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

ОТЧЕТ

по учебной практике

Тема: Визуализация алгоритма Краскала

Студентка гр. 8381 _____ Гречко В.Д.

Студент гр. 8304 _____ Ястребов И.М.

Студент гр. 8304 _____ Кирьянов Д.И.

Руководитель _____ Жангиров Т.Р.

Санкт-Петербург
2020

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студентка Гречко В.Д. группы 8381

Студент Кирьянов Д.И. группы 8304

Студент Ястребов И.М. группы 8304

Тема практики: алгоритм Краскала

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

Студентка	_____	Гречко В.Д.
Студент	_____	Ястребов И.М.
Студент	_____	Кирьянов Д.И.
Руководитель	_____	Жангиров Т.Р.

АННОТАЦИЯ

Цель работы — визуализация работы предложенного алгоритма на языке программирования Java. В ходе прохождения практики предполагается изучение Java и разработка графического приложения.

Практика включает в себя два задания — вводное, для знакомства с языком, и основное, выполняемое в бригаде.

SUMMARY

The purpose of this work is to visualize the operation of the proposed algorithm in the Java programming language. During the internship, you will learn Java and develop a graphical application.

The practice includes two tasks — an introductory one for getting to know the language, and the main one, which is performed in the team.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача о нахождении минимального остовного дерева встречается часто: допустим, есть n городов, которые необходимо соединить дорогами, так, чтобы можно было добраться из любого города в любой другой (напрямую или через другие города). Разрешается строить дороги между заданными парами городов и известна стоимость строительства каждой такой дороги. Требуется решить, какие именно дороги нужно строить, чтобы минимизировать общую стоимость строительства.

Эта задача может быть сформулирована в терминах теории графов как задача о нахождении минимального остовного дерева в графе, вершины которого представляют города, рёбра — это пары городов, между которыми можно проложить прямую дорогу, а вес ребра равен стоимости строительства соответствующей дороги.

Алгоритм Краскала — эффективный алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Также алгоритм используется для нахождения некоторых приближений для задачи Штейнера.

Алгоритм описан Джозефом Краскалом (англ.) в 1956 году, этот алгоритм почти не отличается от алгоритма Борувки, предложенного Отакаром Борувкой в 1926 году.

Суть алгоритма: в начале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1. Требования к вводу исходных данных

Для входных данных определяются следующие требования: возможность рисования графа на холсте пользователем и возможность построения графа на основе заданного файла (уточнить в какой форме принимается граф).

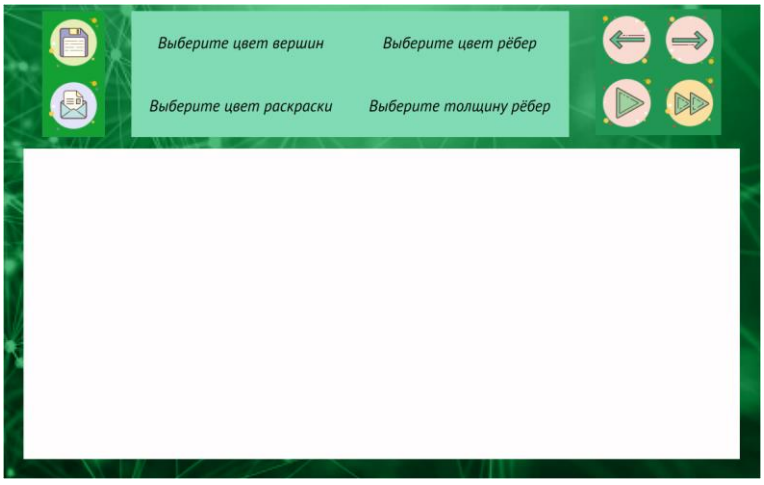
1.1.2. Требования к визуализации

Возможность рисования графа внутри программы (выделение отдельной области внутри приложения): кастомизация графа (толщина и цвет рёбер, цвет вершин, цвет обводки), свободное размещение вершин внутри области рисования.

Предполагается реализация следующих кнопок: выбор файла для считывания, панель кастомизации, пошаговое выполнение графа, возможность выбора раскраски множеств во время выполнения алгоритма, сохранение полученного графа в файл.

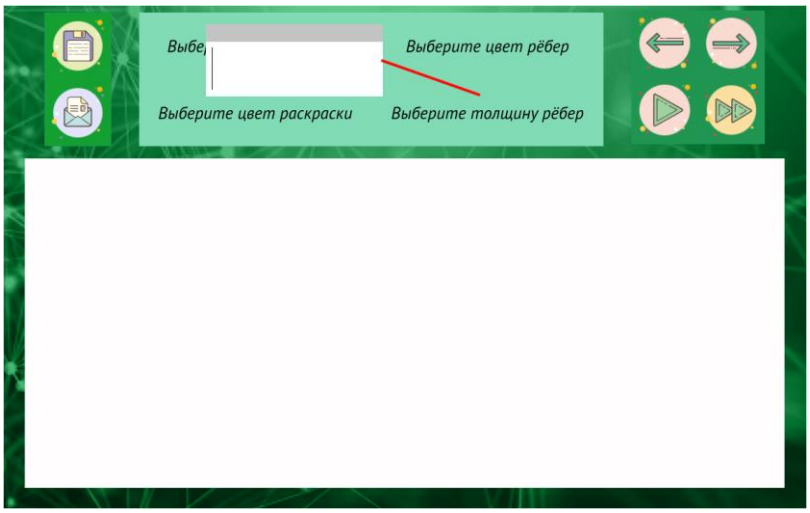
Предполагаемый стиль программы — ведущий цвет зелёный, с использованием белого, черного и серого.

Предполагаемый эскиз программы:



Интерфейс представляет собой три блока кнопок и холст для рисования. Добавление вершин будет осуществляется при нажатии правой кнопки мыши.

При нажатии на кнопку «Выберите …» будет открываться отдельно окно с пользовательским вводом/ список возможных вариантов.



1.1.3. Требования к возможностям программы

Программа предполагает реализацию следующего функционала: два способа считывания данных, панель настроек графа, сохранение графа в файл, редактирование графа, панель настроек алгоритма (настройка двух вариантов выполнения алгоритма, в случае пошагового режима — кнопки вперёд/назад).

1.1.4. Требования к выходным данных

Результат работы алгоритма является нарисованным графом внутри приложения и по желанию пользователя сохранение в текстовый файл (уточнить в какой форме принимается граф).

1.2. Уточнение требований после сдачи прототипа

После сдачи прототипа были внесены следующие уточнения: на момент сдачи программа представляла собой монолитное приложение, что необходимо было исправить.

1.3. Уточнение требований после сдачи первой версии программы

После сдачи первой версии программы необходимы были следующие изменения: разрыв зависимости GUI от текущего алгоритма, изменение структуры хранения вершин и самого графа (в частности хранение цвета), добавление абстрактной фабрики

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

На данный момент предполагается следующий план:

- 2 июля — обсуждение и согласование спецификации, распределение ролей в команде;
- 4-5 июля — разработка прототипа программы (пустой интерфейс).
- 6 июля — реализация алгоритма, без привязки к текущей программе.
- 7 июля — реализация 1 версии программы: автоматическое построение результата работы алгоритма на холсте, считывание графа с холста. Объединение алгоритма и интерфейса.
- 8 июля — показ первой итерации. После неё запуск тестов на работу алгоритма. Написание активной панели кастомизации графа.
- 9 июля — Реализация сохранения в файл и считывание из него. Отрисовка пошагового выполнения.
- 10 июля — сдача готовой программы.

