# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

# «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

**Кафедра МО ЭВМ**

# ОТЧЕТ

**по учебной практике**

**ТЕМА: ВИЗУАЛИЗАЦИЯ АЛГОРИТМА КРАСКАЛА**

|  |  |
| --- | --- |
| Студент гр. 9304 | Колотов М.А. |
| Студент гр. 9304 | Барышев А.А. |
| Студент гр. 9304 | Сорин А.В. |
| Руководитель | Фиалковский М.С. |

Санкт-Петербург 2021

# ЗАДАНИЕ

**НА УЧЕБНУЮ ПРАКТИКУ**

|  |  |  |
| --- | --- | --- |
| Студент Колотов М.А. группы 9304 |  |  |
| Студент Барышев А.А. группы 9304 | | |
| Студент Сорин А.В. группы 9304  Тема практики: визуализация алгоритма Краскала | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.  Алгоритм: алгоритм Краскала. | | |
| Сроки прохождения практики: 01.07.2021 – 14.07.2021 | | |
| Дата сдачи отчета: 14.07.2021 |  |  |
| Дата защиты отчета: 14.07.2021 |  |  |
| Студент |  | Колотов М.А. |
| Студент |  | Барышев А.А. |
| Студент |  | Сорин А.В. |
| Руководитель |  | Фиалковский М.С. |

# АННОТАЦИЯ

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

# SUMMARY

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

# СОДЕРЖАНИЕ

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Требования к программе | 6 |
| 1.1. | Исходные требования к программе\* | 6 |
| 1.1.1 | Требования к визуализации | 6 |
| 1.1.2 | Требования к входным данным | 7 |
| 1.1.3 | Требования к архитектуре | 8 |
| 1.2. | Уточнение требований после сдачи прототипа | 8 |
| 2. | План разработки и распределение ролей в бригаде | 9 |
| 2.1. | План разработки | 9 |
| 2.2. | Распределение ролей в бригаде | 9 |
| 3. | Особенности реализации | 10 |
| 3.1. | Структуры данных | 10 |
| 3.2. | Основные методы | 10 |
| 3.2.1 | Основные методы класса GraphViewer. | 12 |
| 3.2.2 | Основные методы класса View. | 12 |
| 3.2.3 | Основные методы класса Graph. | 12 |
| 3.2.4 | Основные методы класса GraphEdges. | 13 |
| 3.2.5 | Основные методы класса Kruskal. | 13 |
| 4. | Тестирование | 14 |
| 4.1 | Тестирование алгоритма. | 14 |
| 4.2 | Тестирование графа. | 14 |
| 4.3 | Тестирование итогового проекта. | 15 |
|  | Заключение | 16 |
|  | Список использованных источников | 17 |
|  | Приложение А. Исходный код – только в электронном виде | 18 |

**ВВЕДЕНИЕ**

Основная цель практики – реализовать визуализатор алгоритма Краскала. Алгоритм предназначен для построения минимального остовного дерева взвешенного связного неориентированного графа. Для реализации проекта, необходимо реализовать графический интерфейс, сам алгоритм и объединить данные наработки.

# ТРЕБОВАНИЯ К ПРОГРАММЕ

* 1. **Исходные требования к программе**

Приложение будет иметь графический интерфейс, через который будет возможно вводить данные через файл. Программа будет пошагово демонстрировать построение минимального остовного дерева с помощью алгоритма Краскала.

# Требования к визуализации

Окно программы будет выглядеть следующим образом:

