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

ОТЧЕТ

по учебной практике

Тема: Разработка визуализатора алгоритма Борувки

Студент _____ Зуев Д.В.

Студентка _____ Маркова А.В.

Студент _____ Кирсанов А.Я.

Руководитель _____ Жангиров Т.Р.

Санкт-Петербург
2019

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Зуев Д.В. группы 7383

Студентка Маркова А.В. группы 7383

Студент Кирсанов А.Я. группы 7383

Тема практики: Разработка визуализатора алгоритма

Задание на практику:

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

Алгоритм: Алгоритм Борувки.

Сроки прохождения практики: 01.07.2019 – 14.07.2019

Дата сдачи отчета: 12.07.2019

Дата защиты отчета: 12.07.2019

Студент	_____	Зуев Д.В.
Студентка	_____	Маркова А.В.
Студент	_____	Кирсанов А.Я.
Руководитель	_____	Жангиров Т.Р.

АННОТАЦИЯ

Целью данной учебной практики является освоение навыков разработки графического пользовательского интерфейса, разработки алгоритма Борувки построения минимального остовного дерева графа и реализации связи между алгоритмом и интерфейсом на языке программирования Java. Для выполнения задания внутри команды происходит распределение ролей. Каждый член команды выполняет свое задание, потом разработчик ответственный за реализацию связи между графическим интерфейсом и алгоритмом собирает проект полностью.

SUMMARY

The purpose of this education practice is to master the skills of developing a graphical user interface, the development of Borvuka's algorithm for building a minimum spanning tree for the graph and the implementation of the connection between the algorithm and the interface in the Java programming language. To perform the task is the distribution of roles within the team. Each team member performs his task, then the developer responsible for the implementation of the connection between the graphical interface and the algorithm builds the project completely.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной учебной практики является освоение навыков разработки графического пользовательского интерфейса, разработки алгоритма Борувки, построения минимального остовного дерева графа и реализации связи между алгоритмом и интерфейсом на языке программирования Java.

Задача состоит в написании программы, состоящей из классов реализующих граф, алгоритма, графического интерфейса и взаимодействия между ними.

Алгоритм Борувки (Алгоритм Борувки-Соллина) – это алгоритм нахождения минимального остовного дерева в графе.

Работа алгоритма состоит из нескольких итераций, каждая из которых состоит в последовательном добавлении рёбер к остовному лесу графа, до тех пор, пока лес не превратится в дерево, то есть, лес, состоящий из одной компоненты связности.

Псевдокод алгоритма можно описать так:

1. Изначально T – пустое множество рёбер (представляющее собой остовный лес, в котором каждая вершина входит в качестве отдельного дерева).
2. Пока T не является деревом (что эквивалентно условию: пока число рёбер в T меньше, чем $V-1$, где V – число вершин в графе):
   - Для каждой компоненты связности (то есть, дерева в остовном лесе) в подграфе с рёбрами T, найдем самое дешёвое ребро, связывающее эту компоненту с некоторой другой компонентой связности. (Предполагается, что веса рёбер различны, или дополнительно упорядочены так, чтобы всегда можно было найти единственное ребро с минимальным весом).
   - Добавим все найденные рёбра в множество T.
3. Полученное множество T является минимальным остовным деревом входного графа.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные требования к программе

1.1.1 Требования к вводу исходных данных

Граф должен задаваться как набор ребер. Его можно вводить как в редакторе пользовательского интерфейса, так и считывать из файла. Ребро состоит из двух вершин целочисленного типа и веса вещественного типа. Новое ребро должно вводиться с новой строки.

1.1.2 Требования к визуализации

Графический интерфейс должен давать пользователю возможность вводить граф в редакторе или считывать граф из файла. Также с помощью интерфейса пользователь должен иметь возможность пошагово проходить Алгоритм Борувки, возвращаться к предыдущему шагу выполнения алгоритма и сохранять результат работы алгоритма в файл.

На рис.1 представлен эскиз главного окна прототипа, открывающееся при запуске программы.

