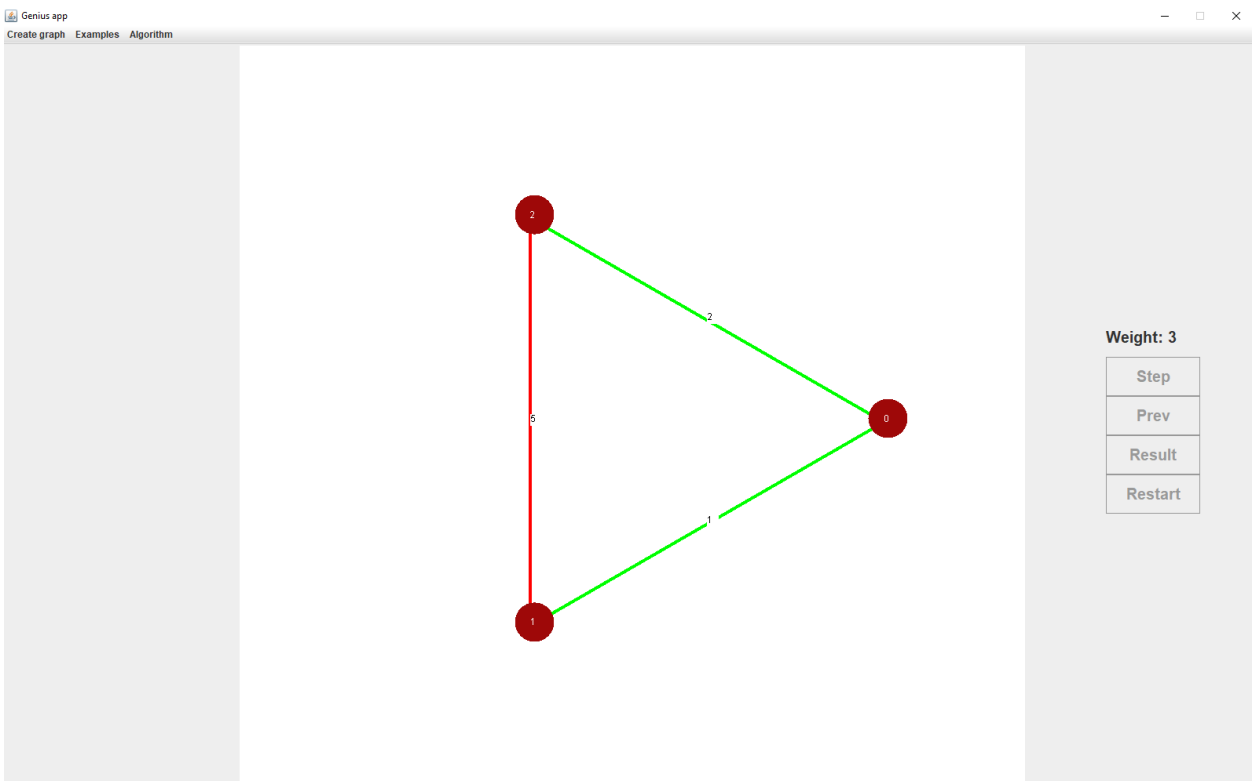

Рисунок 7 – После нажатия кнопки Result

Чтобы не вводить каждый раз новый граф, предусмотрена возможность использования примеров. Всего их реализовано 3. На рис. 7 показан 3-й пример графа.

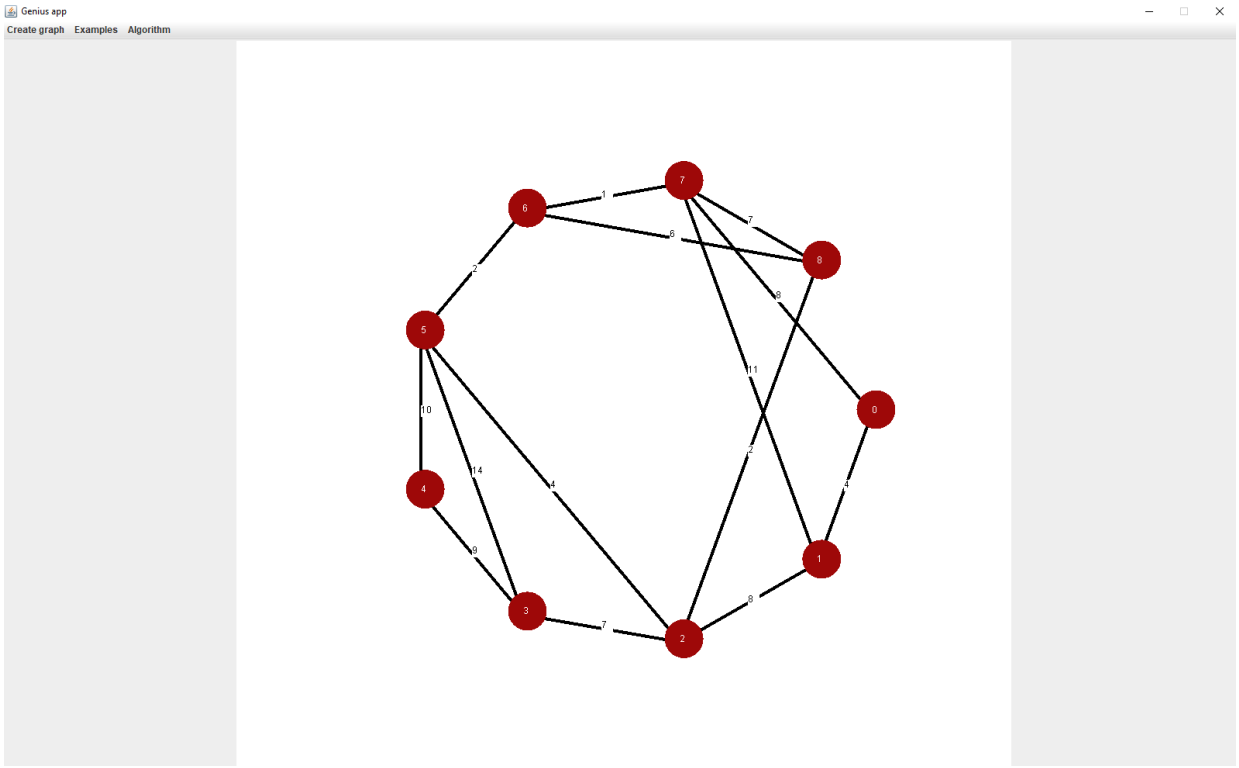

Рисунок 8 – Пример графа номер 3

В окне create new graph учтено, что количество вершин в графе должно быть больше одного, но меньше 16. Примеры неправильного ввода в это окно представлены на рис. 9 и рис. 10.

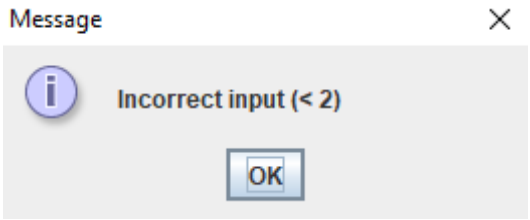

Рисунок 9 – Если в окно create new graph ввести значение -5

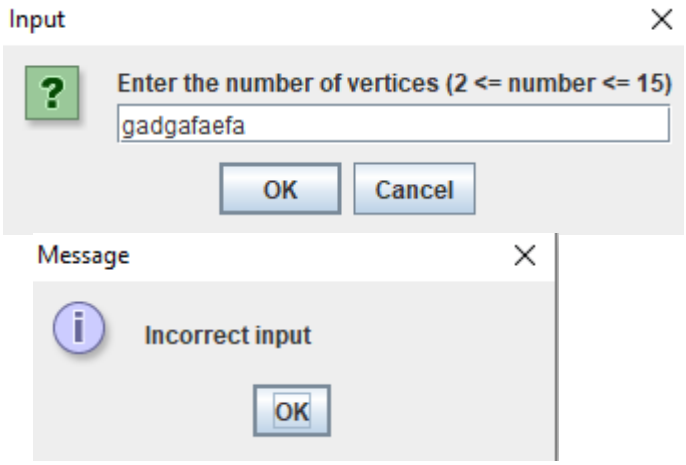

Рисунок 10 – Если в окно create new graph ввести буквы

В окне add edge учтено, что номера вершин не могут быть меньше 0 и больше количества вершин в графе, а значение capacity должно быть больше 0. На рис. 11, 12, 13, 14 приведены примеры ошибок (используется граф из 3-х вершин).

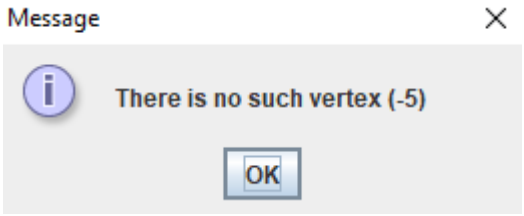

Рисунок 11 – Если в окно add edge ввести -5; 1; 5

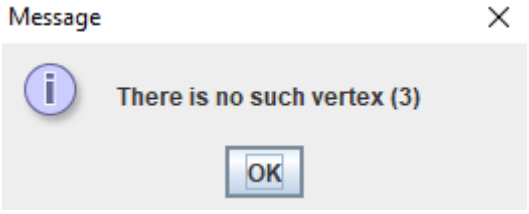

Рисунок 12 – Если в окно add edge ввести 0; 3; 3

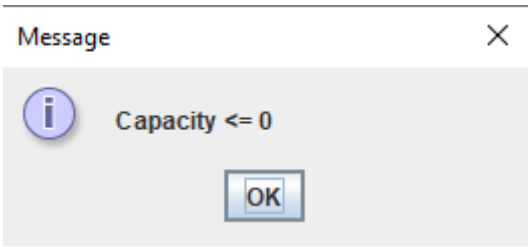

Рисунок 13 – Если в окно add edge ввести capacity = -1

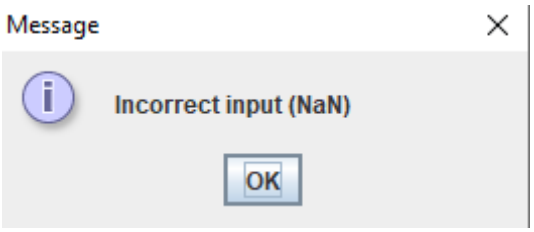

Рисунок 14 – Если в окне add edge ввести в capacity буквы

4.2. Тестирование алгоритма

На рис. 15-19 представлена работа алгоритма. На рис. 15 представлен вводимый граф.