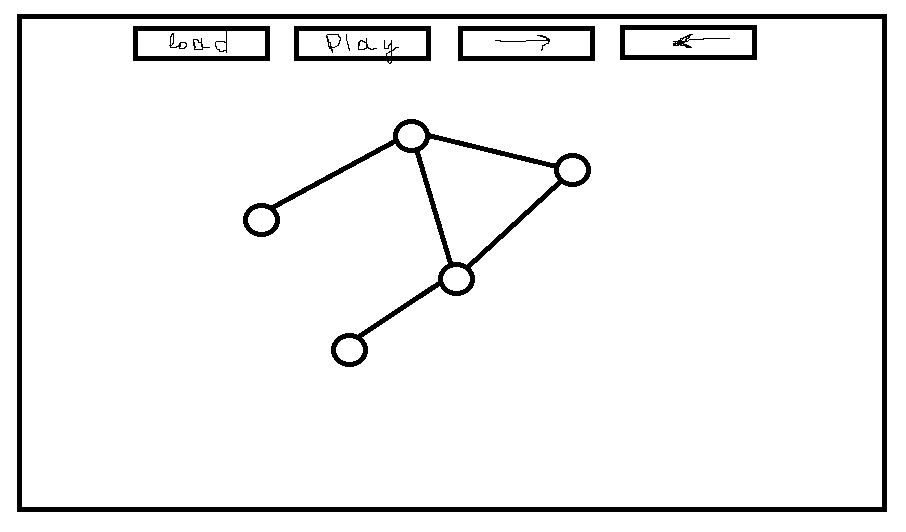


Рисунок 1 – Окно программы

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

Загрузка данных из файла.

Запуск алгоритма. Построится граф. Станут доступны кнопки шага.

Сделать шаг. Данная опция позволит шаг за шагом увидеть работу алгоритма, подсвечивая рёбра.

# Требования к входным данным

Входные данные можно задать, загрузив текстовый файл, содержащий список смежности графа, при помощи кнопки “Loading”. Если пользователь хочет загрузить текстовый файл, то он должен поместить туда данные следующим образом: на первой строке находится количество вершин, а затем рёбра с весом. В случае неверных данных в текстовом файле, программа получит исключение об ошибке и уведомит об этом пользователя, после чего завершится.

Исходя из алгоритма Краскала, требования к входным данным следующие: 1) веса рёбер следует подавать из множества вещественных чисел. Если требования нарушатся, то программа получит исключение об ошибке и ребро не создастся; 2) граф должен быть связным, если при запуске алгоритма будет несколько компонент связности, то программа получит исключение и завершится.

# Требования к архитектуре

Проект выполняется средствами языка программирования Java. Для визуализации используется библиотека Swing. Данная библиотека была выбрана потому, что предоставляет большой набор связанных с ней библиотек, которые в свою очередь позволяют визуализировать графы

Программа разделенная на следующие основные блоки: алгоритм, визуализация, структуры данных, тестирование. За реализацию алгоритма отвечают следующие классы:

public class Kruskal

public class AlgorithmSteps

За реализацию визуальной части отвечают следующие классы:

public class GraphViewer

public class View

За реализацию структур данных отвечают следующие классы:

public class Graph

public class GraphEdges

public class GraphVertices

За реализацию тестирования отвечают следующие классы:

public class GraphTest

public class KruskalTest

# Уточнение требований после попытки сдачи спецификации

Дописать требования к архитектуре.

1. **ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ**

# План разработки

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

До 07.07.2021 – Сдача вводного задания

До 07.07.2021 – Согласование спецификации. Создание прототипа графического интерфейса.

До 10.07.2021 – Сдача второго этапа

До 14.07.2021 – Сдача финальной версии мини-проекта

# Распределение ролей в бригаде

Колотов Максим – лидер, ведение документации.

Барышев Андрей – разработка и дизайн графического интерфейса.

Сорин Алексей – алгоритмист и тестировщик.

# ОСОБЕННОСТИ РЕАЛИЗАЦИИ

* 1. **Структуры данных**

В качестве структур данных используется два списка, список вершин и список рёбер. Данная структура данных используется в алгоритме для построения минимального остовного дерева.

**3.2 Основные методы**

На рисунке 2 представлена UML-диаграмма проекта.