2.2. Распределение ролей в бригаде

Предполагается следующее распределение (условное):

- Иван: алгоритмист;
- Вероника: тестировщик, документация, связь с преподавателем;
- Даниил: фронтенд.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

Были реализованы следующие пакеты:

Graph: структура для хранения графа. Содержит в себе следующие классы и методы:

Класс Edge	private Node first; private Node second; private Integer weight; представление ребра: вершины, содержащие ребро и его вес
	public Edge(Node first, Node second, Integer weight) public Edge() - конструкторы; public void setSecond(Node second) public void setFirst(Node first) public Node getSecond() public Node getFirst() - установка начального/конечного ребра, а также соответственно получение этих значений; public void setWeight(int weight) public int getWeight() - аналогично методы для установки и получения веса ребра static public class EdgeComparator implements Comparator<Edge> - компаратор, необходимый для сортировки рёбер при выполнении алгоритма
Класс Node	protected String name; - имя вершины
	public Node() public Node(String name) конструкторы класса

	public void setName(String name) public String getName - установка и получение имени вершины
Класс Graph	private ArrayList<Node> nodeList; private ArrayList<Edge> edgeList; -список ребер список вершин – хранение графа
	public Graph() public Graph(ArrayList<Edge> edgeList, ArrayList<Node> nodeList) - конструкторы public void setEdgeList(ArrayList<Edge> edgeList) public ArrayList<Edge> getEdgeList() public void setNodeList(ArrayList<Node> nodeList) public ArrayList<Node> getNodeList() - установка и получение доступа к полям public ArrayList<Node> adjacentNodes(Node e) - получение списка вершин, смежных к заданной public boolean isConnected() - проверка на связность графа public void save(String filename) - сохранение в файл текущего графа с холста public static Graph load(String filename) - загрузка графа из файла

NodeGUIdata	содержит отрисовочные и алогоритмические данные для вершин графа в приложении (чтобы не засорять класс вершина тем что не относится к вершине как таковой)
	protected int x = 0;
	protected int y = 0;
	- координаты на холсте у вершины

	protected Integer unionIndex = null; -индекс компоненты связности в которой находится вершина
	public void setX(int x) public void setY(int y) public int getX() public int getY() - установка и получение доступа к полям public void setVisited(boolean visited) public boolean isVisited() - соответственно установка и проверка на посещённость вершины public Integer getUnionIndex() public void setUnionIndex(Integer unionIndex) - установка и получение номера компоненты связности

Alghorithm: реализация алгоритма. Данный пакет хранит только один класс: public Graph KruskalAnalyze(Graph graph) – соответственно сам алгоритм Краскала.

GUI: пакет содержит следующие классы:

public abstract class EdgeFactoryInterface - интерфейс для фабрики-создателя обьектов Edge (ребра)	public abstract Edge getEdge(); public abstract Edge getEdge(Node first, Node second, Integer weight);
public abstract class GraphFactoryInterface - интерфейс для фабрики-создателя обьектов Graph (графа)	public abstract Graph getGraph(); public abstract Graph getGraph(ArrayList<Edge> edgeList, ArrayList<Node> nodeList);

	public abstract Graph getGraph(ArrayList<Edge> edgeList);

public abstract class NodeFactoryInterface
- интерфейс для фабрики-создателя обьектов Node (вершин)
	protected char name = 'a';
	public abstract Node getNode();
	public abstract Node getNode(String name);

public class NodeFactory extends NodeFactoryInterface
- фабрика вершин
	@Override
	public Node getNode(){
	return new Node(String.valueOf(name++));
	}
	@Override
	public Node getNode(String name){
	return new Node(name);
	}

public class GraphFactory extends GraphFactoryInterface
- фабрика графов
	@Override
	public Graph getGraph(){
	return new Graph();
	}
	
	@Override
	public Graph getGraph(ArrayList<Edge> edgeList, ArrayList<Node> nodeList){
	return new Graph(edgeList, nodeList);
	}
	
	@Override

	public Graph getGraph(ArrayList<Edge> edgeList) {
	return new Graph(edgeList);
	}

public class NodeFactory extends NodeFactoryInterface
- фабрика вершин
	@Override
	public Node getNode(){
	return new Node(String.valueOf(name++));
	}
	
	@Override
	public Node getNode(String name){
	return new Node(name);
	}

	private JButton saveButton = new JButton(new ImageIcon("src\\save.png"));
	private JButton loadButton = new JButton(new ImageIcon("src\\load.png"));
	
	private JButton colorNodes = new JButton("Цвет вершин");
	private JButton colorEdge = new JButton("Цвет ребер");
	private JButton colorFlood = new JButton("Стереть");

	private JButton thickness = new JButton("Справка");
	
	private JButton buttonStart = new JButton(new ImageIcon("src\\start.png"));
	private JButton buttonSkip = new JButton(new ImageIcon("src\\skip.png"));
	private JButton nextStep = new JButton(new ImageIcon("src\\next.png"));
	private JButton prevStep = new JButton(new ImageIcon("src\\prev.png"));
	
	// КНОПКИ
	
	GraphFactory factoryGraph = new GraphFactory();
	NodeFactory factoryNode = new NodeFactory();
	EdgeFactory factoryEdge = new EdgeFactory();
	
	//ФАБРИКИ КОТОРЫЕ БУДУТ ДАВАТЬ ОБЬЕКТЫ (ВМЕСТО ВЫЗОВОВ ОПЕРАТОРА NEW типа

	как graph g = new graph теперь будет graph g = factory.getgraph)
	
	boolean edgeAddFlag = false; // ФЛАГ НАЖАТИЯ НА СУЩЕСТВУЮЩУЮ ВЕРШИНУ С ПОПЫТКОЙ СОЗДАТЬ РЕБРО
	int index = 0; // ИНДЕКС ДЛЯ ПОШАГОВЫХ КНОПОК
	int collisionBreaker = 2; // ДОБАВКА К НАЗВАНИЯМ ВЕРШИНЫ ДЛЯ РАЗРЕШЕНИЯ КОЛЛИЗИЙ ИМЕН
	
	public GUI() { /ОГРОМНЫЙ КОНСТРУКТОР ИНТЕРФЕЙСА С ПАНЕЛЯМИ ОБРАБОТЧИКАМИ КНОПОК
	public class jPanel2 extends JPanel { //ХОЛСТ ДЛЯ СОЗДАНИЯ И РЕДАКТИРОВАНИЯ ГРАФОВ
	
	public static void execute() //ЗАПУСКАЕТ ГРАФИЧЕСКОЕ ПРИЛОЖЕНИЕ
	
	}

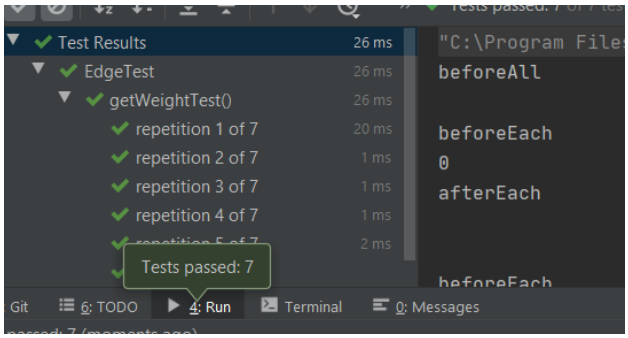
4. ТЕСТИРОВАНИЕ

4.1. Тестирование основных классов приложения

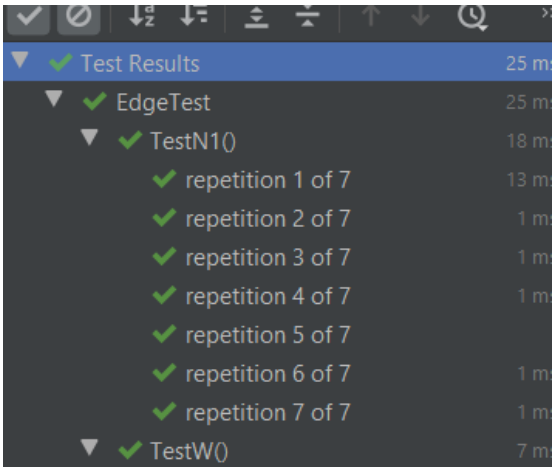
- **Тестирование класса Edge:**

Так как класс Node содержал несложные методы, его тестирование проводилось вручную ещё во время написания алгоритма без привязки к самому приложению.