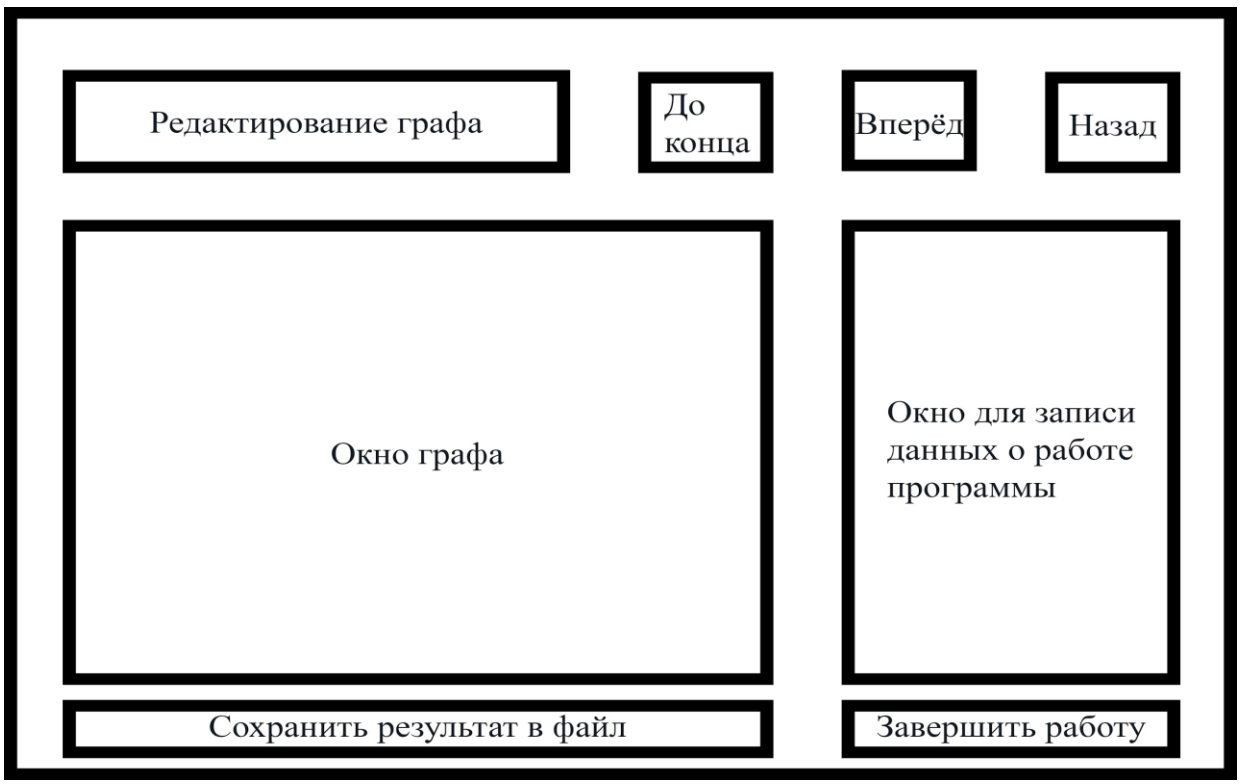

Рисунок 1 – Главное окно

Главное окно содержит кнопку для открытия окна редактирования графа, кнопки управления работой алгоритма (шаг назад, шаг вперед, выполнить до конца), кнопку для сохранения результата в файл, кнопку для завершения работы (закрытия окна), окно, в котором отображается графическое представление графа и окно логов.

Кнопки шаг вперед и выполнить до конца инициализируют выполнение алгоритма. Пока алгоритм не инициализирован кнопки шаг вперед, шаг назад нажать нельзя. При пошаговом выполнении алгоритма редактирование графа будет запрещено. Сохранение результата будет доступно при полном выполнении алгоритма на конкретном графе.

В окне графа будет отображаться введенный граф до начала работы алгоритма, во время работы алгоритма в окне будут отображаться вершины и ребра графа, которые добавлены в остовное дерево на данном шаге. Ребра и вершины, не добавленные на данном шаге, будут более тусклого цвета, чем добавленные. То же самое будет после окончания работы алгоритма. Это сделано для того чтобы не создавать лишнее окно для хранения начального графа.

В окне логов (окно для записи данных о работе программы) будет выводиться информация о пошаговой работе алгоритма (ребра, добавленные на текущем шаге), о создании и редактировании графа, о сохранении результата в файл, о успешной работе алгоритма.

Кнопка «Редактирование графа» открывает окно редактирования графа. Эскиз окна представлен на рис. 2.