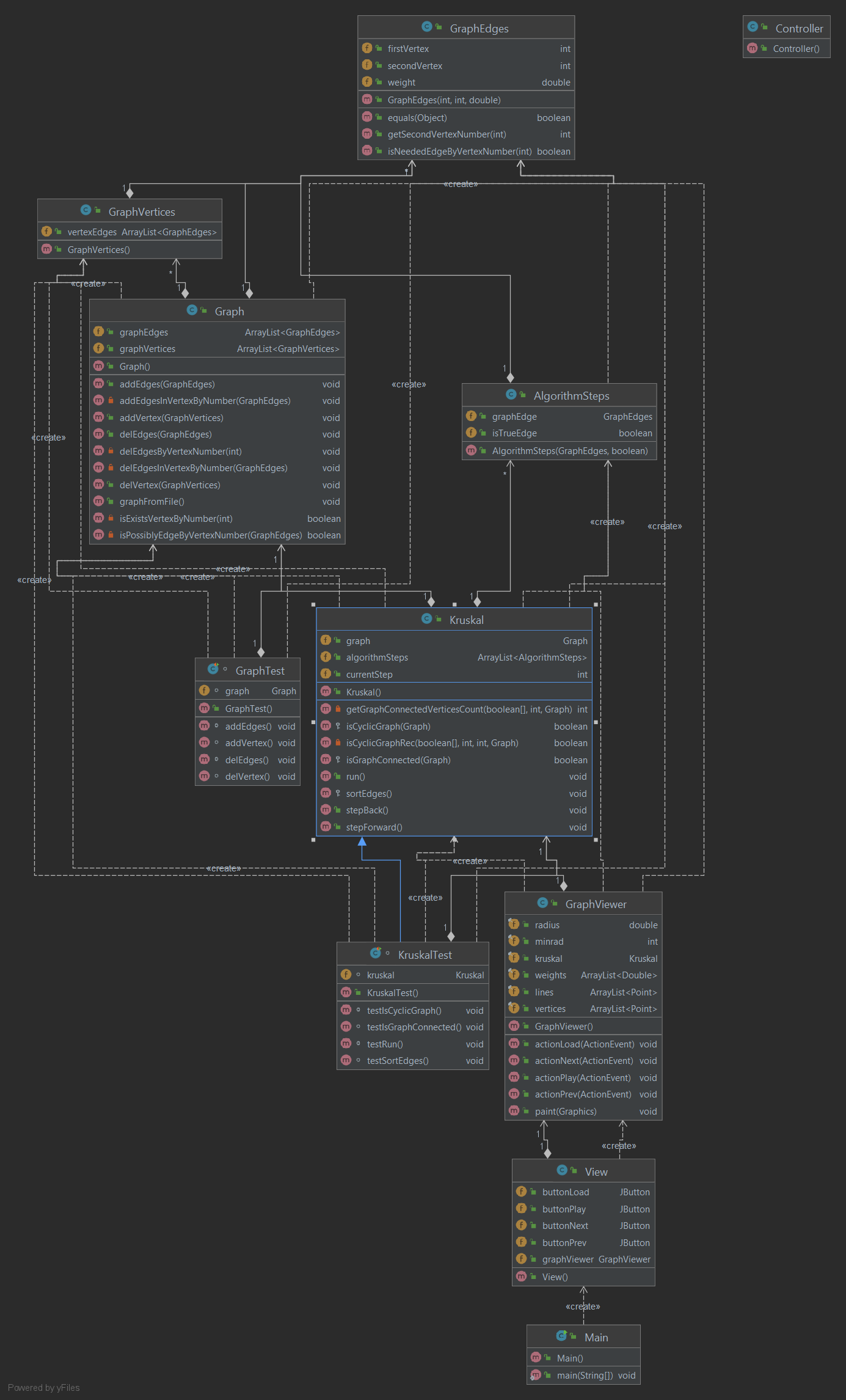


Рисунок 2 — UML-диаграмма

**3.2.1 Основные методы класса GraphViewer.**

Данный класс отвечает за визуализацию работ основных кнопок.

public void paint(Graphics g) – метод рисует граф.

public void actionLoad(ActionEvent e) – метод загружает граф из файла.

public void actionPlay(ActionEvent e) – метод запускает алгоритм Краскала.

public void actionNext(ActionEvent e) – метод, позволяющий перейти на следующий шаг алгоритма.

public void actionPrev(ActionEvent e) – метод, позволяющий перейти на прошлый шаг алгоритма.