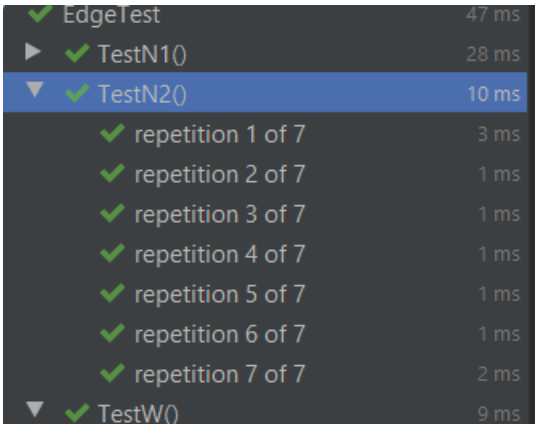
В классе Edge была реализована проверка правильного отображения веса, присваивания Вершин начало и конца. Результаты тестирования приведены ниже:



Прохождение тестов по определению веса



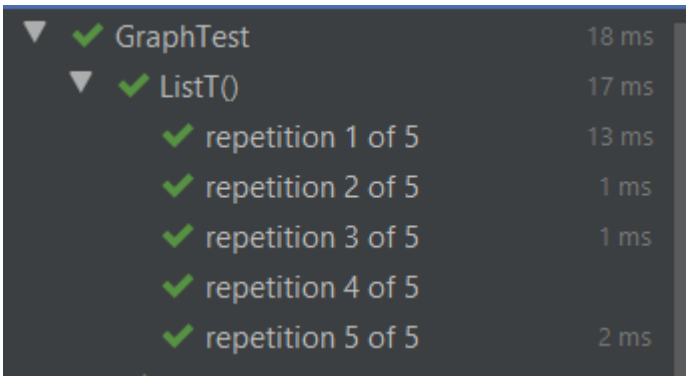
Прохождение тестов по считыванию начала и конца каждой вершины версия 1



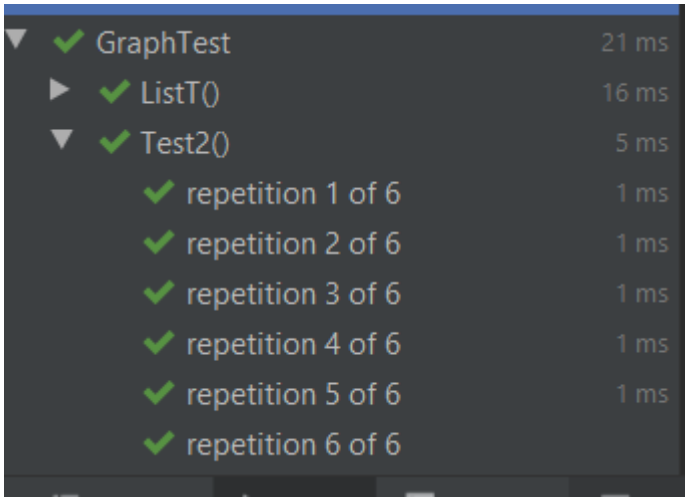
Прохождение тестов по считыванию начала и конца каждой вершины версия 2

- **Тестирование класса Graph:**

В классе Graph были реализованы проверки на следующее: правильное построение графа на основе списков, проверка на смежные вершины и проверка на связность графа.

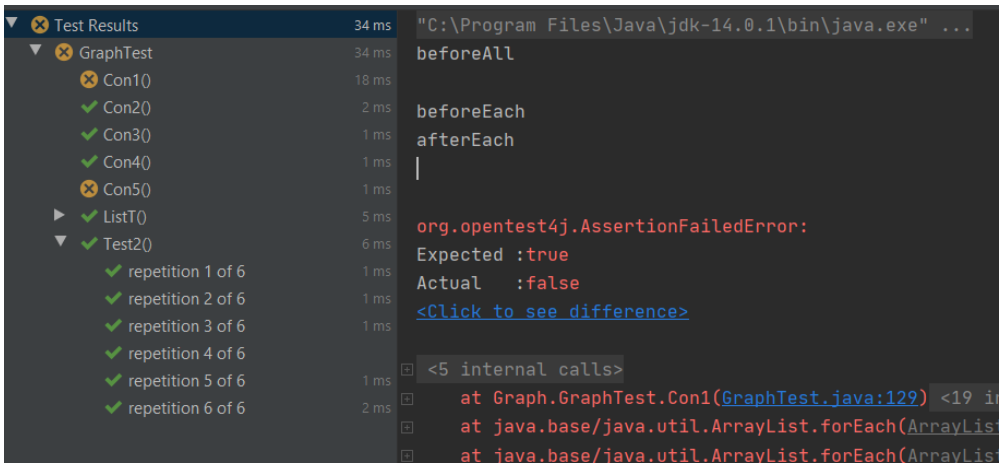


Тестирование на построение графа

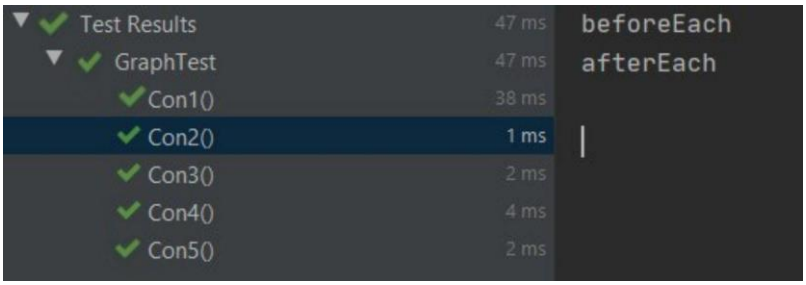


Тестирование на определение смежных вершин

Далее, было проведено тестирование на работу метода проверки связности графа. Однако в результате запуска были выявлены ошибки: при отправки корректного графа метод возвращал отрицательный результат.



Исправленный вариант:



4.2. Тестирование алгоритма

Тестирование алгоритма представляло собой интеграционное тестирование: выполнение алгоритма напрямую зависело от правильно реализованных методов в других классах, построения графа, методов в самом классе Алгоритма.

Для тестирования была написана база тестов с разной сложости графами. Тест был запущен на обработку каждого файла и вывода ответа на консоль в тестовом режиме. Ознакомится с содержимым тестов можно в директории examples. Результат работы:

```
1 test
a e
c d
a b
b c
2 test
a g
e g
c g
e d
a b
f g
3 test
1 3
4 6
2 5
3 6
2 3
```

```
4 test
A B
E D
C E
A E
5 test
4 5
0 1
1 4
0 3
8 9
0 2
6 9
5 6
8 7
6 test
k g
g l
a b
k i
c d
a k
b c
d e
```

```
7 test
d c
b c
f d
a c
8 test
g i
b e
d e
a b
i k
n k
g h
g f
e f
a c
```

Результаты запуска 8 тестов

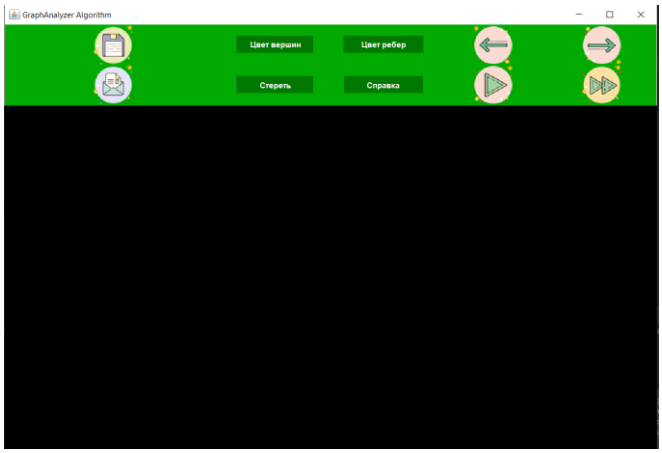
4.3 Тестирование GUI

Тестирование GUI проводилось вручную.