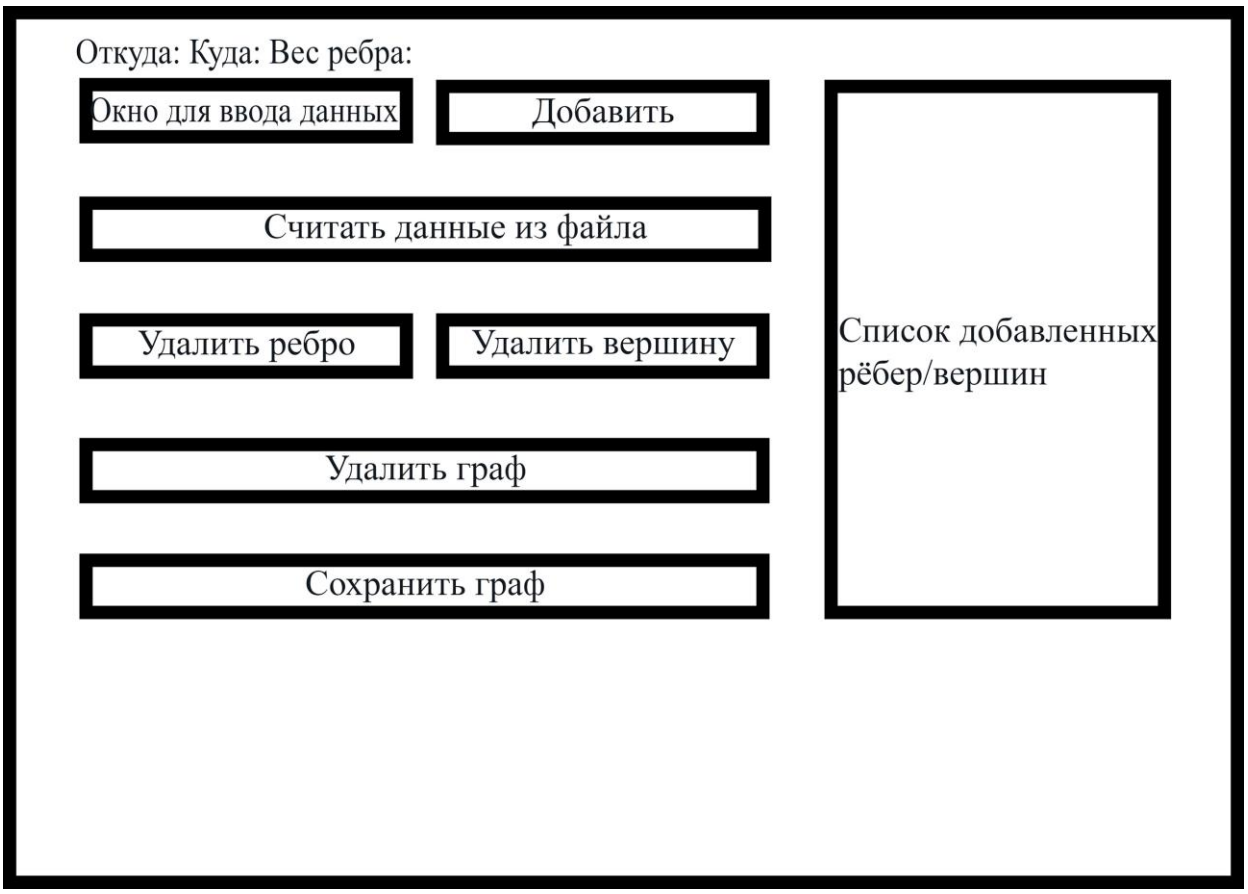

Рисунок 2 – Окно редактирования графа

В поле ввода вводится информация необходимая для редактирования файла. При вводе двух вершин и веса и нажатии кнопки “добавить” в граф добавляется ребро, это изменение отображается в списке ребер. При нажатии кнопки “удалить ребро” или кнопки “удалить вершину” появляется окно, где пользователю предлагается выбрать из списка существующих ребер или вершин то, что нужно удалить. При нажатии кнопки “удалить граф”, граф введенный пользователем удаляется полностью. При нажатии кнопки “считать данные из файла”, анализируется путь до файла, записанный в окне ввода и при существовании файла по данному пути и соблюдении в нем формата ввода графа этот граф считывается. При нажатии кнопки “сохранить граф” граф, введенный пользователем или полученный изменением графа, введенного ранее, сохраняется, и окно редактирования закрывается.

На рис.3,4,5 представлены UML диаграммы классов, реализующих графический интерфейс, работу алгоритма Борувки и взаимодействие пользователя с программой соответственно.