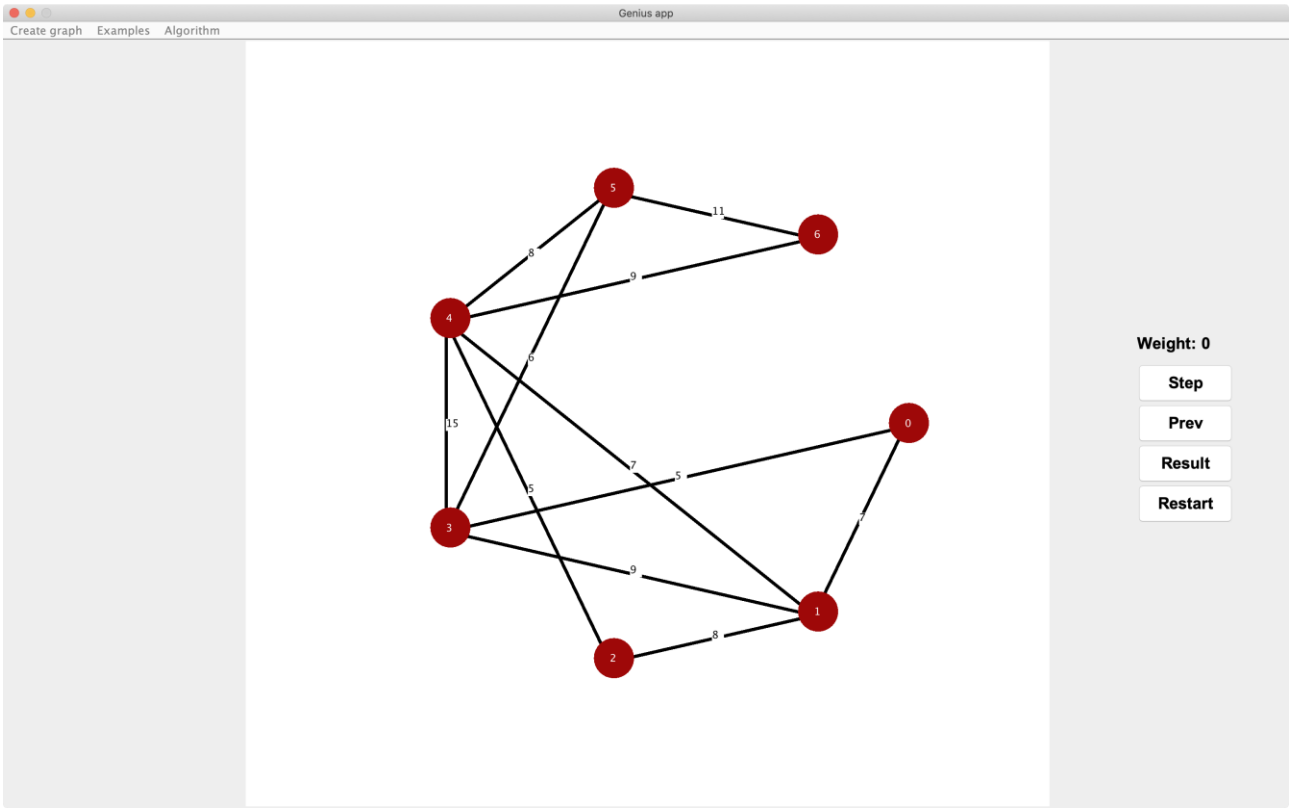

Рисунок 15 – Исходный граф

Когда ребро обрабатывается оно перекрашивается в желтый цвет, как показано на рис. 16.

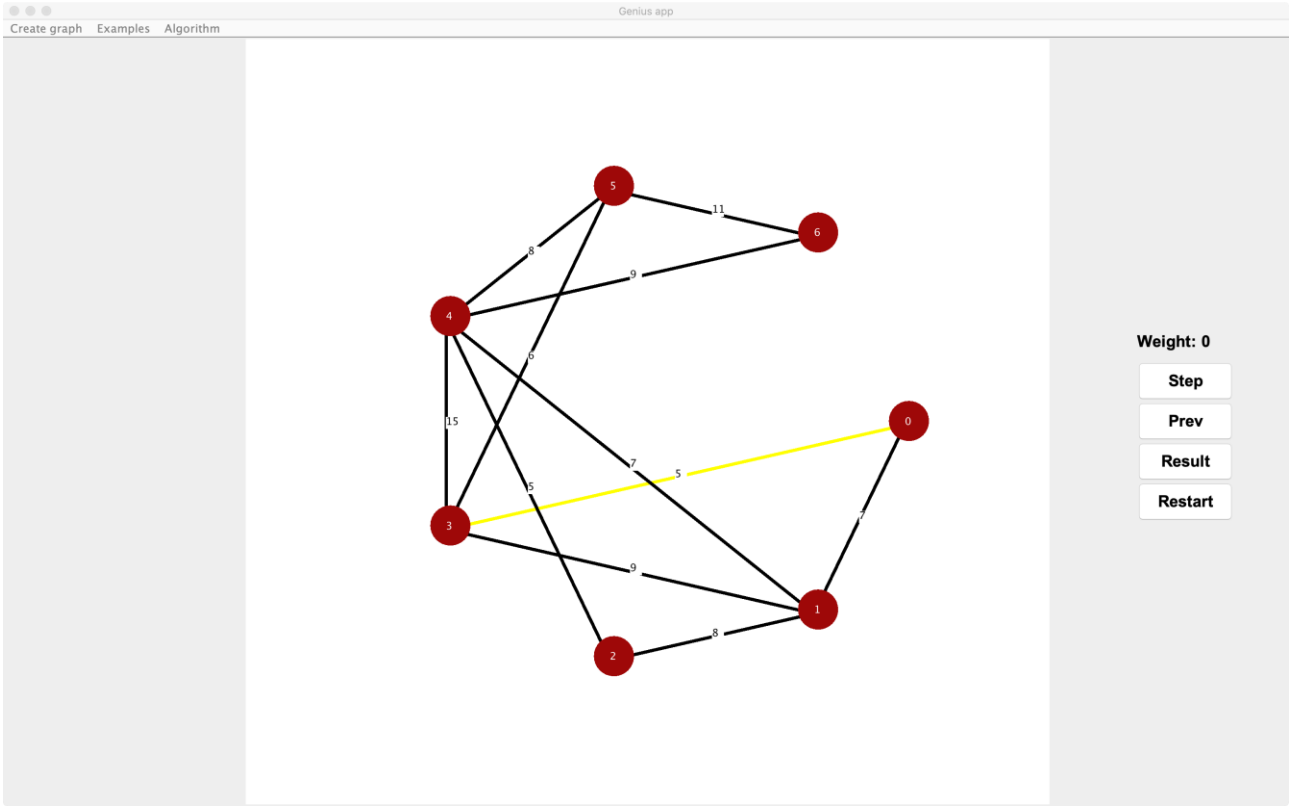

Рисунок 16 – Выбор обрабатываемого ребра

Если ребро принадлежит минимальному остовному дереву, то оно окрашивается в зеленый цвет, как показано на рис. 17.

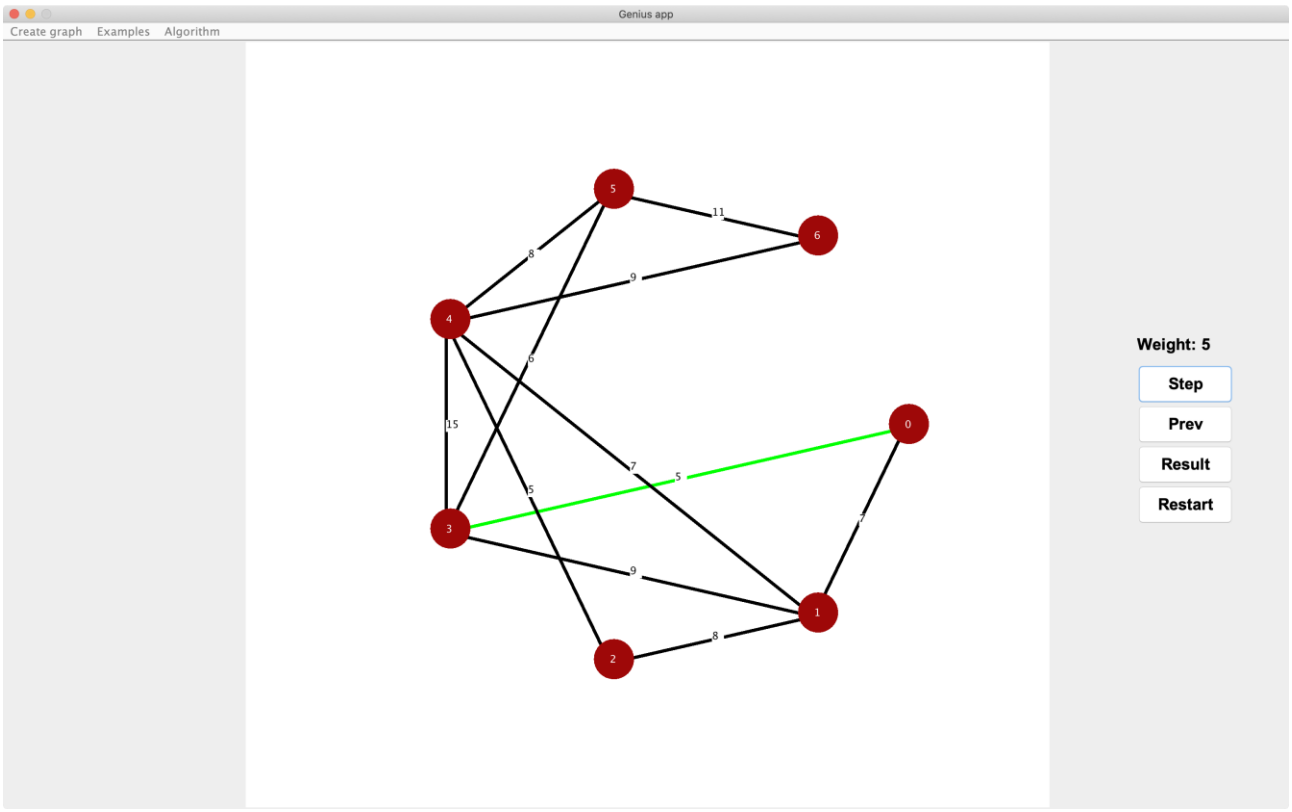

Рисунок 17 – Выбор ребра, принадлежащего минимальному остовному дереву

Если же не принадлежит минимальному остовному дереву, то оно окрашивается в красный цвет, как показано на рис. 18.