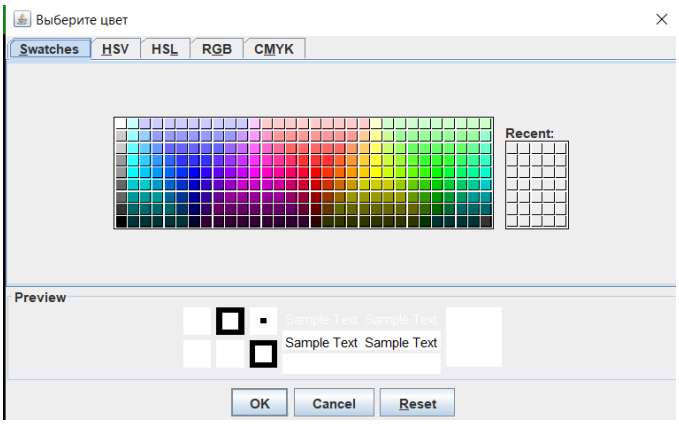
Пример запуска программы:

Была осуществлена проверка функционала приложения:

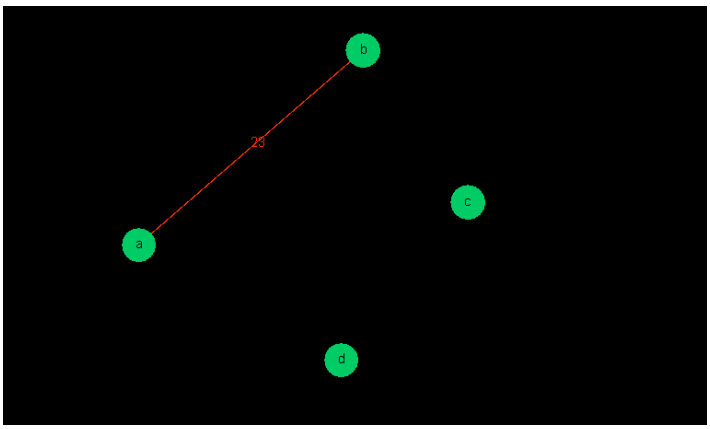
- Чтение из файла: проверка выполнялась ещё в предыдущем пункте тестирования;



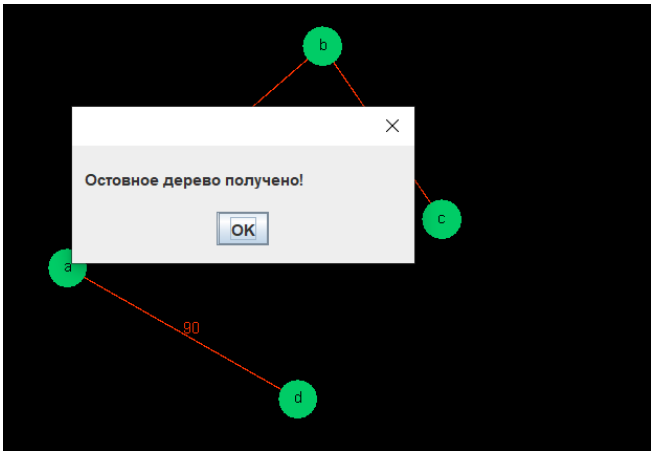
- Выбор цвета вершин/рёбер:



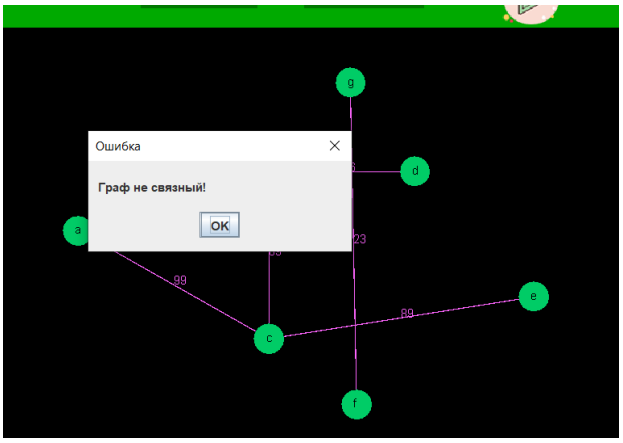
- Кнопки вперёд-назад:

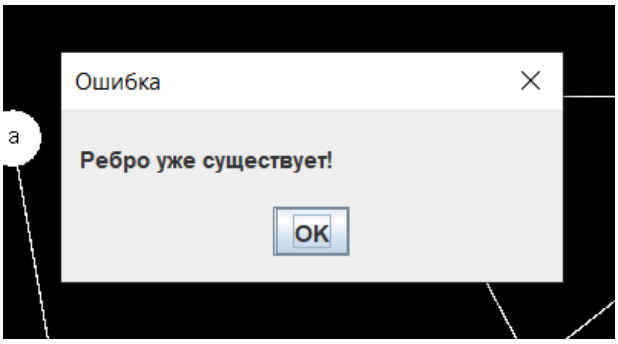


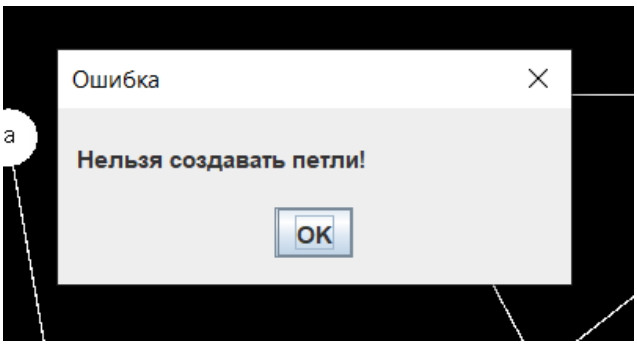
- Кнопка мгновенного запуска:

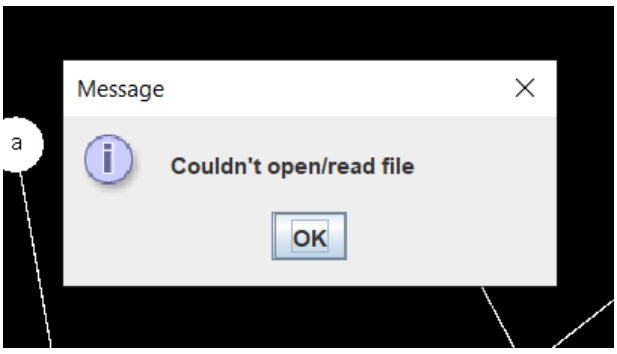


Обработка возможных ошибок:









ЗАКЛЮЧЕНИЕ

В результате работы было написано приложение, визуализирующее работу алгоритма Краскала. Были расширены знания в области объектно-ориентированного программирования, а также получены навыки работы с языком программирования Java и знакомство с написанием GUI на Java Swing.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://refactoring.guru/ru/design-patterns/abstract-factory
2. https://habr.com/ru/post/169381/
3. https://vertex-academy.com/tutorials/ru/samouchitel-po-java-s-nulya/
4. https://javarush.ru/quests/lectures/questsyntax.level00.lecture00