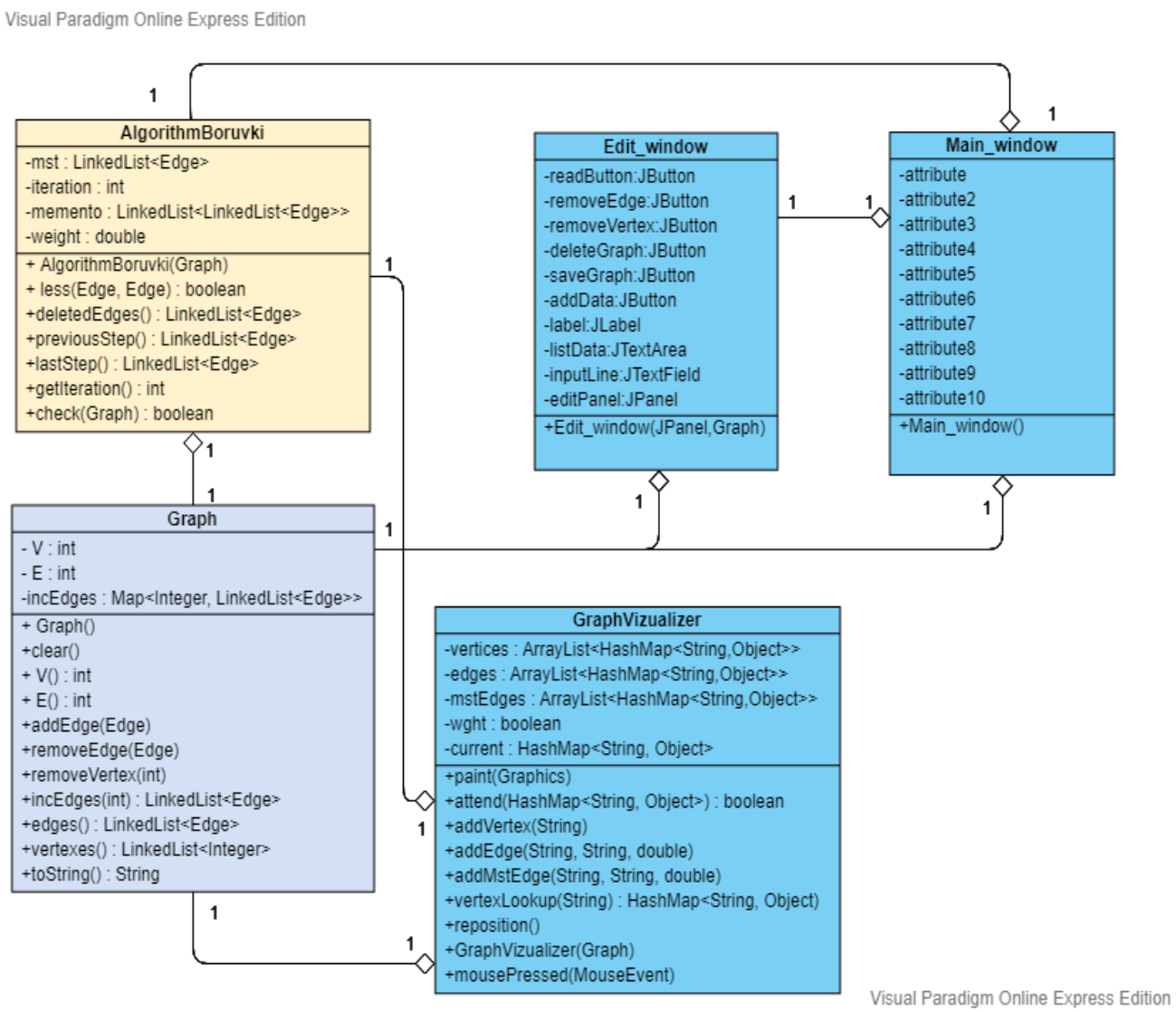

Рисунок 3 – UML диаграмма графического интерфейса

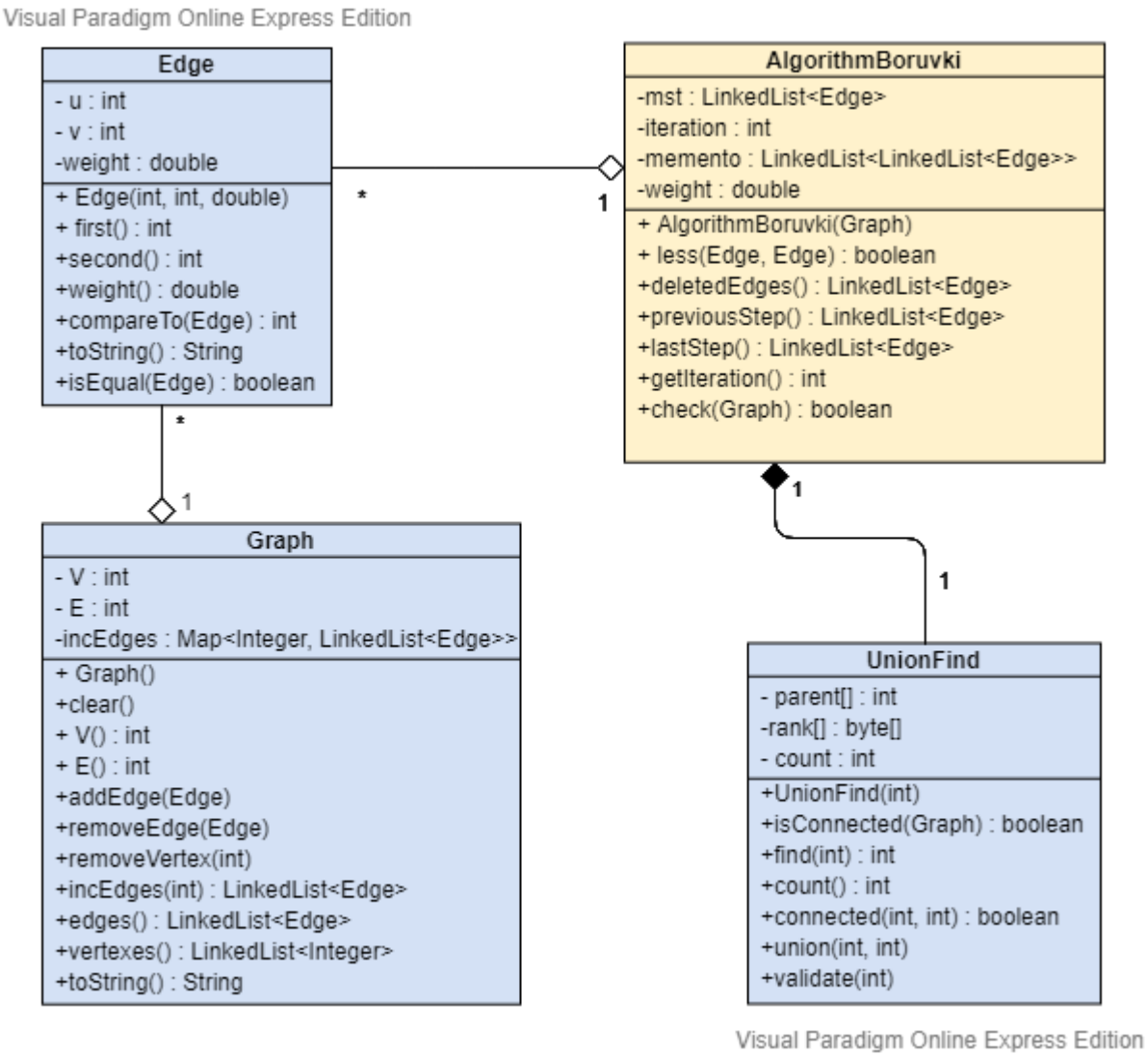

Рисунок 4 – UML диаграмма алгоритма Борувки

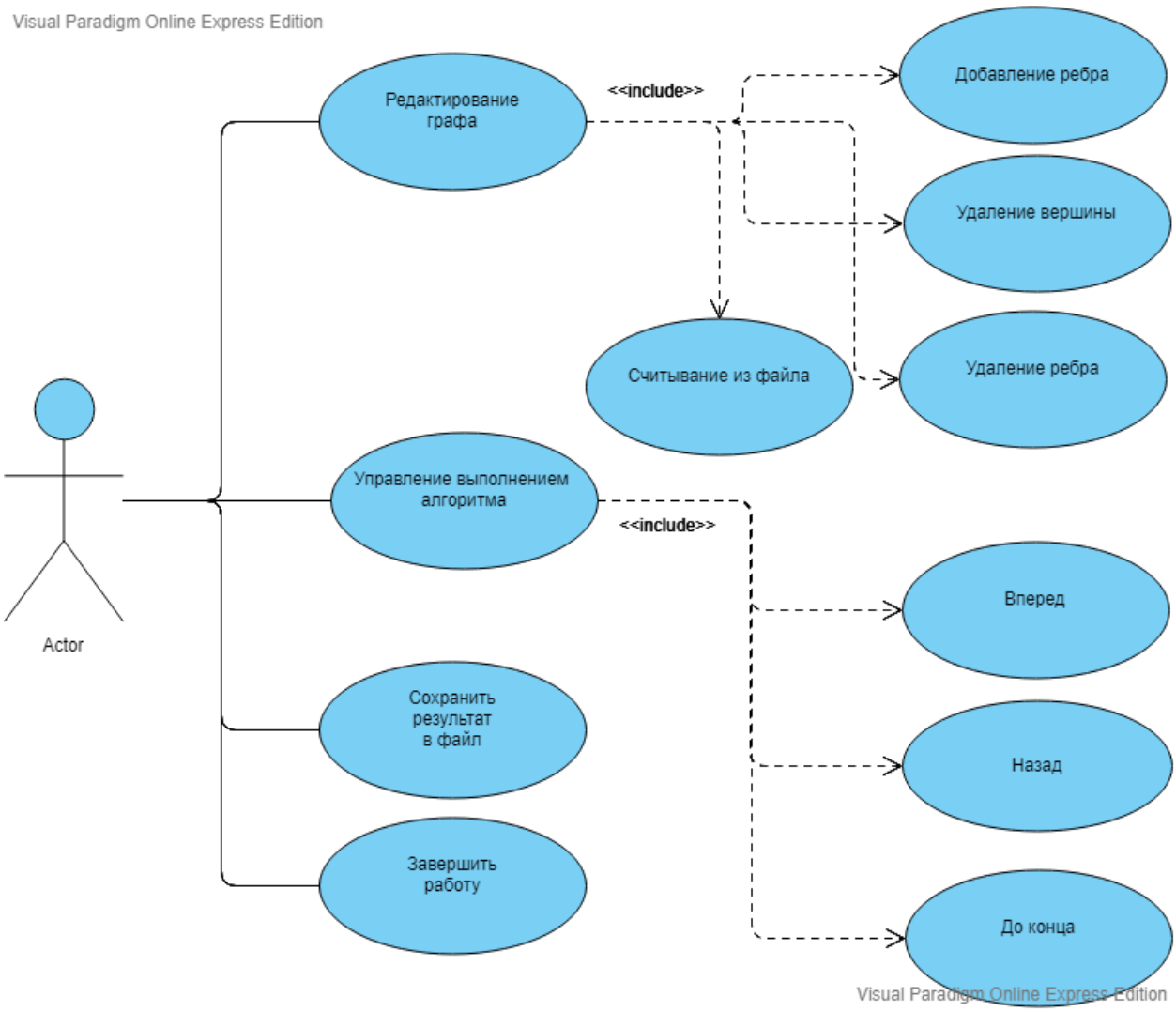

Рисунок 5 – UML диаграмма взаимодействия пользователя с программой

1.2 Уточнение требований после сдачи первого 1-ой версии

После сдачи первой версии программы 08.07.2019 были получены следующие требования: разделить окно ввода на три, чтобы сделать интерфейс интуитивно понятным для пользователя. Добавить возможность сохранения графа в документ.

1.3 Уточнение требований после сдачи второго 2-ой версии

После сдачи второй версии программы 10.07.2019 были получены следующие требования: вызвать стандартное окно проводника при сохранении графа и результатов в файл и при считывании из файла. Добавить таймер, инициирующий выполнение следующего шага алгоритма каждые

полсекунды. Просчитывать шаги по ходу выполнения алгоритма. Организовать хранение данных вне GUI графа. Убрать дублирование кода.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЯ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

01.07.2019 – 04.07.2019 – определение требований к программе; разработка спецификации и согласование её с руководителем.

04.07.2019 – 06.07.2019 – реализация графического пользовательского интерфейса (GUI) без функционала; создание UML диаграмм.

06.07.2019 – 08.07.2019 – разработка класса, реализующего стандартные методы работы с графом; добавление считывания данных с клавиатуры; реализация удаления и добавления вершин и рёбер графа.

08.07.2019 – разработка графической части визуализации графа; добавление кнопки сохранения введенных данных с клавиатуры; исправление работы функции удаления ребер/вершин графа; добавление возможности считывания данных из файла.

09.07.2019 – 10.07.2019 – разработка алгоритма Борувки; разделение поля ввода данных на три окна, чтобы интерфейс был интуитивно понятен пользователю; добавление возможности сохранения графа в документ; добавление вывода сообщений, которые печатают информацию в окно логов о работе программы; реализация возможности запуска алгоритма по шагам итерации, как вперёд, так и назад; добавление визуализации работы алгоритма.