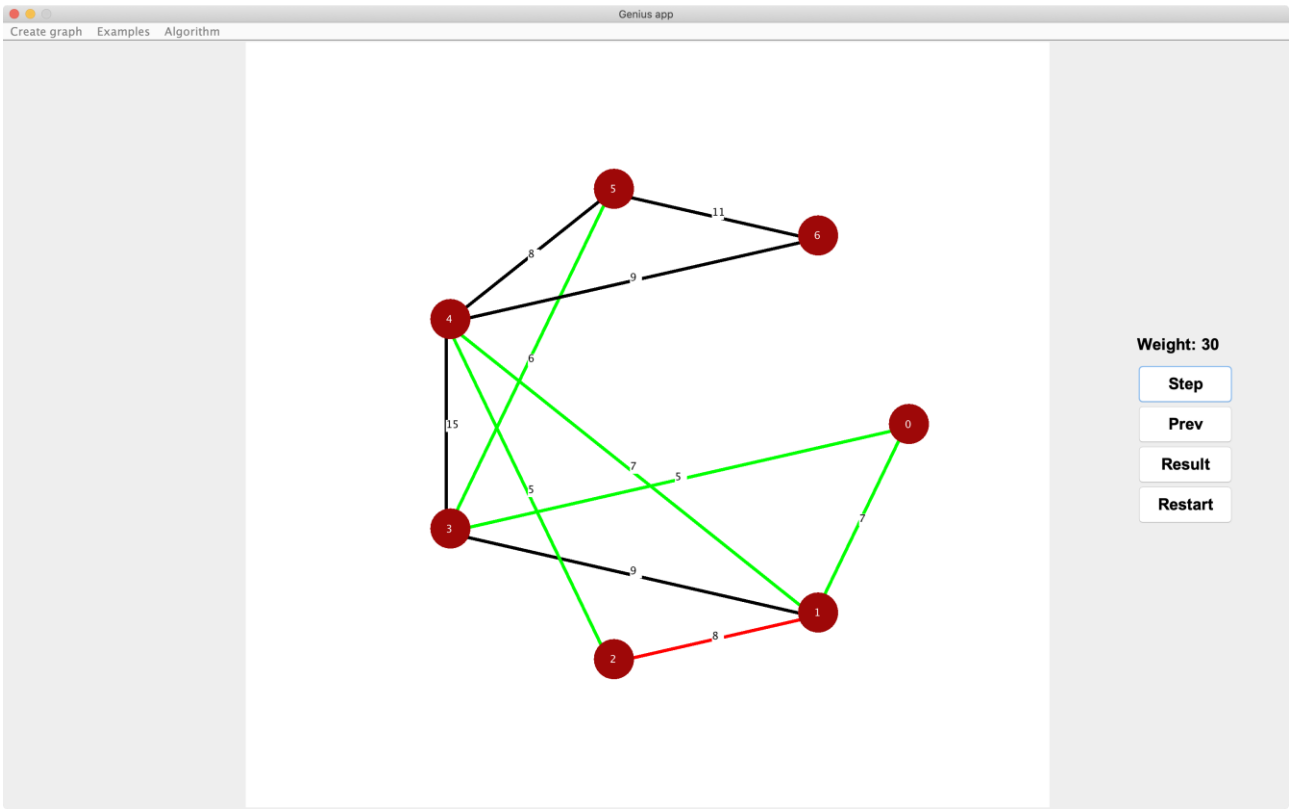

Рисунок 18 – Выбор ребра, не принадлежащего минимальному остовному дереву

На рис. 19 представлен результат работы алгоритма для введенного графа.

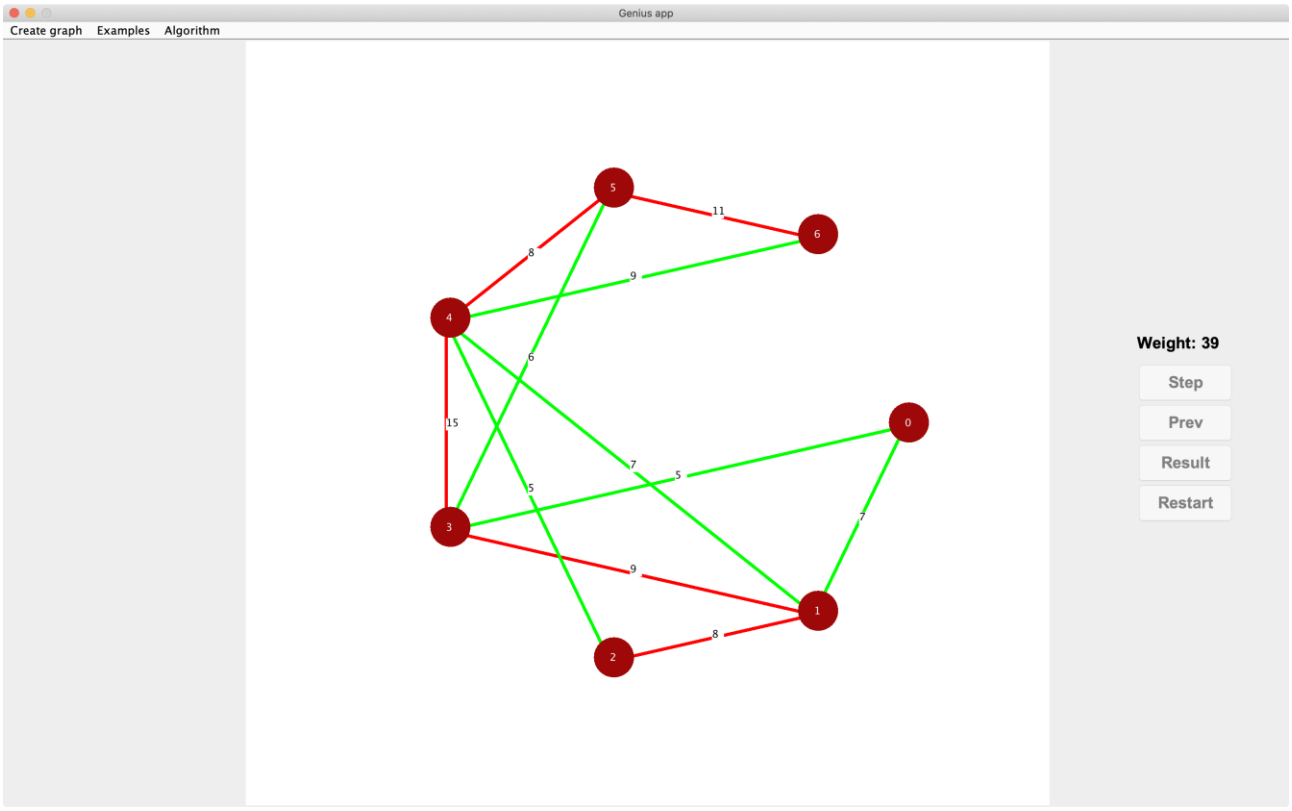

Рисунок 19 – Результат работы алгоритма

На рис. 20 – 22 представлено тестирование программы.