ПРИЛОЖЕНИЕ А

НАЗВАНИЕ ПРИЛОЖЕНИЯ

```
package GUI;
import Graph.*;
//import Graph.Graph;
//import Graph.Node;
//import Graph.Edge;
import Kruskal.*;
import javax.swing.*;
//import javax.swing.colorchooser.AbstractColorChooserPanel;
import java.awt.*;
import java.awt.event.*;
import java.nio.channels.AlreadyConnectedException;
import java.util.ArrayList;
//import java.awt.event.ActionEvent;
//import java.awt.event.ActionListener;

public class GUI extends JFrame {
    private JButton saveButton = new JButton(new ImageIcon("src\\save.png"));
    private JButton loadButton = new JButton(new ImageIcon("src\\load.png"));

    private JButton colorNodes = new JButton("Цвет вершин");
    private JButton colorEdge = new JButton("Цвет ребер");
    private JButton colorFlood = new JButton("Стереть");
    private JButton thickness = new JButton("Справка");

    private JButton buttonStart = new JButton(new ImageIcon("src\\start.png"));
    private JButton buttonSkip = new JButton(new ImageIcon("src\\skip.png"));
    private JButton nextStep = new JButton(new ImageIcon("src\\next.png"));
    private JButton prevStep = new JButton(new ImageIcon("src\\prev.png"));
```

```
GraphFactory factoryGraph = new GraphFactory();
NodeFactory factoryNode = new NodeFactory();
EdgeFactory factoryEdge = new EdgeFactory();

boolean edgeAddFlag = false;
Node saveNode;
int index = 0;
int collisionBreaker = 2;

public GUI() {
   super("GraphAnalyzer Algorithm");

   this.setBounds(500,200,1000,700);
   this.getContentPane().setLayout(new GridBagLayout());
   this.getContentPane().setBackground(new Color(255, 255, 255));
   this.setResizable(true);
   this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);

   JPanel panel1 = new JPanel();
   panel1.setLayout(new GridBagLayout());
   panel1.setBackground(new Color(0, 170, 0));

   GridBagConstraints c = new GridBagConstraints();

   c.fill = GridBagConstraints.NONE;
   c.anchor = GridBagConstraints.CENTER;
   c.gridx = 0;
   c.gridy = 0;
   c.gridheight = 1;
```

```
c.gridwidth = 1;
c.ipadx = -50;
c.ipady = 0;
c.weightx = 1;
c.weighty = 1;
c.insets = new Insets(2, 0, 0, 0);

saveButton.setBackground(new Color(0, 170, 0));
saveButton.setBorderPainted(false);
saveButton.setFocusPainted(false);

panel1.add(saveButton, c);

c.fill = GridBagConstraints.NONE;
c.anchor = GridBagConstraints.CENTER;
c.gridx = 0;
c.gridy = 1;
c.gridheight = 1;
c.gridwidth = 1;
c.ipadx = -50;
c.ipady = 0;
c.weightx = 1;
c.weighty = 1;
c.insets = new Insets(1, 0, 2, 0);

loadButton.setBackground(new Color(0, 170, 0));
loadButton.setBorderPainted(false);
loadButton.setFocusPainted(false);

panel1.add(loadButton, c);
```

```
JPanel panel2 = new JPanel();
panel2.setLayout(new GridBagLayout());
panel2.setBackground(new Color(0, 170, 0));

c.fill = GridBagConstraints.NONE;
c.anchor = GridBagConstraints.CENTER;
c.gridx = 0;
c.gridy = 0;
c.gridheight = 1;
c.gridwidth = 1;
c.ipadx = 5;
c.ipady = 0;
c.weightx = 1;
c.weighty = 1;
c.insets = new Insets(0, 0, 0, 0);

colorNodes.setBackground(new Color(0, 120, 0));
colorNodes.setForeground(new Color(255, 255, 255));
colorNodes.setBorderPainted(false);
colorNodes.setFocusPainted(false);

panel2.add(colorNodes, c);

c.fill = GridBagConstraints.NONE;
c.anchor = GridBagConstraints.CENTER;
c.gridx = 1;
c.gridy = 0;
c.gridheight = 1;
c.gridwidth = 1;
```

```
c.ipadx = 20;
c.ipady = 0;
c.weightx = 1;
c.weighty = 1;
c.insets = new Insets(0, 0, 0, 0);

colorEdge.setBackground(new Color(0, 120, 0));
colorEdge.setForeground(new Color(255, 255, 255));
colorEdge.setBorderPainted(false);
colorEdge.setFocusPainted(false);

panel2.add(colorEdge, c);

c.fill = GridBagConstraints.NONE;
c.anchor = GridBagConstraints.CENTER;
c.gridx = 0;
c.gridy = 1;
c.gridheight = 1;
c.gridwidth = 1;
c.ipadx = 35;
c.ipady = 0;
c.weightx = 1;
c.weighty = 1;
c.insets = new Insets(0, 0, 0, 0);

colorFlood.setBackground(new Color(0, 120, 0));
colorFlood.setForeground(new Color(255, 255, 255));
colorFlood.setBorderPainted(false);
colorFlood.setFocusPainted(false);
```

```
panel2.add(colorFlood, c);

c.fill = GridBagConstraints.NONE;
c.anchor = GridBagConstraints.CENTER;
c.gridx = 1;
c.gridy = 1;
c.gridheight = 1;
c.gridwidth = 1;
c.ipadx = 35;
c.ipady = 0;
c.weightx = 1;
c.weighty = 1;
c.insets = new Insets(0, 0, 0, 0);

thickness.setBackground(new Color(0, 120, 0));
thickness.setForeground(new Color(255, 255, 255));
thickness.setBorderPainted(false);
thickness.setFocusPainted(false);

panel2.add(thickness, c);

JPanel panel3 = new JPanel();
panel3.setLayout(new GridBagLayout());
panel3.setBackground(new Color(0, 170, 0));

c.fill = GridBagConstraints.NONE;
c.anchor = GridBagConstraints.CENTER;
c.gridx = 0;
c.gridy = 0;
c.gridheight = 1;
```

```
c.gridwidth = 1;
c.ipadx = -50;
c.ipady = 0;
c.weightx = 1;
c.weighty = 1;
c.insets = new Insets(2, 0, 1, 0);

prevStep.setBackground(new Color(0, 170, 0));
prevStep.setBorderPainted(false);
prevStep.setFocusPainted(false);

panel3.add(prevStep, c);

c.fill = GridBagConstraints.NONE;
c.anchor = GridBagConstraints.CENTER;
c.gridx = 1;
c.gridy = 0;
c.gridheight = 1;
c.gridwidth = 1;
c.ipadx = -50;
c.ipady = 0;
c.weightx = 1;
c.weighty = 1;
c.insets = new Insets(2, 0, 1, 0);

nextStep.setBackground(new Color(0, 170, 0));
nextStep.setBorderPainted(false);
nextStep.setFocusPainted(false);

panel3.add(nextStep, c);
```

```
c.fill = GridBagConstraints.NONE;
c.anchor = GridBagConstraints.CENTER;
c.gridx = 0;
c.gridy = 1;
c.gridheight = 1;
c.gridwidth = 1;
c.ipadx = -50;
c.ipady = 0;
c.weightx = 1;
c.weighty = 1;
c.insets = new Insets(1, 0, 4, 0);

buttonStart.setBackground(new Color(0, 170, 0));
buttonStart.setBorderPainted(false);
buttonStart.setFocusPainted(false);

panel3.add(buttonStart, c);

c.fill = GridBagConstraints.NONE;
c.anchor = GridBagConstraints.CENTER;
c.gridx = 1;
c.gridy = 1;
c.gridheight = 1;
c.gridwidth = 1;
c.ipadx = -50;
c.ipady = 0;
c.weightx = 1;
c.weighty = 1;
c.insets = new Insets(0, 0, 2, 0);
```

```
buttonSkip.setBackground(new Color(0, 170, 0));
buttonSkip.setBorderPainted(false);
buttonSkip.setFocusPainted(false);

panel3.add(buttonSkip, c);

JPanel panel = new JPanel();
panel.setLayout(new GridLayout(1,3,0,0));
panel.add(panel1);
panel.add(panel2);
panel.add(panel3);

c.fill = GridBagConstraints.HORIZONTAL;
c.anchor = GridBagConstraints.NORTH;
c.gridx = 0;
c.gridy = 0;
c.gridheight = 1;
c.gridwidth = 1;
c.ipadx = 0;
c.ipady = -55;
c.weightx = 0;
c.weighty = 0;
c.insets = new Insets(0, 0, 0, 0);

this.add(panel, c);

jPanel2 holst = new jPanel2();

holst.setLayout(null);
```

```
        holst.setBackground(new Color(0, 0, 0));

        c.fill = GridBagConstraints.BOTH;
        c.anchor = GridBagConstraints.CENTER;
        c.gridx = 0;
        c.gridy = 1;
        c.gridheight = 1;
        c.gridwidth = 1;
        c.ipadx = 0;
        c.ipady = 0;
        c.weightx = 1;
        c.weighty = 1;
        c.insets = new Insets(0, 0, 0, 0);

        this.add(holst, c);

        thickness.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null, "1. Для добавления вершины нажмите на холст.\n" +
                        "2. Для добавления ребра нажмите сначала на первую вершину, затем на вторую. После чего введите вес ребра.\n" +
                        "3. Для пошагового выполнения алгоритма нажмите на кнопку \"Play\".\nУправляйте процессом при помощи стрелок. " +
                        "Стрелка влево - шаг назад. Стрелка вправо - шаг вперед.\n" +
                        "4. Для мгновенного получения ответа нажмите на кнопку \"Skip\".\n" +
                        "5. Вы можете выбрать цвет вершин и ребер, нажав на соответствующие кнопки на панеле.\n" +
```

```
                "6. Вы можете стереть все с холста, нажав на кнопку
\"Стереть\".\n" +
                "7. Вы можете произвести ввод из файла и вывод в файл, нажав
на соответствующие кнопки на панеле.", "Справка",
JOptionPane.PLAIN_MESSAGE);
            }
        });

        loadButton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e) {
                FileDialog fd = new FileDialog((Dialog) null, "Выберите файл",
FileDialog.LOAD);
                fd.setDirectory("C:\\");
                fd.setVisible(true);

                String filename = fd.getFile();

                if (filename == null){
                    return;
                }

                Graph loadedGraph = Graph.load(filename);

                holst.testList = loadedGraph.getNodeList();
                holst.testListEdges = loadedGraph.getEdgeList();

                factoryEdge = new EdgeFactory();
                factoryGraph = new GraphFactory();
                factoryNode = new NodeFactory();
                holst.repaint();
```

```
        }

    });

    saveButton.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e) {
            FileDialog fd = new FileDialog((Dialog) null, "Выберите файл",
FileDialog.SAVE);
            fd.setDirectory("C:\\");
            fd.setVisible(true);