10.07.2019 – 12.07.2019 – добавление вызова стандартного окна проводника при сохранении графа и результатов в файл, а также при считывании из файла; добавление таймера, инициирующего выполнение следующего шага алгоритма каждые полсекунды; доработка просчитывания шагов по ходу выполнения алгоритма; организация хранение данных вне GUI графа; избавление от дублирования кода; добавление кнопки в окно редактирования графа для возможности интуитивного возвращения в главное окно; разработка тестирования; написание отчёта.

2.2. Распределение ролей в бригаде

Зуев Даниил – ответственный за архитектуру проекта, реализация связи между алгоритмом и интерфейсом.

Маркова Ангелина – реализация графического интерфейса, визуализация.

Кирсанов Артём – разработка и реализация алгоритма и его последующее тестирование.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Использованные структуры данных

`public class Main` – главный класс, который обеспечивает работу всей программы, так как содержит отправную точку выполнения.

Поля:

- `public static Graph graph` – граф.
- `public static AlgorithmBoruvki algorithm` – экземпляр класса алгоритма Борувки.

`public class Edge implements Comparable<Edge>` – класс, хранящий ребро графа.

Поля:

- `private final int u` – начальная вершина.
- `private final int v` – конечная вершина.
- `private final double weight` – вес ребра.

`public class Graph` – класс, хранящий граф как структуру вершин и инцидентных им ребер.

Поля:

- `private int V` – количество вершин.
- `private int E` – количество ребер.
- `private Map<Integerm, LinkedList<Edge>> incEdges` – список вершин и инцидентных им ребер.

`public class UnionFind` – класс, поддерживающий методы Union - связывание двух компонент связности графа и Find - поиск корня компоненты связности.

Поля:

- `private int[] parent` – массив вершин и соответствующих им компонент связности.
- `private byte[] rank` – массив, хранящий количество добавленных вершин к компонентам связности.

- `private int count` – количество компонент связности.

`public class AlgorithmBoruvki` – класс, реализующий алгоритм Борувки. Вычисление очередного состояния минимального остовного дерева выполняется методом `step`. При каждом вызове метода создается массив ближайших ребер. Затем для каждого ребра из списка ребер графа вычисляются корни начальной и конечной вершин ребра. По вычисленному индексу корня в массив `closest` заносится ребро, если соответствующее поле в массиве пустое, или там находится ребро с большим весом. После заполнения массива `closest`, перебираются все вершины графа. Для каждой вершины из `closest` берется соответствующее ей ребро с минимальным весом, выбранное на предыдущем шаге. Для выбранного ребра производится проверка, принадлежат ли его вершины одной компоненте связности. Если не принадлежат, ребро добавляется в минимальное остовное дерево и производится связывание двух компонент связности с помощью метода `union`, в который передаются начальная и конечная вершины выбранного ребра.

Поля:

- `private LinkedList<Edge> mst` – список ребер минимального остовного дерева.
- `private int iteration` – номер итерации.
- `private LinkedList<LinkedList<Edge>> memento` – список остовных деревьев для каждого шага алгоритма.
- `private Graph graph` – граф.
- `private LinkedList<Integer> listOfVertexes` – список вершин.
- `private UnionFind uf` – компоненты связности графа.

`public class Main_window` – класс, реализующий графический пользовательский интерфейс (GUI), а также отвечающий за работу главного окна программы. Создается при запуске программы в методе main класса Main,

получая на вход ссылку на объект графа и алгоритма. В конструкторе класса содержатся действия по настройке окна – установке цвета элементов, их положения на окне, действий, если таковые необходимы и различных дополнительных опций. При попытке пользователя выполнить недопустимое действие выводится соответствующее сообщение. Например, сохранение результата, без его фактического получения, выполнением алгоритма Борувки до конца. Так же защита от недопустимых действий состоит в определенном поведении кнопок движения по алгоритму. Например, кнопка «Назад» будет недоступна, пока алгоритм не совершит первый шаг, кнопка «Вперед» станет недоступна при полном выполнении алгоритма. Также кнопки движения по алгоритму не выполняют никаких действий при отсутствии алгоритма.

Для демонстрации работы алгоритма по итерациям есть запуск алгоритма по таймеру, с переходом на следующий шаг через определенное время. Для остановки движения по таймеру есть кнопка «Пауза». При движении по таймеру кнопки пошагового выполнения недоступны.

Пользователю запрещено менять размер окон, он задается автоматически программой, основываясь на разрешении конкретного монитора.

Поля:

- `private JButton editButton` – кнопка, вызывающая окно редактирования графа.
- `private JButton moveForward` – кнопка, запускающая один шаг итерации алгоритма Борувки вперёд.
- `private JButton moveBackward` – кнопка, делающая один шаг итерации алгоритма Борувки назад.
- `private JButton moveToEnd` – кнопка, запускающая алгоритм Борувки до конца.
- `private JButton saveButton` – кнопка, которая вызывает диалоговое окно с выбором файла, в который сохранится результат работы алгоритма.