**3.2.2 Основные методы класса View.**

Данный класс нужен для визуализации основного окна приложения, а так же всех используемых кнопок в нём.

public View() – метод для визуализации основного окна и кнопок.

**3.2.3 Основные методы класса Graph.**

Данный класс отвечает за хранение графа.  
public void addVertex(GraphVertices graphVertex) – метод добавляет вершину в граф.  
public void addEdges(GraphEdges graphEdge) – метод добавления ребра в граф.  
public void delEdges(GraphEdges graphEdge) – метод удаления ребра из графа.  
public void delVertex(GraphVertices graphVertex) – метод удаления вершины из графа.

public void graphFromFile() – метод загрузки графа из файла.

**3.2.4 Основные методы класса GraphEdges.**

Данный класс отвечает за хранение рёбер.  
public int getSecondVertexNumber(int vertexNumber) – метод получения номера смежной вершины.  
public boolean equals(Object obj) – метод сравнения рёбер.

**3.2.5 Основные методы класса Kruskal.**

Данный класс отвечает за работу алгоритма.  
protected void sortEdges() – метод сортировки рёбер.  
protected boolean isGraphConnected(Graph graph1) – метод проверки связности графа.  
protected boolean isCyclicGraph(Graph graph1) – метод проверки на цикл.  
public void run() – метод запуска алгоритма.  
public void stepForward() – метод шага алгоритма вперёд.  
public void stepBack() – метод шага алгоритма назад.

1. **ТЕСТИРОВАНИЕ**

# Тестирование алгоритма

Для тестирования алгоритма был написан класс KruskalTest, содержащий юнит- тесты, которые тестируют методы класса Kruskal. Тестировались следующие методы:

void testSortEdges() – проверялся результат сортировки рёбер по весам.

void testIsGraphConnected() – проверялась работа метода по проверке связности графа.

void testIsCyclicGraph() – проверялась работа метода по проверке графа на цикличность.

void testRun() – проверялось получает ли метод минимальное остовное дерево.

# Тестирование графа

Для тестирования графа был написан класс GraphTest, содержащий юнит- тесты, которые тестируют методы класса Graph. Тестировались следующие методы:

void addVertex() – проверялась работа метода по добавлению вершины в граф.

void addEdges() – проверялась работа метода по добавлению ребра в граф.

void delEdges() – проверялась работа метода по удалению ребра из графа.

void delVertex() – проверялась работа метода по удалению вершины из графа.

* 1. **Тестирование итогового проекта**

Интерфейс программы тестировался вручную, для этого в визуализаторе прогонялись несколько раз различные графы, для которых было необходимо найти минимальное остовное дерево. Попутно проверялся весь функционал программы, например загрузка графа из файла и шаги алгоритма.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения проекта была реализована программа, представляющая собой визуализатор алгоритма Краскала. Для реализации интерфейса использовалась библиотека Swing. На момент сдачи практики, все поставленные задачи были выполнены успешно.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. http://java-online.ru/libs-swing.xhtml

2. https://stepik.org/course/187/promo

3. https://habr.com/ru/post/77382/

4. Java 8 Полное руководство. Герберт Шилдт.

# ПРИЛОЖЕНИЕ А

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Файл AlgorithmSteps.java:

package com.github.elunmuks.BTS;

public class AlgorithmSteps {

public GraphEdges graphEdge;

public boolean isTrueEdge;

public AlgorithmSteps(GraphEdges graphEdge, boolean isTrueEdge) {

this.graphEdge = graphEdge;

this.isTrueEdge = isTrueEdge;

}

}

Файл Controller.java:

package com.github.elunmuks.BTS;

public class Controller {

}

Файл Graph.java:

package com.github.elunmuks.BTS;

import java.io.IOException;

import java.net.URISyntaxException;

import java.nio.file.Files;

import java.nio.file.Path;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Objects;

// main data structure of the program

public class Graph {

public ArrayList<GraphEdges> graphEdges;

public ArrayList<GraphVertices> graphVertices;

public Graph() {

graphEdges = new ArrayList<>();

graphVertices = new ArrayList<>();

}

private boolean isExistsVertexByNumber(int vertexNumber) {

return vertexNumber >= 0 && vertexNumber < graphVertices.size();