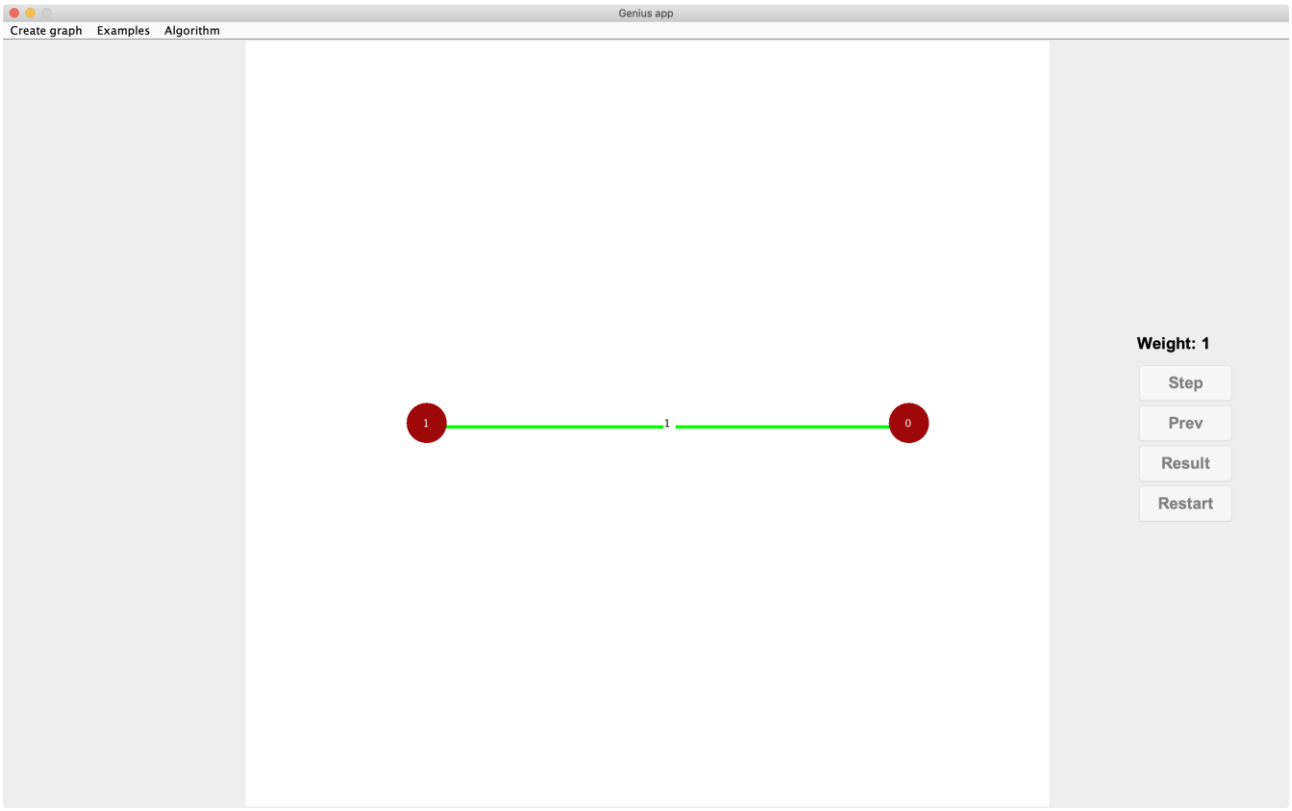

Рисунок 20 – Тест 1

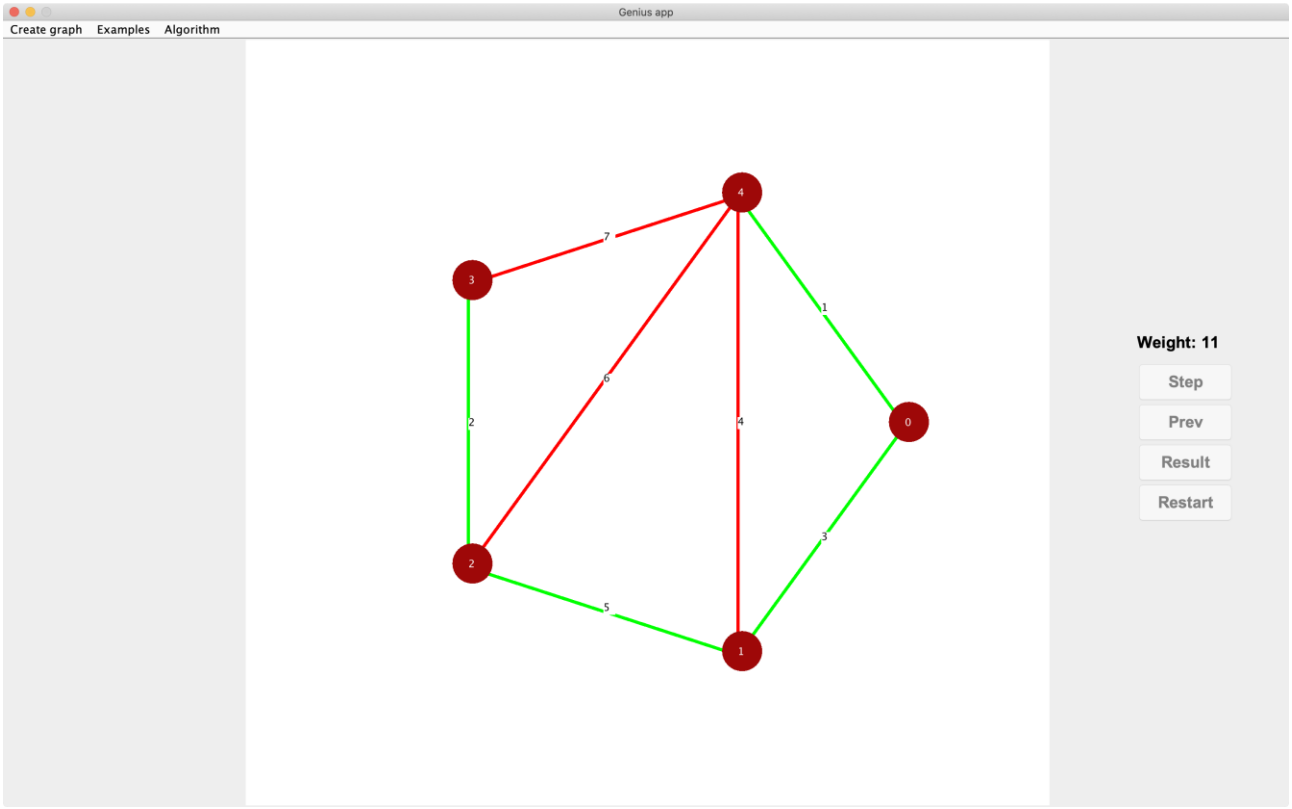

Рисунок 21 – Тест 2 (пример №2)

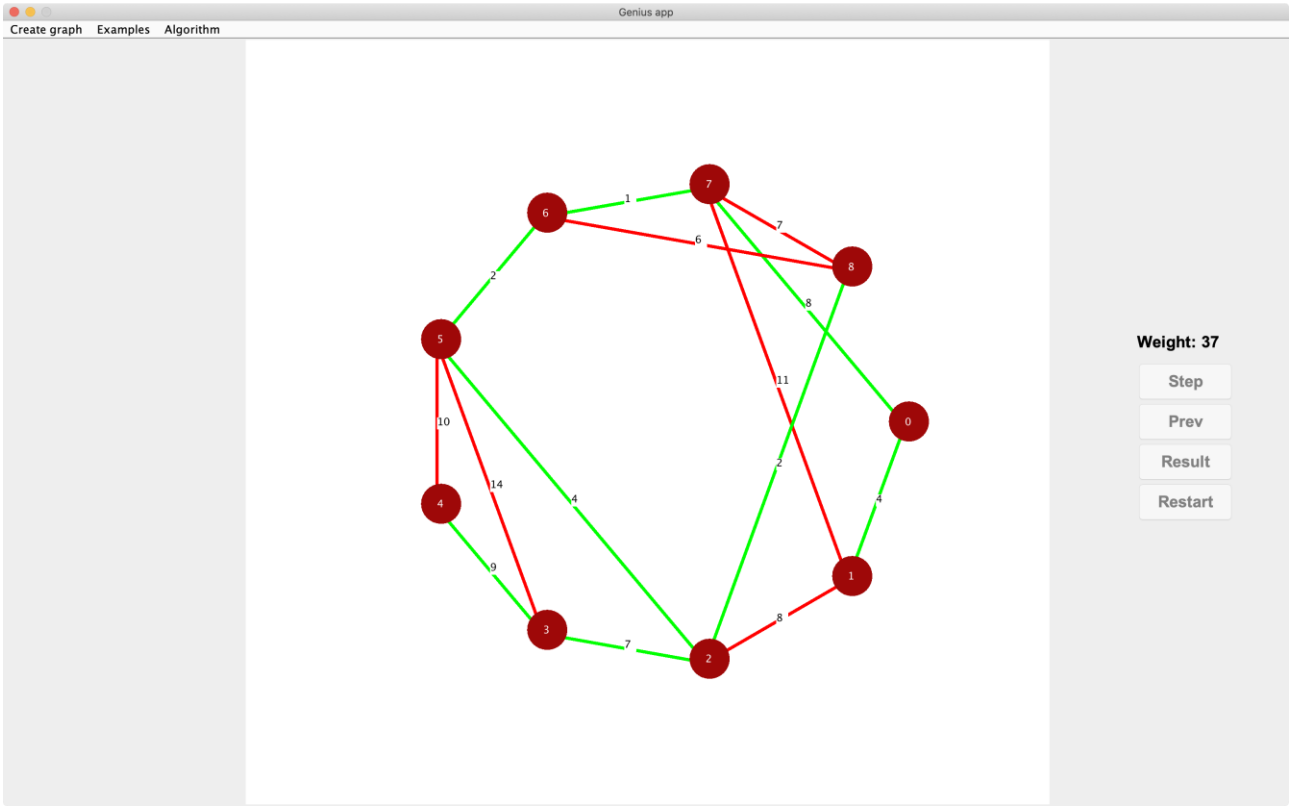

Рисунок 22 – Тест 3 (пример №3)

ЗАКЛЮЧЕНИЕ

Был реализован алгоритм Краскала на языке Java и визуализирован при помощи библиотеки Java Swing. Были приобретены навыки работы над проектом в бригаде.

Был освоен новый язык программирования Java, улучшены навыки в создании десктопных приложений, UML-диаграмм и плана разработки. Получены навыки работы с системой контроля версий Git.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Свободная энциклопедия // Wikipedia. URL: https://ru.wikipedia.org/wiki/Алгоритм_Краскала#:~:text=Алгоритм%20Краскала%20%20эффективный%20алгоритм%20построения,дерева%20взвешенного%20связного%20неориентированного%20графа.&text=Алгоритм%20описан%20Дж озефом%20Краскалом,Отакаром%20Борувкой%20в%201926%20году. (дата обращения: 08.07.2020).

2. Документация Java Swing // docs.oracle. URL: https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html (дата обращения: 05.07.2020).

ПРИЛОЖЕНИЕ А

MAIN.JAVA

```
package kolmykov_shishkin_stepanov;

public class Main {
    public static void main(String[] agrs) {
        Window window = new Window();
        window.setVisible(true);
    }
}
```