- `private JButton exitButton` – кнопка, которая завершает работу всей программы.
- `private JButton start` – кнопка, отвечающая за работу алгоритма Борувки по таймеру, между шагами итерации проходит несколько миллисекунд.
- `private JButton stop` – кнопка, позволяющая остановиться на определенной итерации, если алгоритм был запущен по таймеру.
- `private JTextArea logging` – текстовое окно, в котором выводится информация о работе алгоритма.
- `private JPanel rootPanel` – основное рабочее окно, на котором расположены кнопки и поля.
- `private GraphVizualizer graphPanel` – окно для построения графа.
- `private Timer timer` – таймер.
- `public static final double WIDTH` – константа, отвечающая за ширину главного окна.
- `public static final double HEIGHT` – константа, отвечающая за высоту главного окна.
- `public static double coeff` – коэффициент, служащий для корректного масштабирования на разных мониторах.

`public class Edit_window` – класс GUI, отвечающий за работу окна редактирования класса.

При нажатии пользователем кнопки “Редактирования графа” в главном окне, появляется окно редактирования графа, в котором предлагается ввести данные для инициализации графа с клавиатуры или считать их из файла. При некорректном вводе, программа сообщит об ошибке и укажет, в каком именно текстовом окне она была допущена. При введении в графу “Путь до файла” строку пути до файла, из которого нужно произвести считывание, программа производит поиск по файловой системе и если такой файл найден, то

начинается считывание, а также проверка на корректность данных, если же такого файла не существует, то будет выведена ошибка. Если пользователь не ввёл путь до файла, но при этом нажал на кнопку “Считать данные из файла”, то вызывается диалоговое окно JFileChooser – контейнер, в котором расположены несколько компонентов, списков и кнопок, “управляющих” выбором файлов. После того как данные добавлены, информация об этом выводится в окне логов, а также считанные вершины и рёбра появляются в соответствующих выпадающих списках, где можно их впоследствии выбрать и удалить из графа. Кнопка “Удалить граф” позволяет полностью очистить данные о существующем графе. Введённые данные с клавиатуры можно сохранить в файл, чтобы в следующий раз просто считать их с файла, а не печатать вручную. После окончания редактирования графа пользователь может вернуться к главному окну, нажав на “Крестик” или кнопку “Вернуться на главное окно”, после этих действий программа уточнит намерения о выходе из текущего окна. Как только выполнено вышеописанное действие, запускается метод `setGraph`, который отвечает за отрисовку созданного графа. Когда пользователь оказывается на главном окне, то видит граф, данные которого он задал, при необходимости, можно вернуться к окну редактирования и что-либо изменить.

Для удобства пользователя при вводе данных с клавиатуры можно нажать Enter, что приведет к автоматическому переходу на следующее текстовое поле, при нажатии в последнем окне, введённые данные добавляются, если они корректны.

Поля:

- `private JButton readButton` – кнопка, позволяющая считывать данные из файла, который находится по пути, введенном пользователем, а если строка `Path` пустая, то вызывается диалоговое окно, в котором можно выбрать нужный файл.
- `private JButton removeEdge` – кнопка для удаления ребра графа, а также вершин, которые являются одиночной компонентой связности.

- `private JButton removeVertex` – кнопка для удаления вершины и инцидентных ей рёбер.
- `private JButton deleteGraph` – кнопка для удаления текущего графа.
- `private JButton saveGraph` – кнопка, позволяющая сохранить введенный пользователем граф с клавиатуры, для этого вызывается диалоговое окно с выбором файла, в который и вносятся данные графа.
- `private JButton addData` – кнопка, добавляющая новые вершины и рёбра между ними.
- `private JButton returnMainWindow` – кнопка для возвращения в главное окно программы.
- `private JComboBox edgeList` – выпадающий список рёбер, которые можно удалить.
- `private JComboBox vertexList` – выпадающий список вершин, которые можно удалить.
- `private JLabel labelFrom` – надпись “Откуда” над текстовым полем `inputLineFrom`.
- `private JLabel labelTo` – надпись “Куда” над текстовым полем `inputLineTo`.
- `private JLabel labelWeight` – надпись “Вес ребра” над текстовым полем `inputLineWeight`.
- `private JLabel labelPath` – надпись “Путь до файла” рядом с текстовым окном `inputLinePath`.
- `private JTextArea listData` – текстовое поле для вывода информации о добавленных рёбрах и вершинах, а также об их удалении.
- `private JTextField inputLineFrom` – текстовое поля для ввода первой вершины.
- `private JTextField inputLineTo` – текстовое поле для ввода второй вершины.

- `private JTextField inputLineWeight` – текстовое поле для ввода веса ребра между первой и второй вершинами.
- `private JTextField inputLinePath` – текстовое поле для ввода пути до файла, в которой хотим сохранить введенный граф.
- `private JPanel editPanel` – окно редактирования графа, на котором расположены кнопки и поля.
- `private int from` – переменная, содержащая первую вершину.
- `private int to` – переменная, содержащая вторую вершину.
- `private double weight` – переменная, содержащая вес ребра.

`public class GraphVizualizer` – класс, конвертирующий абстрактный граф в его плоскостное представление, служит для визуализации. Создается в конструкторе класса окна редактирования как действие на выход из окна редактирования. При создании получает на вход граф, в зависимости от количества вершин в графе задает для каждой вершины координаты как вершины правильного многоугольника. Метод `paint` располагает объекты графа на окне в зависимости от их координат, полученных методом `arrange`. Сначала рисуются ребра, проверяя принадлежность их к списку ребер добавленных в минимальное остовное дерево для установки цвета ребра. Затем рисуются вершины как эллипсы. Для добавления ребер минимального остовного дерева используется метод `setMSTEdges`. Так же в классе установлены действия на нажатие и перемещение компьютерной мыши. При нажатии определяется положение курсора и сравнивается с положениями вершин. При совпадении вершину можно будет перемещать. При перемежении вершине устанавливаются координаты курсора и граф перерисовывается.