            String filename = fd.getFile();

            if (filename == null){
                return;
            }

            Graph toBeSaved = factoryGraph.getGraph(holst.testListEdges,
holst.testList);

            toBeSaved.save(filename);
        }

    });

    colorEdge.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            Color color=JColorChooser.showDialog(null, "Выберите цвет", new
Color(255,255,255));
```

```
            if (color!=null)
                holst.colorEdge = color;

            holst.repaint();
        }
    });

    colorNodes.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            Color color=JColorChooser.showDialog(null, "Выберите цвет", new
Color(255,255,255));

            if (color!=null)
                holst.colorNode = color;

            holst.repaint();
        }
    });

    buttonStart.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            index = 0;

            holst.after = new ArrayList<Edge>();

            Graph ready = new Graph(holst.testListEdges, holst.testList);

            if ((ready.isConnected())) {
                GraphAnalyzer analyzer= new GraphAnalyzer();
                ready = analyzer.KruskalAnalyze(ready);
```

```
                holst.testListEdges = new ArrayList<Edge>();

                holst.after = ready.getEdgeList();
            }
            else
                JOptionPane.showMessageDialog(null, "Граф не связный!",
"Ошибка", JOptionPane.PLAIN_MESSAGE);

            holst.repaint();
        }
    });

    nextStep.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            if (!holst.after.isEmpty()) {
                try {
                    holst.testListEdges.add(holst.after.get(index));
                    ++index;

                    holst.repaint();
                }
                catch (Exception x) {
                    JOptionPane.showMessageDialog(null, "Остовное дерево
получено!", "", JOptionPane.PLAIN_MESSAGE);
                }
            }
            else
                JOptionPane.showMessageDialog(null, "Алгоритм еще не применен!",
"Ошибка", JOptionPane.PLAIN_MESSAGE);
```

```
        }
    });

    prevStep.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            if (!holst.after.isEmpty()) {
                try {
                    holst.testListEdges.remove(index - 1);
                    --index;
                    holst.repaint();
                } catch (Exception x) {
                    JOptionPane.showMessageDialog(null, "Вы на начальном шаге!",
"", JOptionPane.PLAIN_MESSAGE);
                }
            }
            else
                JOptionPane.showMessageDialog(null, "Алгоритм еще не применен!",
"Ошибка", JOptionPane.PLAIN_MESSAGE);
        }
    });

    buttonSkip.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e) {
            Graph ready;

            if (holst.after.isEmpty())
                ready = new Graph(holst.testListEdges, holst.testList);
            else {
                ready = new Graph(holst.after, holst.testList);
```

```
                index = holst.after.size();
            }

            if ((ready.isConnected())) {
                GraphAnalyzer analyzer= new GraphAnalyzer();
                ready = analyzer.KruskalAnalyze(ready);

                holst.testListEdges = new ArrayList<Edge>();
                holst.after = ready.getEdgeList();

                index = holst.after.size();

                for (Edge mda : holst.after)
                    holst.testListEdges.add(mda);
            }
            else
                JOptionPane.showMessageDialog(null, "Граф не связный!",
"Ошибка", JOptionPane.PLAIN_MESSAGE);

            holst.repaint();
        }

    });

    colorFlood.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            holst.testListEdges = new ArrayList<Edge>();
            holst.testList = new ArrayList<Node>();

            factoryEdge = new EdgeFactory();
```

```
            factoryNode = new NodeFactory();
            factoryGraph = new GraphFactory();
            holst.repaint();
        }
    });

    holst.addMouseListener(new MouseAdapter() {
        public void mousePressed(MouseEvent evt) {
            boolean flag = true;

            for (Node v : holst.testList){
                if  (Math.sqrt((v.getX()-evt.getX())*(v.getX()-evt.getX())+(v.getY()-
evt.getY())*(v.getY()-evt.getY()))<15){
                    flag = false;

                    if (edgeAddFlag) {
                        try{
                            String string = JOptionPane.showInputDialog(null, "Введите вес
ребра", "", JOptionPane.PLAIN_MESSAGE);

                            int askedWeight;

                            if (string != null) {

                                for (Edge e : holst.testListEdges)
                                {
                                    if((e.getFirst() == saveNode) && (e.getSecond() == v) ||
                                            (e.getFirst() == v) && (e.getSecond() == saveNode)){
```

```
                    JOptionPane.showMessageDialog(null, "Ребро уже существует!", "Ошибка", JOptionPane.PLAIN_MESSAGE);
                    edgeAddFlag = false;
                }

                if((saveNode == v)){
                JOptionPane.showMessageDialog(null, "Нельзя создавать петли!", "Ошибка", JOptionPane.PLAIN_MESSAGE);

                }
                }



                if(edgeAddFlag) {
                    askedWeight = Integer.parseInt(string);
                    holst.testListEdges.add(factoryEdge.getEdge(saveNode, v, askedWeight));
                }
            }

            edgeAddFlag = false;
            break;
        }
        catch(NumberFormatException e){
            JOptionPane.showMessageDialog(null, "Требуется целочисленное значение!", "Ошибка", JOptionPane.PLAIN_MESSAGE);

            edgeAddFlag = false;
            break;
```

```
                }
                finally{
                }
            }

            saveNode = v;
            edgeAddFlag = true;
            break;
        }

        else if (Math.sqrt((v.getX()-evt.getX())*(v.getX()-evt.getX())+(v.getY()-
evt.getY())*(v.getY()-evt.getY()))<30){
            JOptionPane.showMessageDialog(null, "Вершины пересекаются!",
"Ошибка", JOptionPane.PLAIN_MESSAGE);

            flag = false;
            edgeAddFlag = false;
            break;
        }
    }
     if (flag) {
        Node tmp = factoryNode.getNode();

        boolean foundName = false;

        while(!foundName) {
            foundName = true;

            for (Node e : holst.testList) {
                if (e.getName().equals(tmp.getName())) {
```

```
                    tmp.setName(tmp.getName() + (collisionBreaker++));

                    foundName = false;
                    continue;
                }
            }
        }

        collisionBreaker = 2;

        tmp.setX(evt.getX());
        tmp.setY(evt.getY());

        holst.testList.add(tmp);

        edgeAddFlag = false;
    }

    holst.repaint();
  }
});

this.addWindowListener(new WindowListener() {
  public void windowActivated(WindowEvent event) {}
  public void windowClosed(WindowEvent event) {}
  public void windowDeactivated(WindowEvent event) {}
  public void windowDeiconified(WindowEvent event) {}
  public void windowIconified(WindowEvent event) {}
  public void windowOpened(WindowEvent event) {}
  public void windowClosing(WindowEvent event) {
```

```
            Object[] options = { "Да", "Нет!" };

            int rc = JOptionPane.showOptionDialog(
                    event.getWindow(), "Вы действительно хотите выйти?",
                    "Требуется подтверждение", JOptionPane.DEFAULT_OPTION,
                    JOptionPane.PLAIN_MESSAGE, null, options, options[0]);

            if (rc == 0) {
               event.getWindow().setVisible(false);
               System.exit(0);
            }
         }

      });
   }

   public class jPanel2 extends JPanel {
      ArrayList<Node> testList = new ArrayList<Node>();
      ArrayList<Edge> testListEdges = new ArrayList<Edge>();
      ArrayList<Edge> after = new ArrayList<Edge>();

      Color colorEdge = new Color(255, 255, 255);
      Color colorNode = new Color(255, 255, 255);
      Color black = new Color(0, 0, 0);

      public void paintComponent(Graphics g) {
         super.paintComponent(g);

         g.setColor(colorEdge);
```

```
            for (Edge p : testListEdges){
                g.drawLine(p.getFirst().getX(), p.getFirst().getY(), p.getSecond().getX(),
p.getSecond().getY());

                g.drawString(Integer.toString(p.getWeight()),
                        (p.getFirst().getX() + p.getSecond().getX()) / 2 ,
                        (p.getFirst().getY() + p.getSecond().getY()) / 2 );
            }
            for (Node a : testList){
                g.setColor(colorNode);
                g.fillOval(a.getX() - 15, a.getY() - 15, 30, 30);

                g.setColor(black);