ПРИЛОЖЕНИЕ Б

WINDOW.JAVA

```
package kolmykov_shishkin_stepanov;

import kolmykov_shishkin_stepanov.algorithm.AlgorithmManager;
import kolmykov_shishkin_stepanov.algorithm.KruskalAlgorithm;
import kolmykov_shishkin_stepanov.exceptions.AddEdgeException;
import kolmykov_shishkin_stepanov.graphics.GraphicsPanel;
import kolmykov_shishkin_stepanov.listeners.*;

import javax.swing.*;
import javax.swing.event.AncestorEvent;
import javax.swing.event.AncestorListener;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;

public class Window extends JFrame {
    private JMenuBar menuBar;
    private JMenu createMenu;
    private JMenuItem createNodesMenuItem;
    private JMenuItem createEdgeMenuItem;
    private GraphicsPanel graphicsPanel;

    private JMenu createExampleMenu;
    private JMenuItem createFirstExampleItem;
    private JMenuItem createSecondExampleItem;
    private JMenuItem createThirdExampleItem;

    private JMenu createAlgorithmMenu;                              //
Algorithm
    private JMenuItem createRunItem;                                //
Algorithm -> Run

    private JButton stepButton;
    private JButton showResultButton;
    private JButton restartButton;
    private JButton prevButton;

    private JLabel weightOfMST;
```

```
    private int nodesQuantity;

    private AlgorithmLoggingWindow alw;

    private AlgorithmManager algorithmManager;


    public Window() {
        super("Genius app");                                        //
создание формы
        this.setSize(1600, 1000);                              // выбор
размера формы
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);     //
выбор действия при закрытии формы
        this.setLocationRelativeTo(null);                          //
открытие формы посередине экрана
        this.setLayout(new GridBagLayout());


        menuBar = new JMenuBar();                    // создание и
установка меню-бара
        this.setJMenuBar(menuBar);

        createMenu      =      new      JMenu("Create      graph");
// создание пункта меню
        menuBar.add(createMenu);
// добавление пункта
        createNodesMenuItem = new JMenuItem("Create new graph");
// создание подпункта меню
        createNodesMenuItem.addActionListener(new
CreateNodeActionListener(this));     // добавление метода в пункт
        createMenu.add(createNodesMenuItem);
// добавление подпукнкта
        createEdgeMenuItem = new JMenuItem("Add edge");
        createEdgeMenuItem.addActionListener(new
CreateEdgeActionListener(this));
        createMenu.add(createEdgeMenuItem);
        createEdgeMenuItem.setEnabled(false);
// блокировка кнопки "Add edge"

        createExampleMenu = new JMenu("Examples");
        menuBar.add(createExampleMenu);
```

```
        createFirstExampleItem    =    new    JMenuItem("First");
// first example
        createFirstExampleItem.addActionListener(new
CreateFirstExampleActionListener(this));
        createExampleMenu.add(createFirstExampleItem);

        createSecondExampleItem    =    new    JMenuItem("Second");
// second example
        createSecondExampleItem.addActionListener(new
CreateSecondExampleActionListener(this));
        createExampleMenu.add(createSecondExampleItem);

        createThirdExampleItem    =    new    JMenuItem("Third");
//  third example
        createThirdExampleItem.addActionListener(new
CreateThirdExampleActionListener(this));
        createExampleMenu.add(createThirdExampleItem);

        alw = new AlgorithmLoggingWindow(this);
        alw.setVisible(true);

        JPanel buttonsPanel = new JPanel();
        buttonsPanel.setLayout(new GridLayout(0,1));
        add(buttonsPanel,new  GridBagConstraints(1,  0,  1,  1,  0.1,
0.4,
                GridBagConstraints.CENTER,
                GridBagConstraints.CENTER,
                new Insets(2, 2, 2, 2),
                0, 0) );

        Font bigFontTR = new Font("Arial", Font.BOLD, 20);

        weightOfMST = new JLabel("Weight: 0");
        weightOfMST.setFont(bigFontTR);
        buttonsPanel.add(weightOfMST);
        weightOfMST.setVisible(false);

        stepButton = new JButton("Step");
        stepButton.setMaximumSize(new Dimension(20, 20));
        stepButton.setFont(bigFontTR);
        buttonsPanel.add(stepButton);
        stepButton.setVisible(false);
        stepButton.setEnabled(false);
        stepButton.addActionListener(new ActionListener() {
            @Override
```

```
            public void actionPerformed(ActionEvent e) {
                boolean isRun = algorithmManager.makeStep();
                redraw();
                weightOfMST.setText("Weight:          "          +
algorithmManager.getIntermediateResultOfAlgorithm());
                if (!isRun) {
                    changeEnableOfResultButton();
                }
            }
        });

        prevButton = new JButton("Prev");
        prevButton.setMaximumSize(new Dimension(20, 20));
        prevButton.setFont(bigFontTR);
        prevButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    algorithmManager.makePrev();
                    redraw();
                    weightOfMST.setText("Weight:        "        +
algorithmManager.getIntermediateResultOfAlgorithm());
                }catch (Exception ex) {

JOptionPane.showMessageDialog(((JButton)e.getSource()),
ex.getMessage());
                }
            }
        });
        buttonsPanel.add(prevButton);
        prevButton.setVisible(false);
        prevButton.setEnabled(false);

        showResultButton = new JButton("Result");
        showResultButton.setPreferredSize(new Dimension(120, 50));
        showResultButton.setFont(bigFontTR);
        //showResultButton.addActionListener(new
ResultActionListener(this));
        showResultButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                algorithmManager.getResultOfAlgorithm();
                changeEnableOfResultButton();
                redraw();
```

```
                weightOfMST.setText("Weight:           "           +
algorithmManager.getIntermediateResultOfAlgorithm());
                log("Algorithm finished");
            }
        });
        buttonsPanel.add(showResultButton);
        showResultButton.setVisible(false);
        showResultButton.setEnabled(false);

        restartButton = new JButton("Restart");
        restartButton.setPreferredSize(new Dimension(120, 50));
        restartButton.setFont(bigFontTR);
        restartButton.addActionListener(new
RestartActionListener(this));
        buttonsPanel.add(restartButton);
        restartButton.setVisible(false);
        restartButton.setEnabled(false);
        restartButton.addAncestorListener(new AncestorListener() {
            @Override
            public void ancestorAdded(AncestorEvent event) {
                redraw();
            }

            @Override
            public void ancestorRemoved(AncestorEvent event) {
                redraw();
            }

            @Override
            public void ancestorMoved(AncestorEvent event) {
                redraw();
            }
        });
        restartButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                algorithmManager.restartAlgorithm();
                redraw();
                weightOfMST.setText("Weight: 0");
                alw.printCondition("Algorithm restarted");
            }
        });

        createAlgorithmMenu = new JMenu("Algorithm");
        menuBar.add(createAlgorithmMenu);
```

```
        createRunItem = new JMenuItem("Run");
        createRunItem.setEnabled(false);
        createRunItem.addActionListener(new
RunActionListener(this));
        createAlgorithmMenu.add(createRunItem);

        graphicsPanel = new GraphicsPanel();
        this.add(graphicsPanel,
                new GridBagConstraints(0, 0, 1, 1, 1, 1,
                        GridBagConstraints.NORTH,
                        GridBagConstraints.BOTH,
                        new Insets(2, 2, 2, 2),
                        0, 0));

        algorithmManager = new AlgorithmManager(this);
        algorithmManager.setAlgorithm(new
KruskalAlgorithm(algorithmManager));

        addComponentListener(new ComponentAdapter() {
            @Override
            public void componentResized(ComponentEvent e) {
                super.componentResized(e);
                redraw();
            }

            @Override
            public void componentMoved(ComponentEvent e) {
                super.componentMoved(e);
                redraw();
            }

            @Override
            public void componentShown(ComponentEvent e) {
                super.componentShown(e);
                redraw();
            }

            @Override
            public void componentHidden(ComponentEvent e) {
                super.componentHidden(e);
                redraw();
            }
        });

        setResizable(false);
```

```
        setVisible(true);     // отображение формы
    }

    public void addEdge(int number1, int number2, int capacity) {
        try {
            algorithmManager.addEdge(number1, number2, capacity);
            log("Added edge " + number1 + " - " + number2 + " with
capacity " + capacity);
        }
        catch (AddEdgeException eex){
            JOptionPane.showMessageDialog(this, eex.getMessage());
            log("Error: " + eex.getMessage());
        }

    }

    public void setNumberOfNodes(int num) {
        nodesQuantity = num;
        algorithmManager.setNumOfNodes(num);
        makeInvisibleAlgoButtons();
    }

    public void changeEnableOfCreateMenu() {     // "Create new
graph" – нельзя, "Add edge" – можно, "Examples" - нельзя
        createNodesMenuItem.setEnabled(false);
        createEdgeMenuItem.setEnabled(true);
        createExampleMenu.setEnabled(true);
        createRunItem.setEnabled(true);
    }


    public void changeEnableOfRunAlgButton() {
        createAlgorithmMenu.setEnabled(false);
        createMenu.setEnabled(false);
        createExampleMenu.setEnabled(false);

        weightOfMST.setText("Weight: 0");
        stepButton.setEnabled(true);
        prevButton.setEnabled(true);
        showResultButton.setEnabled(true);
        restartButton.setEnabled(true);
        weightOfMST.setVisible(true);
        prevButton.setVisible(true);
        stepButton.setVisible(true);
        showResultButton.setVisible(true);
```

```
        restartButton.setVisible(true);
    }

    public void changeEnableOfExample(){
        createNodesMenuItem.setEnabled(true);
        createEdgeMenuItem.setEnabled(false);
        createRunItem.setEnabled(true);
    }

    public void changeEnableOfResultButton() {
        createAlgorithmMenu.setEnabled(true);
        createMenu.setEnabled(true);
        createExampleMenu.setEnabled(true);

        stepButton.setEnabled(false);
        prevButton.setEnabled(false);
        showResultButton.setEnabled(false);
        restartButton.setEnabled(false);
        createRunItem.setEnabled(false);
    }

    public GraphicsPanel getGraphicsPanel() {
        return graphicsPanel;
    }

    public void redraw() {
        graphicsPanel.redrawGraph();
    }

    public void runAlgorithm() throws Exception{
algorithmManager.runAlgorithm();}

    public void makeInvisibleAlgoButtons() {
        weightOfMST.setVisible(false);
        stepButton.setVisible(false);
        prevButton.setVisible(false);
        showResultButton.setVisible(false);
        restartButton.setVisible(false);
    }

    public void makeDrawGraphRequest(Node[] nodes) {
        graphicsPanel.drawGraph(nodes);
    }

    public void log(String str) {
```

```
            alw.printCondition(str);
        }
    }
```

ПРИЛОЖЕНИЕ В

NODE.JAVA

```
package kolmykov_shishkin_stepanov;

import kolmykov_shishkin_stepanov.algorithm.Edge;
import kolmykov_shishkin_stepanov.exceptions.AddEdgeException;

import java.util.HashMap;
import java.util.Map;

public class Node {
    private int number;

    private Map<Integer, Edge> edgesMap = new HashMap<>();
//Словарь номер вершины - длина ребра для хранения исходящих ребер

    public Node(int number){ this.number = number; }

    public int getNumber() {
        return number;
    }

    public void addEdge(int number, Edge edge) throws
AddEdgeException {
        edgesMap.put(number, edge);
    }

    public Map<Integer, Edge> getEdgesMap() {
        return edgesMap;
    }
}
```