}

private boolean isPossiblyEdgeByVertexNumber(GraphEdges graphEdge) {

return isExistsVertexByNumber(graphEdge.firstVertex) && isExistsVertexByNumber(graphEdge.secondVertex);

}

public void addVertex(GraphVertices graphVertex) {

graphVertices.add(graphVertex);

}

private void addEdgesInVertexByNumber(GraphEdges graphEdge) {

graphVertices.get(graphEdge.firstVertex).vertexEdges.add(graphEdge);

graphVertices.get(graphEdge.secondVertex).vertexEdges.add(graphEdge);

}

private void delEdgesInVertexByNumber(GraphEdges graphEdge) {

graphVertices.get(graphEdge.firstVertex).vertexEdges.remove(graphEdge);

graphVertices.get(graphEdge.secondVertex).vertexEdges.remove(graphEdge);

}

public void addEdges(GraphEdges graphEdge) {

if (!isPossiblyEdgeByVertexNumber(graphEdge))

throw new RuntimeException("The edge has a non-existent vertex.\n");

for (GraphEdges graphEdges : this.graphEdges) {

if (graphEdge.equals(graphEdges))

return;

}

graphEdges.add(graphEdge);

addEdgesInVertexByNumber(graphEdge);

}

public void delEdges(GraphEdges graphEdge) {

if (!graphEdges.contains(graphEdge))

throw new RuntimeException("This edge not found.\n");

graphEdges.remove(graphEdge);

delEdgesInVertexByNumber(graphEdge);

}

private void delEdgesByVertexNumber(int vertexNumber) {

Iterator<GraphEdges> iterator = graphEdges.iterator();

while (iterator.hasNext()) {

GraphEdges rem = iterator.next();

if (rem.isNeededEdgeByVertexNumber(vertexNumber)) {

iterator.remove();

}

}

}

public void delVertex(GraphVertices graphVertex) {

int k;

if (!graphVertices.contains(graphVertex))

throw new RuntimeException("This vertex not found.\n");

k = graphVertices.indexOf(graphVertex);

graphVertices.remove(graphVertex);

delEdgesByVertexNumber(k);

}

public void graphFromFile() throws IOException, URISyntaxException {

Path path = Path.of(Objects.requireNonNull(getClass().getResource("/forGraph.txt")).toURI());

List<String> list = Files.readAllLines(path);

int vertexNumber = Integer.parseInt(list.get(0));

for (int i = 0; i < vertexNumber; i++) {

addVertex(new GraphVertices());

}

for (String str : list) {

if (str.compareTo(list.get(0)) == 0)

continue;

String[] S = str.split(" ");

addEdges(new GraphEdges(Integer.parseInt(S[0]), Integer.parseInt(S[1]), Double.parseDouble(S[2])));

}

}

}

Файл GraphEdges.java:

package com.github.elunmuks.BTS;

//

public class GraphEdges {

public int firstVertex;

public int secondVertex;

public double weight;

public GraphEdges(int firstVertex, int secondVertex, double weight) {

this.firstVertex = firstVertex;

this.secondVertex = secondVertex;

this.weight = weight;

}

public boolean isNeededEdgeByVertexNumber(int vertexNumber) {

return vertexNumber == firstVertex || vertexNumber == secondVertex;

}

public int getSecondVertexNumber(int vertexNumber) {

int res = (vertexNumber == firstVertex) ? secondVertex : (vertexNumber == secondVertex) ? firstVertex : -1;

if (res == -1)

throw new RuntimeException("Vertex number does not belong to edge.");

return res;

}

@Override

public boolean equals(Object obj) {

if (obj instanceof GraphEdges) {

return ((GraphEdges) obj).firstVertex == firstVertex &&

((GraphEdges) obj).secondVertex == secondVertex && ((GraphEdges) obj).weight == weight;

}

return false;

}

}

Файл GraphVertices.java:

package com.github.elunmuks.BTS;

import java.util.ArrayList;

public class GraphVertices {

public ArrayList<GraphEdges> vertexEdges;

public GraphVertices() {

vertexEdges = new ArrayList<>();