                g.drawString(a.getName(), a.getX() - 3, a.getY() + 3);
            }

        }
    }

    public static void execute() {
        GUI app = new GUI();
        app.setVisible(true);
    }

}
```

```
package GUI;

import Graph.Edge;
import Graph.Node;

public class EdgeFactory extends EdgeFactoryInterface{
    @Override
    public Edge getEdge(){
        return new Edge();
    }


    @Override
    public Edge getEdge(Node first, Node second, Integer weight)
    {
        return new Edge(first, second, weight);
    }
}


package GUI;

import Graph.Edge;
import Graph.Node;

public abstract class EdgeFactoryInterface {
    public abstract Edge getEdge();

    public abstract Edge getEdge(Node first, Node second, Integer weight);
```

```
}

package GUI;

import Graph.Edge;
import Graph.Graph;
import Graph.Node;

import java.util.ArrayList;

public class GraphFactory extends GraphFactoryInterface {
    @Override
    public Graph getGraph(){
        return new Graph();
    }

    @Override
    public Graph getGraph(ArrayList<Edge> edgeList, ArrayList<Node> nodeList){
        return new Graph(edgeList, nodeList);
    }

    @Override
    public Graph getGraph(ArrayList<Edge> edgeList) {
        return new Graph(edgeList);
    }

}

package GUI;
```

```
import Graph.Edge;
import Graph.Graph;
import Graph.Node;

import java.util.ArrayList;

public abstract class GraphFactoryInterface {
   public abstract Graph getGraph();

   public abstract Graph getGraph(ArrayList<Edge> edgeList, ArrayList<Node>
nodeList);

   public abstract Graph getGraph(ArrayList<Edge> edgeList);
}

package GUI;

import Graph.Node;

public class NodeFactory extends NodeFactoryInterface {

   @Override
   public Node getNode(){
      return new Node(String.valueOf(name++));
   }

   @Override
   public Node getNode(String name){
      return new Node(name);
   }
```

```
}

package GUI;

import Graph.Node;

public abstract class NodeFactoryInterface {
    protected char name = 'a';

    public abstract Node getNode();
    public abstract Node getNode(String name);
}


package Graph;

import java.util.Comparator;
import java.util.Objects;

public class Edge {
    private Node first;
    private Node second;
    private Integer weight;

    public Edge(){
        first = null;

        second = null;

        weight = null;
```

```
}

public Edge(Node first, Node second, Integer weight){
   this.first = first;
   this.second = second;
   this.weight = weight;
}

public void setSecond(Node second) {
   this.second = second;
}

public void setFirst(Node first) {
   this.first = first;
}

public Node getSecond() {
   return second;
}

public Node getFirst() {
   return first;
}

public void setWeight(int weight) {
   this.weight = weight;
}

public int getWeight() {
   return weight;
```

```
    }

    static public class EdgeComparator implements Comparator<Edge> {

        @Override
        public int compare(Edge o1, Edge o2) {
            return o1.getWeight() - o2.getWeight();
        }
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Edge edge = (Edge) o;
        return Objects.equals(first, edge.first) &&
                Objects.equals(second, edge.second) &&
                Objects.equals(weight, edge.weight);
    }

    @Override
    public int hashCode() {
        return Objects.hash(first, second, weight);
    }
}

package Graph;

import GUI.EdgeFactory;
import GUI.GraphFactory;
```

```
import GUI.NodeFactory;

import javax.swing.*;
import java.util.Queue;
import java.util.LinkedList;
import java.io.*;

import java.util.ArrayList;

public class Graph {

    private ArrayList<Node> nodeList;
    private ArrayList<Edge> edgeList;

    public Graph(){
        nodeList = new ArrayList<Node>();
        edgeList = new ArrayList<Edge>();
    }

    public Graph(ArrayList<Edge> edgeList, ArrayList<Node> nodeList){
        this.edgeList = edgeList;
        this.nodeList = nodeList;
    }

    public Graph(ArrayList<Edge> edgeList){
        this.edgeList = edgeList;

        this.nodeList = new ArrayList<Node>();

        for (Edge e : edgeList){
```

```
        if (!nodeList.contains(e.getFirst())){
            nodeList.add(e.getFirst());
        }


        if (!nodeList.contains(e.getSecond())){
            nodeList.add(e.getSecond());
        }
    }
}

public void setEdgeList(ArrayList<Edge> edgeList) {
    this.edgeList = edgeList;
}

public ArrayList<Edge> getEdgeList() {
    return edgeList;
}

public void setNodeList(ArrayList<Node> nodeList) {
    this.nodeList = nodeList;
}

public ArrayList<Node> getNodeList() {
    return nodeList;
}

public ArrayList<Node> adjacentNodes(Node e){
    ArrayList<Node> result = new ArrayList<Node>();
```

```
    for (Edge edge : edgeList){
        if (edge.getFirst() == e) {
            result.add(edge.getSecond());
        }

        if  (edge.getSecond() == e) {
            result.add(edge.getFirst());
        }
    }

    return result;
}

public boolean isConnected(){
    Queue<Node> currentQueue = new LinkedList<Node>();
    Node current = nodeList.get(0);

    ArrayList<Node> adjacentToCurrent = adjacentNodes(current);

    for(Node e : adjacentToCurrent){
        if (!e.isVisited()){
            currentQueue.add(e);
        }
    }

    current.setVisited(true);

    while(!currentQueue.isEmpty()){
        current = currentQueue.poll();
```

```
        adjacentToCurrent = adjacentNodes(current);

        for(Node e : adjacentToCurrent){
            if (!e.isVisited()){
                currentQueue.add(e);
            }
        }

        current.setVisited(true);
    }

    for(Node e : nodeList) {
        if (!e.isVisited()) {
            return false;
        }
    }

    return true;
}

public void save(String filename){

    BufferedWriter writer;
    NodeFactory factoryNode = new NodeFactory();
    EdgeFactory factoryEdge = new EdgeFactory();
    GraphFactory factoryGraph = new GraphFactory();

    try{
        writer = new BufferedWriter(new FileWriter(filename));
```

```
        for(Edge e : edgeList){
            String toPut = e.getFirst().getName() + " " + e.getSecond().getName() + " "
                    + e.getWeight() + "\n";

            writer.write(toPut);
        }

        for(Node e : nodeList){
            if(adjacentNodes(e).size() == 0){
                String toPut = e.getName() + "\n";

                writer.write(toPut);
            }
        }

        writer.close();

    } catch (FileNotFoundException e) {
        JOptionPane.showMessageDialog(null, "Couldn't open/read file");

        return;
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, "Couldn't open/read file");

        return;
    } finally {}

}

public static Graph load(String filename){
```

```
BufferedReader reader;
NodeFactory factoryNode = new NodeFactory();
EdgeFactory factoryEdge = new EdgeFactory();
GraphFactory factoryGraph = new GraphFactory();

try{
   reader = new BufferedReader(new FileReader(filename));

   ArrayList<Edge> edgeList = new ArrayList<Edge>();

   String line = reader.readLine();

   ArrayList<Node> originalNodes = new ArrayList<Node>();

   while(line != null){
      String [] parsed = line.split("[\\s+]");

      Node first = null, second = null;

      for(Node n : originalNodes){
         if (n.getName().equals(parsed[0])){
            first = n;
         }

         if (n.getName().equals(parsed[1])){
            second = n;
         }
      }

      if(first == null){
```

```
                first = (factoryNode).getNode(parsed[0]);
                originalNodes.add(first);
            }


            if(second == null){
                second = factoryNode.getNode(parsed[1]);
                originalNodes.add(second);
            }

            edgeList.add(factoryEdge.getEdge(first, second,
Integer.parseInt(parsed[2])));

            line = reader.readLine();
        }

        Graph result = factoryGraph.getGraph(edgeList);

        double angle = 0;

        int[] x = new int[result.getNodeList().size()];
        int[] y = new int[result.getNodeList().size()];

        for(int i = 0 ; i < result.getNodeList().size()  ;++i)
        {
            angle = i * (360/result.getNodeList().size());

            x[i] = (int) (550 + 200 * Math.cos(Math.toRadians(angle)));
            y[i] = (int) (250 + 200 * Math.sin(Math.toRadians(angle)));
```

```
            }

            int i = 0;

            for(Node e : result.getNodeList())
            {
               e.setX(x[i]);
               e.setY(y[i]);

               i++;
            }

            return result;
         } catch (FileNotFoundException e) {
            JOptionPane.showMessageDialog(null, "Couldn't open/read file");

            return null;
         } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "Couldn't open/read file");

            return null;
         } finally {}

      }
   }
```

```
package Graph;

import Graph.GUIdata.NodeGUIdata;
```