ПРИЛОЖЕНИЕ Г

ALGORITHMLOGGINGWINDOW.JAVA

```
package kolmykov_shishkin_stepanov;

import kolmykov_shishkin_stepanov.algorithm.Edge;

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

public class AlgorithmLoggingWindow extends JFrame {
    private Window window;
    private JTextArea textArea;

    public AlgorithmLoggingWindow(Window window) {
        this.window = window;

        setSize(300, 120);

        textArea = new JTextArea(8, 20);
        textArea.setText("");

textArea.setCaretPosition(textArea.getDocument().getLength());
        //textArea.setCaretPosition(0);
        //textArea.setAutoscrolls(true);
        textArea.setLineWrap(true);
        textArea.setEditable(false);

        JScrollPane scrollPane = new JScrollPane(textArea);

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLB
AR_ALWAYS);
        add(scrollPane);
    }

    public void printCondition(String log){
        textArea.append(log + "\n");
    }
}
```

ПРИЛОЖЕНИЕ Д

ADDEDGEWINDOW.JAVA

```
package kolmykov_shishkin_stepanov;

import kolmykov_shishkin_stepanov.exceptions.AddEdgeException;

import javax.swing.*;
import javax.swing.text.NumberFormatter;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.ParseException;

public class AddEdgeWindow extends JFrame {
    private Window window;

    private JLabel v1Label;
    private JLabel v2Label;
    private JLabel capacityLabel;
    private JFormattedTextField capacityInput;
    private JFormattedTextField v1Input;
    private JFormattedTextField v2Input;
    private JButton addButton;

    public AddEdgeWindow(Window window) {
        this.window = window;
        setSize(350, 200);
        setLocationRelativeTo(null);
        setLayout(new GridBagLayout());
        setTitle("Adding edge");

        v1Label = new JLabel("Vertex 1 number");
        v2Label = new JLabel("Vertex 2 number");
        capacityLabel = new JLabel("Capacity       ");
        capacityInput      =      new      JFormattedTextField(new
NumberFormatter());
        v1Input = new JFormattedTextField(new NumberFormatter());
        v2Input = new JFormattedTextField(new NumberFormatter());
        addButton = new JButton("Add");

        add(v1Label, new GridBagConstraints(0, 0, 2, 1, 0, 0.5,
                GridBagConstraints.SOUTH,
GridBagConstraints.HORIZONTAL,
                new Insets(1, 1, 1, 1), 20, 0));
```

```
        add(v2Label, new GridBagConstraints(2, 0, 2, 1, 0, 0.5,
                GridBagConstraints.SOUTH,
GridBagConstraints.HORIZONTAL,
                new Insets(1, 1, 1, 1), 20, 0));
        add(capacityLabel, new GridBagConstraints(4, 0, 2, 1, 0,
0.5,
                GridBagConstraints.SOUTH,
GridBagConstraints.HORIZONTAL,
                new Insets(1, 1, 1, 1), 20, 0));
        add(v1Input, new GridBagConstraints(0, 1, 2, 1, 0, 0.5,
                GridBagConstraints.NORTH,
GridBagConstraints.HORIZONTAL,
                new Insets(1, 1, 1, 1), 20, 0));
        add(v2Input, new GridBagConstraints(2, 1, 2, 1, 0, 0.5,
                GridBagConstraints.NORTH,
GridBagConstraints.HORIZONTAL,
                new Insets(1, 1, 1, 1), 20, 0));
        add(capacityInput, new GridBagConstraints(4, 1, 2, 1, 0,
0.5,
                GridBagConstraints.NORTH,
GridBagConstraints.HORIZONTAL,
                new Insets(1, 1, 1, 1), 20, 0));
        add(addButton, new GridBagConstraints(0, 2, 9, 1, 0, 0.5,
                GridBagConstraints.NORTH,
GridBagConstraints.HORIZONTAL,
                new Insets(1, 1, 1, 1), 20, 0));

        addButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int v1num = 0;
                int v2num = 0;
                int capacity = 0;
                try {
                    v1num = Integer.parseInt(v1Input.getText());
                    v2num = Integer.parseInt(v2Input.getText());
                    capacity                                    =
Integer.parseInt(capacityInput.getText());

                    window.addEdge(v1num, v2num, capacity);
                }catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(window,
"Incorrect input (NaN)");
                }
            }
```

```
            });
        }
    }
```

ПРИЛОЖЕНИЕ Е

ALGORITHM.JAVA

```
package kolmykov_shishkin_stepanov.algorithm;

import kolmykov_shishkin_stepanov.exceptions.AddEdgeException;

public abstract class Algorithm {
    protected AlgorithmManager algorithmManager;

    public Algorithm(AlgorithmManager algorithmManager) {
        this.algorithmManager = algorithmManager;
    }

    public abstract void run() throws Exception;

    public abstract boolean step();

    public abstract void getResult();

    public abstract int getIntermediateResult();

    public abstract void restart();

    public abstract void prev() throws Exception;

    public abstract boolean isValid();

    public abstract void setNumOfNodes(int num);

    public abstract void addEdge(int from, int to, int capacity)
throws AddEdgeException;
}
```

ПРИЛОЖЕНИЕ Ж

ALGORITHMMANAGER.JAVA

```
package kolmykov_shishkin_stepanov.algorithm;

import kolmykov_shishkin_stepanov.Node;
import kolmykov_shishkin_stepanov.Window;
import kolmykov_shishkin_stepanov.exceptions.AddEdgeException;

public class AlgorithmManager {
    private Window window;
    private Algorithm algorithm;

    public AlgorithmManager(Window window, Algorithm algorithm) {
        this.window = window;
        this.algorithm = algorithm;
    }

    public AlgorithmManager(Algorithm algorithm) {
        this.algorithm = algorithm;
    }

    public AlgorithmManager(Window window) {
        this.window = window;
    }

    public AlgorithmManager() {
    }

    public void setWindow(Window window) {
        this.window = window;
    }

    public void setAlgorithm(Algorithm algorithm) {
        this.algorithm = algorithm;
    }

    //-----------------------------------------------------------

    public void runAlgorithm() throws Exception {
        algorithm.run();
    }

    public boolean makeStep() {
        return algorithm.step();
```

```
    }

    public void makePrev() throws Exception{
        algorithm.prev();
    }

    public void getResultOfAlgorithm() {
        algorithm.getResult();
    }

    public int getIntermediateResultOfAlgorithm() {
        return algorithm.getIntermediateResult();
    }

    public void restartAlgorithm() {
        algorithm.restart();
    }

    public void setNumOfNodes(int num) {
        algorithm.setNumOfNodes(num);
    }

    public void addEdge(int from, int to, int capacity) throws
AddEdgeException {
        algorithm.addEdge(from, to, capacity);
    }

    //-------------------------------------------------------------
-----------

    public void makeDrawGraphRequest(Node[] nodes) {
        window.makeDrawGraphRequest(nodes);
    }

    public void log(String str) {
        window.log(str);
    }
}
```

ПРИЛОЖЕНИЕ 3

DSF.JAVA

```
package kolmykov_shishkin_stepanov.algorithm;

import java.util.ArrayList;

public class DSF { // класс системы непересекающихся множеств
    int[] set; // номер множества
    int[] rnk; // ранг
    int size = 0;

    ArrayList<ArrayList<Integer>>        prevSet        =        new
ArrayList<ArrayList<Integer>>();
    ArrayList<ArrayList<Integer>>        prevRnk        =        new
ArrayList<ArrayList<Integer>>();

    DSF(int size) {
        this.size = size;
        set = new int[size];
        rnk = new int[size];
        for (int i = 0; i < size; i++) {
            set[i] = i;
        }
    }

    // Возвращает множество, которому принадлежит x
    int set(int x) {
        return x == set[x] ? x : (set[x] = set(set[x]));
    }

    // Если u и v лежат в разных множествах, то сливаем их и
возвращаем true
    boolean union(int u, int v) {   //функция слияния
        if ((u = set(u)) == (v = set(v))) {
            ArrayList <Integer> tmpSet = new ArrayList<>();
            ArrayList <Integer> tmpRnk = new ArrayList<>();
            for (int i = 0; i < size; i++) {
                tmpSet.add(set[i]);
                tmpRnk.add(rnk[i]);
            }
            prevSet.add(tmpSet);
            prevRnk.add(tmpRnk);
            return false;
        }
```

```
        ArrayList <Integer> tmpSet = new ArrayList<>();
        ArrayList <Integer> tmpRnk = new ArrayList<>();
        for (int i = 0; i < size; i++) {
            tmpSet.add(set[i]);
            tmpRnk.add(rnk[i]);
        }
        prevSet.add(tmpSet);
        prevRnk.add(tmpRnk);

        if (rnk[u] < rnk[v]) {
            set[u] = v;
        } else {
            set[v] = u;
            if (rnk[u] == rnk[v])
                rnk[u]++;
        }
        return true;
    }

    void unUnionForPrev() {          //функция, которая отменяет
предыдущее слияние для отката назад

        ArrayList <Integer> tmpSet = new ArrayList<>();
        ArrayList <Integer> tmpRnk = new ArrayList<>();
        tmpSet = prevSet.get(prevSet.size() - 1);
        tmpRnk = prevRnk.get(prevRnk.size() - 1);

        for (int i = 0; i < size; i++) {
            set[i] = tmpSet.get(i);
            rnk[i] = tmpRnk.get(i);
        }

        prevSet.remove(prevSet.size() - 1);
        prevRnk.remove(prevRnk.size() - 1);
    }
}
```

ПРИЛОЖЕНИЕ И

EDGE.JAVA

```
package kolmykov_shishkin_stepanov.algorithm;

import java.awt.*;

public class Edge implements Comparable<Edge> {
    private int from;
    private int to;
    private int capacity;
    Color color = Color.BLACK;


    public Edge(int from, int to, int capacity){
        this.from = from;
        this.to = to;
        this.capacity = capacity;
    }

    @Override
    public int compareTo(Edge o) {
        if (capacity != o.capacity) {
            return capacity < o.capacity ? -1 : 1;
        }
        return 0;
    }

    public int getCapacity () {return capacity;}
    public int getFrom () {return from;}
    public int getTo () {return  to;}

    public Color getColor() {
        return color;
    }

    public void setColor(Color color) {
        this.color = color;
    }
}
```