}

}

Файл GraphViewer.java:

package com.github.elunmuks.BTS;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.io.IOException;

import java.net.URISyntaxException;

import java.util.ArrayList;

public class GraphViewer extends JPanel {

public final double radius = 300;

public final int minrad = 30;

public final Kruskal kruskal = new Kruskal();

public final ArrayList<Double> weights = new ArrayList();

public final ArrayList<Point> lines = new ArrayList();

public final ArrayList<Point> vertices = new ArrayList();

@Override

public void paint(Graphics g) {

g.setColor(Color.BLACK);

g.fillRect(0, 0, getWidth(), getHeight());

g.setColor(Color.WHITE);

for (Point e: lines)

g.drawLine( vertices.get(e.x).x, vertices.get(e.x).y,

vertices.get(e.y).x, vertices.get(e.y).y);

g.setColor(Color.GREEN);

for (int i = 0; i < kruskal.currentStep; i++) {

AlgorithmSteps step = kruskal.algorithmSteps.get(i);

if (step.isTrueEdge) g.setColor(Color.GREEN);

else g.setColor(Color.RED);

g.drawLine( vertices.get(step.graphEdge.firstVertex).x, vertices.get(step.graphEdge.firstVertex).y,

vertices.get(step.graphEdge.secondVertex).x, vertices.get(step.graphEdge.secondVertex).y);

}

g.setColor(Color.BLUE);

for (Point v: vertices)

g.fillOval(v.x - minrad / 2, v.y - minrad / 2, minrad, minrad);

g.setColor(Color.MAGENTA);

for (int i = 0; i < vertices.size(); i++)

g.drawString(i + "", vertices.get(i).x - minrad, vertices.get(i).y - minrad);

g.setColor(Color.YELLOW);

for (int i = 0; i < lines.size(); i++) {

int v1 = (vertices.get(lines.get(i).x).x + vertices.get(lines.get(i).y).x) / 2;

int v2 = (vertices.get(lines.get(i).x).y + vertices.get(lines.get(i).y).y) / 2;

g.drawString(weights.get(i) + "", v1, v2);

}

}

public void actionLoad(ActionEvent e) {

try {

kruskal.graph.graphFromFile();

} catch (Exception exception) {

JOptionPane.showMessageDialog(this, "Graph is empty");

}

}

public void actionPlay(ActionEvent e) {

int size = kruskal.graph.graphVertices.size();

for (int i = 0; i < size; i++) {

double angle = 2 \* Math.PI \* ((double)i) / ((double)size);

vertices.add(new Point((int)(radius \* Math.cos( angle ) + getWidth() / 2),

(int)(radius \* Math.sin( angle ) + getHeight() / 2)));

}

for (GraphEdges edges: kruskal.graph.graphEdges) {

lines.add(new Point(edges.firstVertex, edges.secondVertex));

weights.add(edges.weight);

}

kruskal.run();

repaint();

}

public void actionNext(ActionEvent e) {

kruskal.stepForward();

repaint();

}

public void actionPrev(ActionEvent e) {

kruskal.stepBack();

repaint();

}

}

Файл Kruskal.java:

package com.github.elunmuks.BTS;

import java.util.ArrayList;

import java.util.Comparator;

// class of the main algorithm of the program

public class Kruskal implements Runnable {

public Graph graph;

public ArrayList<AlgorithmSteps> algorithmSteps;

public int currentStep;

public Kruskal() {

graph = new Graph();

algorithmSteps = new ArrayList<>();

}

// sorting the vertices of the graph by their weight

protected void sortEdges() { graph.graphEdges.sort(Comparator.comparingDouble((o1) -> (o1.weight))); }

private int getGraphConnectedVerticesCount(boolean[] isVisited, int vertexNumber, Graph graph1) {

int result = 0;

for (GraphEdges graphEdges : graph1.graphVertices.get(vertexNumber).vertexEdges) {

int number = graphEdges.getSecondVertexNumber(vertexNumber);

if (!isVisited[number]) {

isVisited[number] = true;

result += getGraphConnectedVerticesCount(isVisited, number, graph1);