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

public class Node extends NodeGUIdata {
    protected String name;

    public Node(){
        name = null;
        visited = false;
    }

    public Node(String name){
        this.name = name;
        visited = false;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }


    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
```

```
        if (o == null || getClass() != o.getClass()) return false;
        Node node = (Node) o;
        return Objects.equals(name, node.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name);
    }
}
```

```
package Graph.GUIdata;

public class NodeGUIdata {
    protected boolean visited = false;
    protected int x = 0;
    protected int y = 0;

    protected Integer unionIndex = null;

    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }

    public int getX() {
```

```
        return x;
    }

    public int getY() {
        return y;
    }

    public void setVisited(boolean visited) {
        this.visited = visited;
    }

    public boolean isVisited() {
        return visited;
    }

    public Integer getUnionIndex() {
        return this.unionIndex;
    }

    public void setUnionIndex(Integer unionIndex) {
        this.unionIndex = unionIndex;
    }


}
package Kruskal;


import Graph.*;
import GUI.*;
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

public class GraphAnalyzer implements Alghoritms {
   @Override
   public Graph KruskalAnalyze(Graph graph){
      GraphFactory factoryGraph = new GraphFactory();

      ArrayList<Edge> result = new ArrayList<Edge>();

      Map<Integer, ArrayList<Node>> Unions = new HashMap<Integer,
ArrayList<Node>>();

      graph.getEdgeList().sort(new Edge.EdgeComparator());

      int i = 0;

      for (Edge e : graph.getEdgeList()){
         if ((e.getFirst().getUnionIndex() == null) && (e.getSecond().getUnionIndex()
== null)){
            result.add(e);

            Unions.put(i, new ArrayList<Node>());
            Unions.get(i).add(e.getFirst());
            Unions.get(i).add(e.getSecond());

            e.getFirst().setUnionIndex(i);
            e.getSecond().setUnionIndex(i);
```

```
            ++i;
        }

        else if((e.getFirst().getUnionIndex() != null) &&
(e.getSecond().getUnionIndex() == null)){
            result.add(e);

            Unions.get(e.getFirst().getUnionIndex()).add(e.getSecond());

            e.getSecond().setUnionIndex(e.getFirst().getUnionIndex());
        }

        else if((e.getFirst().getUnionIndex() == null) &&
(e.getSecond().getUnionIndex() != null)){
            result.add(e);

            Unions.get(e.getSecond().getUnionIndex()).add(e.getFirst());

            e.getFirst().setUnionIndex(e.getSecond().getUnionIndex());
        }

        else if (e.getFirst().getUnionIndex() == e.getSecond().getUnionIndex()) {
            continue;
        }

        else{
            result.add(e);

            for (Node n : Unions.get(e.getFirst().getUnionIndex())) {
```

```
                Unions.get(e.getSecond().getUnionIndex()).add(n);

                n.setUnionIndex(e.getSecond().getUnionIndex());
            }

        }

    }

    for (Node k : graph.getNodeList()){
        k.setUnionIndex(null);
    }

    for (Node k:graph.getNodeList()){
        k.setVisited(false);
    }

    return factoryGraph.getGraph(result);

  };

}
```