ПРИЛОЖЕНИЕ К

KRUSKALALGORITHM.JAVA

```
package kolmykov_shishkin_stepanov.algorithm;

import kolmykov_shishkin_stepanov.Node;
import kolmykov_shishkin_stepanov.exceptions.AddEdgeException;
import
kolmykov_shishkin_stepanov.exceptions.IncoherentGraphException;
import
kolmykov_shishkin_stepanov.exceptions.PrevBeforeStepException;

import java.awt.*;
import java.util.ArrayList;

public class KruskalAlgorithm extends Algorithm {
    private int nodesNumber = 0;
    private Node[] nodes = new Node[0];
    private ArrayList<Edge> edges = new ArrayList<>();
    private DSF dsf;
    private Edge currentEdge;
    private int currentEdgeIndex;
    private int minMSTWeight = 0;

    public KruskalAlgorithm(AlgorithmManager algorithmManager) {
        super(algorithmManager);
    }

    public void setNumOfNodes(int num) {
        this.nodesNumber = num;
        nodes = new Node[num];
        for (int i = 0; i < num; i++) {
            nodes[i] = new Node(i);
        }
        edges = new ArrayList<>();
        makeDrawRequest();
    }

    public void addEdge(int number1, int number2, int capacity)
throws AddEdgeException {
        if (number1 >= nodes.length || number1 < 0) {
            throw new AddEdgeException("There is no such vertex ("
+ number1 + ")");
        } else if (number2 >= nodes.length || number2 < 0) {
```

```
            throw new AddEdgeException("There is no such vertex ("
+ number2 + ")");
        }
        else if (number1 == number2) {
            throw new AddEdgeException("Same vertex");
        } else if (capacity <= 0) {
            throw new AddEdgeException("Capacity <= 0");
        }

        Edge edge = new Edge(number1, number2, capacity);
        nodes[number1].addEdge(number2, edge);
        nodes[number2].addEdge(number1, edge);
        edges.add(edge);
        makeDrawRequest();
    }

    private void makeDrawRequest() {
        algorithmManager.makeDrawGraphRequest(nodes);
    }

    public void run() throws IncoherentGraphException{
        if (!isValid()) {
            throw    new    IncoherentGraphException("Graph    is
inherit");
        }
        currentEdge = null;
        currentEdgeIndex = 0;
        minMSTWeight = 0;

        dsf = new DSF(nodesNumber); // CHM
        edges.sort(Edge::compareTo); // Сортируем ребра
    }

    public boolean step() {

        if (currentEdge == null) {
            chooseEdge();
        } else {
            handleEdge();
            currentEdge = null;
        }

        if (currentEdgeIndex == edges.size()) {
            return false;
        }
```

```
        return true;
    }

    private void chooseEdge() {
        currentEdge = edges.get(currentEdgeIndex);
        currentEdge.setColor(Color.YELLOW);
        algorithmManager.log("Select edge " +
currentEdge.getFrom() + " - " + currentEdge.getTo() + " to
check");
    }

    private void handleEdge() {
        if (dsf.union(currentEdge.getFrom(), currentEdge.getTo()))
{// если ребра принадлежат разным компонентам
            currentEdge.setColor(Color.GREEN);
            minMSTWeight += currentEdge.getCapacity(); //
добавляем вес ребра к стоимости MST
            algorithmManager.log("Edge " + currentEdge.getFrom() +
" - " + currentEdge.getTo() +
                    " doesn't form cycles and is added to the
spanning tree");
        } else {
            currentEdge.setColor(Color.RED);
            algorithmManager.log("Edge " + currentEdge.getFrom() +
" - " + currentEdge.getTo() + " form cycles");
        }
        currentEdgeIndex++;
    }

    public int getIntermediateResult() {
        return minMSTWeight;
    }

    public void getResult() {
        while (step()) {
//            window.redraw();
//            try {
//                Thread.sleep(500);
//            } catch (InterruptedException e) {
//                e.printStackTrace();
//            }
        }
    }
```

```
    public void restart() {
        currentEdge = null;
        currentEdgeIndex = 0;
        minMSTWeight = 0;
        dsf = new DSF(nodesNumber); // CHM

        for (Edge edge : edges) {
            edge.setColor(Color.BLACK);
        }
    }

    public boolean isValid() { //проверка связности
        ArrayList<Integer> visited = new ArrayList<>();
        visited.add(0);
        for (int i = 0; i < visited.size(); i++) {
            for          (Integer          neighborNumber          :
nodes[visited.get(i)].getEdgesMap().keySet()) {
                if (!visited.contains(neighborNumber)) {
                    visited.add(neighborNumber);
                }
            }
        }
        return (visited.size() == nodes.length);
    }

    public void prev() throws PrevBeforeStepException{
        boolean isPrevAtIndex0 = false;
        if (currentEdgeIndex == 0) {
            currentEdge = edges.get(currentEdgeIndex);
            if (currentEdge.getColor() == Color.YELLOW) {
                isPrevAtIndex0 = true;
                currentEdge.setColor(Color.BLACK);
            }
            currentEdge = null;
        }

        if (currentEdgeIndex <= 0) {
            if (!isPrevAtIndex0)
                throw new PrevBeforeStepException("You can't click
this button because you haven't pressed the button \"step\" yet");
            return;
        }

        if (currentEdge == null) {
            rehandleEdgeForPrev();
```

```
        }
        else if (currentEdge.getColor() == Color.YELLOW) {
            rechooseEdgeForPrev();
            currentEdge = null;
        }
    }

    private void rechooseEdgeForPrev() {     //Если перед нажатием
"prev" ребро закрашено в желтый
        currentEdge = edges.get(currentEdgeIndex);
        currentEdge.setColor(Color.BLACK);
        algorithmManager.log("Edge " + currentEdge.getFrom() + " -
" + currentEdge.getTo() + " is painted black");
    }

    private void rehandleEdgeForPrev() {     //Если перед нажатием
"prev" ребро закрашено в зеленый или красный
        currentEdgeIndex--;
        currentEdge = edges.get(currentEdgeIndex);
        if (currentEdge.getColor() == Color.GREEN) {
            minMSTWeight -= currentEdge.getCapacity();
        }
        currentEdge.setColor(Color.YELLOW);
        dsf.unUnionForPrev();
        algorithmManager.log("Edge " + currentEdge.getFrom() + " -
" + currentEdge.getTo() + " is painted yellow");
    }
}
```

ПРИЛОЖЕНИЕ Л

ADDEDGEEXCEPTION.JAVA

```
package kolmykov_shishkin_stepanov.exceptions;

public class AddEdgeException extends Exception {
    public AddEdgeException() {}

    public AddEdgeException(String message) {
        super(message);
    }

    public AddEdgeException(String message, Throwable cause) {
        super(message, cause);
    }

    public AddEdgeException(Throwable cause) {
        super(cause);
    }

    public AddEdgeException(String message, Throwable cause,
boolean enableSuppression, boolean writableStackTrace) {
        super(message,          cause,          enableSuppression,
writableStackTrace);
    }

}
```

ПРИЛОЖЕНИЕ М

INCOHERENTGRAPHEXCEPTION.JAVA

```
package kolmykov_shishkin_stepanov.exceptions;

public class IncoherentGraphException extends Exception {
    public IncoherentGraphException(String message) {
        super(message);
    }
}
```

ПРИЛОЖЕНИЕ Н

PREVBEFORESTEPEXCEPTION.JAVA

```
package kolmykov_shishkin_stepanov.exceptions;

public class PrevBeforeStepException extends Exception {
    public PrevBeforeStepException(String message) {
        super(message);
    }
}
```

ПРИЛОЖЕНИЕ О

CIRCLE.JAVA

```
package kolmykov_shishkin_stepanov.graphics;

import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;

public class Circle extends Ellipse2D {
    double x;
    double y;
    double w;
    double h;

    public Circle(double x, double y, double w, double h) {
        setFrame(x, y, w, h);
    }

    @Override
    public double getX() {
        return x;
    }

    @Override
    public double getY() {
        return y;
    }

    @Override
    public double getWidth() {
        return w;
    }

    @Override
    public double getHeight() {
        return h;
    }

    @Override
    public boolean isEmpty() {
        return false;
    }

    @Override
    public void setFrame(double x, double y, double w, double h) {
```

```
        this.x = x;
        this.y = y;
        this.w = w;
        this.h = h;
    }

    @Override
    public Rectangle2D getBounds2D() {
        return null;
    }
}
```