}

}

return result > 0 ? (result + 1) : 1;

}

protected boolean isGraphConnected(Graph graph1) {

boolean[] isVisited = new boolean[graph1.graphVertices.size()];

if (isVisited.length == 0)

return true;

isVisited[0] = true;

return getGraphConnectedVerticesCount(isVisited, 0, graph1) == graph1.graphVertices.size();

}

private boolean isCyclicGraphRec(boolean[] isVisited, int vertexNumber, int previousVertexNumber, Graph graph1) {

for (GraphEdges graphEdges : graph1.graphVertices.get(vertexNumber).vertexEdges) {

int number = graphEdges.getSecondVertexNumber(vertexNumber);

if (number != previousVertexNumber) {

if (!isVisited[number]) {

isVisited[number] = true;

if (isCyclicGraphRec(isVisited, number, vertexNumber, graph1)) return true;

} else {

return true;

}

}

}

return false;

}

protected boolean isCyclicGraph(Graph graph1) {

boolean[] isVisited = new boolean[graph1.graphVertices.size()];

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

if (!isVisited[i]) {

isVisited[i] = true;

if (isCyclicGraphRec(isVisited, i, -1, graph1)) return true;

}

}

return false;

}

@Override

public void run() {

Graph spanningTree = new Graph();

if (graph.graphVertices.size() == 0)

throw new RuntimeException("Graph is empty.");

if (graph.graphVertices.size() == 1)

return;

sortEdges();

if (!isGraphConnected(graph))

throw new RuntimeException("Graph is not connected.");

for (int i = 0; i < graph.graphVertices.size(); i++)

spanningTree.addVertex(new GraphVertices());

while (currentStep < graph.graphEdges.size()) {

spanningTree.addEdges(graph.graphEdges.get(currentStep));

if (isCyclicGraph(spanningTree)) {

algorithmSteps.add(new AlgorithmSteps(graph.graphEdges.get(currentStep), false));

spanningTree.delEdges(graph.graphEdges.get(currentStep));

} else {

algorithmSteps.add(new AlgorithmSteps(graph.graphEdges.get(currentStep), true));

}

currentStep++;

if (isGraphConnected(spanningTree)) {

currentStep = 0;

graph = spanningTree;

return;

}

}

currentStep = 0;

graph = spanningTree;

}

public void stepForward() {

if (currentStep < algorithmSteps.size())

currentStep++;

}

public void stepBack() {

if (currentStep > 0)

currentStep--;

}

}

Файл Main.java:

package com.github.elunmuks.BTS;

import java.io.IOException;

import java.net.URISyntaxException;

public class Main {

public static void main(String[] args) throws IOException, URISyntaxException {

View view = new View();

}

}

Файл View.java:

package com.github.elunmuks.BTS;

import javax.swing.\*;

import javax.swing.plaf.basic.BasicSplitPaneUI;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class View extends JFrame {

public JButton buttonLoad = new JButton("load graph");

public JButton buttonPlay = new JButton("run algorithm");

public JButton buttonNext = new JButton("next step");

public JButton buttonPrev = new JButton("prev step");

public GraphViewer graphViewer = new GraphViewer();

public View() throws HeadlessException {

JPanel buttons = new JPanel();

buttons.setBackground(Color.BLACK);

buttons.setLayout(new BoxLayout(buttons, BoxLayout.X\_AXIS));

buttons.add(buttonLoad);

buttons.add(buttonPlay);

buttons.add(buttonNext);

buttons.add(buttonPrev);

JPanel main = new JPanel();

main.setBackground(Color.BLACK);

main.setLayout(new BoxLayout(main, BoxLayout.Y\_AXIS));

main.add(buttons);

main.add(graphViewer);

buttonLoad.addActionListener( graphViewer::actionLoad );

buttonPlay.addActionListener( graphViewer::actionPlay );

buttonNext.addActionListener( graphViewer::actionNext );

buttonPrev.addActionListener( graphViewer::actionPrev );

JFrame frame = new JFrame("Kruskal");

frame.setContentPane(main);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(1000, 800);

frame.setResizable(false);

frame.setVisible(true);

}

}