ПРИЛОЖЕНИЕ П

GRAPHICSPANEL.JAVA

```
package kolmykov_shishkin_stepanov.graphics;

import kolmykov_shishkin_stepanov.Node;
import kolmykov_shishkin_stepanov.algorithm.Edge;

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

public class GraphicsPanel extends JPanel {
    private final double RADIUS = 300; // было 450, на 13-дюймовый
экран не все вершины влезают
    private final double H = 50;
    private final double W = 50;
    private final double X0 = 800;
    private final double Y0 = 450;
    private final int RECT_H = 15;
    private final int RECT_W = 15;

    private Node[] lastNodes;

    public void redrawGraph() {
        if(lastNodes != null) {
            drawGraph(lastNodes);
        }
    }

    public void drawGraph(Node[] nodes) { //Для каждой вершины
отрисовывается вершина и ребра к ее соседям
        this.lastNodes = nodes;
        clear();
        for (Node node : nodes) {
            drawNode(node.getNumber(), nodes.length);
            Map<Integer, Edge> edgesMap = node.getEdgesMap();
            for (Integer neighbourNumber : edgesMap.keySet()) {
                drawEdge(node.getNumber(),         neighbourNumber,
edgesMap.get(neighbourNumber).getCapacity(),          nodes.length,
edgesMap.get(neighbourNumber).getColor());
            }
        }
    }
```

```
    public void drawNode(int number, int quantity) {
        double angle = Math.PI* 2/quantity * number;
        double x = X0 + RADIUS * Math.cos(angle);
        double y = Y0 + RADIUS * Math.sin(angle);
        drawCircle(x, y, W, H);
        drawNumber(number, x, y, Color.WHITE);
    }

    public void drawEdge(int number1, int number2, int value, int
quantity, Color color) {
        double angle1 = Math.PI* 2/quantity * number1;
        double x1 = X0 + RADIUS * Math.cos(angle1) + 20;
        double y1 = Y0 + RADIUS * Math.sin(angle1) + 30;

        double angle2 = Math.PI* 2/quantity * number2;
        double x2 = X0 + RADIUS * Math.cos(angle2) + 20;
        double y2 = Y0 + RADIUS * Math.sin(angle2) + 30;

        drawLine(x1, y1, x2, y2, color);
        drawSquare((int)((x1 + x2)/2), (int)((y1 + y2)/2 - 10),
RECT_W, RECT_H);
        drawNumber(value, (x1 + x2 - 40)/2, (y1 + y2 - 60)/2,
Color.BLACK);
        drawNode(number1, quantity);
        drawNode(number2, quantity);
    }

    private void drawLine(double x1, double y1, double x2, double
y2, Color color) {
        Graphics2D graphics2D = (Graphics2D) getGraphics();
        graphics2D.setColor(color);
        graphics2D.setBackground(color);
        graphics2D.setStroke(new BasicStroke(4));
        graphics2D.draw(new Line(x1, y1, x2, y2));
        graphics2D.dispose();
    }

    private void drawCircle(double x, double y, double w, double
h) {
        Graphics2D graphics2D = (Graphics2D) getGraphics();
        graphics2D.setColor(new Color(158, 8, 8));
        graphics2D.fill(new Circle(x, y, w, h));
        graphics2D.dispose();
    }
```

```
    private void drawNumber(int number, double x, double y, Color
color) {
        Graphics2D graphics2D = (Graphics2D) getGraphics();
        graphics2D.setColor(color);
        graphics2D.drawString(String.valueOf(number), (int)x + 20,
(int)y + 30);
    }

    private void drawSquare(int x, int y, int w, int h) {
        Graphics2D graphics2D = (Graphics2D) getGraphics();
        graphics2D.setColor(Color.WHITE);
        graphics2D.fillRect(x, y, w, h);
    }

    private void clear() {
        Graphics2D graphics2D = (Graphics2D) getGraphics();
        graphics2D.setBackground(Color.WHITE);
        graphics2D.setColor(Color.WHITE);
        graphics2D.fillRect(300, 0, 1000, 1000);
    }
}
```

ПРИЛОЖЕНИЕ Р

LINE.JAVA

```
package kolmykov_shishkin_stepanov.graphics;

import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;

public class Line extends Line2D {
    private Point p1;
    private Point p2;

    public Line(double x1, double y1, double x2, double y2) {
        this.p1 = new Point(x1, y1);
        this.p2 = new Point(x2, y2);
    }

    @Override
    public double getX1() {
        return p1.getX();
    }

    @Override
    public double getY1() {
        return p1.getY();
    }

    @Override
    public Point2D getP1() {
        return p1;
    }

    @Override
    public double getX2() {
        return p2.getX();
    }

    @Override
    public double getY2() {
        return p2.getY();
    }

    @Override
    public Point2D getP2() {
```

```
        return p2;
    }

    @Override
    public void setLine(double x1, double y1, double x2, double
y2) {
        this.p1.setLocation(x1, y1);
        this.p2.setLocation(x2, y2);
    }

    @Override
    public Rectangle2D getBounds2D() {
        return null;
    }
}
```

ПРИЛОЖЕНИЕ С

POINT.JAVA

```
package kolmykov_shishkin_stepanov.graphics;

import java.awt.geom.Point2D;

public class Point extends Point2D {
    private double x;
    private double y;

    public Point(double x, double y) {
        setLocation(x, y);
    }

    @Override
    public double getX() {
        return x;
    }

    @Override
    public double getY() {
        return y;
    }

    @Override
    public void setLocation(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

ПРИЛОЖЕНИЕ Т

CREATEEDGEACTIONLISTENER.JAVA

```
package kolmykov_shishkin_stepanov.listeners;

import kolmykov_shishkin_stepanov.AddEdgeWindow;
import kolmykov_shishkin_stepanov.Window;

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

public class CreateEdgeActionListener implements ActionListener {
    Window window;

    public CreateEdgeActionListener(Window window) {
        this.window = window;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        new AddEdgeWindow(window).setVisible(true);
    }
}
```

ПРИЛОЖЕНИЕ У

CREATEFIRSTEXAMPLEACTIONLISTENER.JAVA

```
package kolmykov_shishkin_stepanov.listeners;

import kolmykov_shishkin_stepanov.Window;

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

public class CreateFirstExampleActionListener implements
ActionListener {
    private Window window;

    public CreateFirstExampleActionListener(Window window) {
        this.window = window;
    }

    @Override
    public void actionPerformed(ActionEvent e) {

        window.setNumberOfNodes(7);
        //window.changeEnableOfCreateMenu();
        window.changeEnableOfExample();

        window.addEdge(0, 1, 7);
        window.addEdge(0, 3, 5);
        window.addEdge(1, 2, 8);
        window.addEdge(1, 3, 9);
        window.addEdge(1, 4, 7);
        window.addEdge(2, 4, 5);
        window.addEdge(3, 5, 6);
        window.addEdge(3, 4, 15);
        window.addEdge(4, 5, 8);
        window.addEdge(4, 6, 9);
        window.addEdge(5, 6, 11);

        window.log("First example loaded");
    }
}
```

ПРИЛОЖЕНИЕ Ф

CREATENODEACTIONLISTENER.JAVA

```
package kolmykov_shishkin_stepanov.listeners;

import kolmykov_shishkin_stepanov.Window;

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

public class CreateNodeActionListener implements ActionListener {
    Window window;

    public CreateNodeActionListener(Window window) {
        this.window = window;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String input = JOptionPane.showInputDialog("Enter the
number of vertices (2 <= number <= 15)");
        int num;
        if(input == null) {
            return;
        }
        try {
            num = Integer.parseInt(input);
        }catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(window, "Incorrect
input");
            window.log("Error: Incorrect input");
            return;
        }
        if (num < 2) {
            JOptionPane.showMessageDialog(window, "Incorrect input
(< 2)");
            window.log("Error: Incorrect input (< 2)");
            return;
        } else if (num > 15) {
            JOptionPane.showMessageDialog(window, "Incorrect input
(> 15)");
            window.log("Error: Incorrect input (> 15)");
            return;
        }
```

```
            window.setNumberOfNodes(num);

            window.changeEnableOfCreateMenu();

            window.log("Created a new graph with " + num + "
vertices");
        }
}
```

ПРИЛОЖЕНИЕ X

CREATESECONDEXAMPLEACTIONLISTENER.JAVA

```
package kolmykov_shishkin_stepanov.listeners;

import kolmykov_shishkin_stepanov.Window;

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

public class CreateSecondExampleActionListener implements
ActionListener {

    private Window window;

    public CreateSecondExampleActionListener(Window window) {
        this.window = window;
    }

    @Override
    public void actionPerformed(ActionEvent e) {

        window.setNumberOfNodes(5);
        //window.changeEnableOfCreateMenu();
        window.changeEnableOfExample();

        window.addEdge(0, 1, 3);
        window.addEdge(0, 4, 1);
        window.addEdge(1, 4, 4);
        window.addEdge(1, 2, 5);
        window.addEdge(2, 4, 6);
        window.addEdge(2, 3, 2);
        window.addEdge(3, 4, 7);

        window.log("Second example loaded");
    }
}
```

ПРИЛОЖЕНИЕ Ц

CREATETHIRDEXAMPLEACTIONLISTENER.JAVA

```
package kolmykov_shishkin_stepanov.listeners;

import kolmykov_shishkin_stepanov.Window;

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

public class CreateThirdExampleActionListener implements
ActionListener {

    private Window window;

    public CreateThirdExampleActionListener(Window window) {
        this.window = window;
    }

    @Override
    public void actionPerformed(ActionEvent e) {

        window.setNumberOfNodes(9);
        //window.changeEnableOfCreateMenu();
        window.changeEnableOfExample();

        window.addEdge(0, 1, 4);
        window.addEdge(0, 7, 8);
        window.addEdge(1, 7, 11);
        window.addEdge(1, 2, 8);
        window.addEdge(7, 8, 7);
        window.addEdge(8, 2, 2);
        window.addEdge(8, 6, 6);
        window.addEdge(7, 6, 1);
        window.addEdge(6, 5, 2);
        window.addEdge(2, 5, 4);
        window.addEdge(2, 3, 7);
        window.addEdge(3, 5, 14);
        window.addEdge(5, 4, 10);
        window.addEdge(3, 4, 9);

        window.log("Third example loaded");
    }
}
```

ПРИЛОЖЕНИЕ Ч

RESTARTACTIONLISTENER.JAVA

```
package kolmykov_shishkin_stepanov.listeners;

import kolmykov_shishkin_stepanov.Window;

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

public class RestartActionListener implements ActionListener {
    private Window window;

    public RestartActionListener(Window window) {
        this.window = window;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        //window.changeEnableOfResultButton();
        window.redraw();
    }


}
```

ПРИЛОЖЕНИЕ Ш

RESULTACTIONLISTENER.JAVA

```
package kolmykov_shishkin_stepanov.listeners;

import kolmykov_shishkin_stepanov.Window;

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

public class ResultActionListener implements ActionListener {
    private Window window;

    public ResultActionListener(Window window) {
        this.window = window;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        window.changeEnableOfResultButton();
        window.redraw();

    }
}
```

ПРИЛОЖЕНИЕ Щ

RUNACTIONLISTENER.JAVA

```
package kolmykov_shishkin_stepanov.listeners;

import kolmykov_shishkin_stepanov.AddEdgeWindow;
import kolmykov_shishkin_stepanov.AlgorithmLoggingWindow;
import kolmykov_shishkin_stepanov.Window;

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

public class RunActionListener implements ActionListener {
    private Window window;

    public RunActionListener(Window window) {
        this.window = window;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            window.runAlgorithm();
            window.changeEnableOfRunAlgButton();
            window.redraw();
            window.log("Algorithm starts");
        }
        catch (Exception ex){
            JOptionPane.showMessageDialog(window,
ex.getMessage());
            window.log("Error: " + ex.getMessage());
        }
    }
}
```

ПРИЛОЖЕНИЕ Ы

EDGECOLOR.JAVA

```
package kolmykov_shishkin_stepanov.algorithm;

public enum EdgeColor {
    RED,YELLOW, GREEN, DEFAULT
}
```