Поля:

- `private LinkedList<Edge> listOfEdges` – список рёбер исходного графа.

- `private LinkedList<Edge> listOfEdgesOfMSTStep` – список рёбер минимального остовного дерева.
- `private LinkedList<Integer> listOfVerteces` – список вершин исходного графа.
- `private ArrayList<Coordinates> coordsOfVerteces` – массив координат вершин графа.
- `public Integer indexOfSelectedVertex` – индекс выбранной вершины графа.

`public class Coordinates` – вспомогательный класс для визуализации графа, хранящий координаты.

Поля:

- `public int x` – координата абсциссы.
- `public int y` – координата ординаты.

3.2. Основные методы

Метод класса `Main`:

- `public static void main(Strings[] args)` – метод, который исполняет корректный запуск программы. Это место с которого начинается выполнение, отправная точка выполнения.

Методы класса `Edge`:

- `public Edge(int, int, double)` – конструктор класса, инициализирующий поля.
- `public double weight()` – метод, возвращающий вес ребра.
- `public int first()` – метод, возвращающий исходную вершину ребра.
- `public int second(int)` – метод, принимающий исходную вершину и возвращающий конечную вершину ребра.
- `public int compareTo(Edge)` – метод, выполняющий сравнение вершин по весам.
- `public String toString()` – перевод вершины в строку.

- `public boolean isEqual(Edge)` – класс, проверяющий совпадение начальной и конечной вершин.

Методы класса `Graph`:

- `public Graph()` – конструктор класса.
- `public void clear()` – приводит граф к начальному состоянию.
- `public void addEdge(Edge)` – добавляет ребро в список инцидентности начальной и конечной вершин.
- `public void removeEdge(Edge)` – удаляет ребро из списка инцидентности начальной и конечной вершин.
- `public void removeVertex(int)` – удаляет вершину из графа. Если вершина обособленная, удаляет все инцидентные ей ребра.
- `public LinkedList<Edge> incEdges(int)` – возвращает список инцидентных ребер для переданной вершины.
- `public LinkedList<Edge> edges()` – возвращает список ребер графа.
- `public LinkedList<Integer> vertexes()` – возвращает список вершин графа.
- `public String toString()` – перевод графа в строку.

Методы класса `UnionFind`:

- `public UnionFind(int)` – конструктор класса. Создает массив, где каждой вершине ставится в соответствие своя компонента связности.
- `public boolean isConnected(Graph)` – метод, проверяющий, что граф имеет одну компоненту связности.
- `public int find(int)` – возвращает корень компоненты связности.
- `public int count()` – возвращает количество компонент связности.
- `public boolean connected(int, int)` – проверяет принадлежность двух вершин одной компоненте связности.

- `public void union(int, int)` – соединяет две компоненты связности, если переданные вершины лежат в разных компонентах связности.
- `public void validate(int)` – проверяет, лежит ли индекс вершины в диапазоне массива созданных вершин.

Методы класса `AlgorithmBoruvki`:

- `public AlgorithmBoruvki(Graph)` – конструктор класса. Инициализирует поля класса.
- `private void step()` – выполняет очередной шаг итерации алгоритма Борувки и записывает получившееся остовное дерево в список остовных деревьев.
- `private static boolean less(Edge, Edge)` – выполняет сравнение вершин по их весу.
- `public LinkedList<Edge> nextStep()` – выполняет проверку возможности сделать очередной шаг алгоритма и вызывает step(). Если очередной шаг сделать невозможно, возвращает остовное дерево, получившееся на последней итерации.
- `public LinkedList<Edge> deletedEdges()` – возвращает список составляющих разницу текущего и предыдущего шагов выполнения алгоритма.
- `public LinkedList<Edge> previousStep()` – возвращает остовное дерево предыдущего шага.
- `public LinkedList<Edge> lastStep()` – возвращает остовное дерево последнего шага.
- `public int getIteration()` – возвращает номер текущей итерации.

Методы класса `Main_window`:

- `public Main_window` – конструктор класса, инициализирующий кнопки и поля главного окна GUI. Задает их расположение, цвет, формат шрифта надписей, рамки, а также действия при нажатии.
- `private void closeAction()` – метод, задающий действия для кнопок, которые закрывают программу (“Крестик” и “Завершить работу”).
- `private boolean saveAction()` – метод, задающий действия для кнопок, которые сохраняют результат работы алгоритма.

Методы класса `Edit_window`:

- `public Edit_window(Jpanel, Graph, GraphVizualizer)` – конструктор класса, инициализирующий кнопки и поля окна редактирования графа. Задает их расположение, цвет, формат шрифта надписей, рамки, а также действия при нажатии.
- `private boolean reloadGraph(ArrayList<String>, Graph, LinkedList<Edge>, LinkedList<Integer>)` – метод, получающий на вход массив строк из файла, проходит по нему и проверяет на корректность входных данных, при успешном считывании записывает в граф, при неудаче выводит сообщение об ошибке.
- `private boolean readFrom()` – метод, реализующий считывание с поля ввода “Откуда”, а также обеспечивающий проверку правильности введенных данных.
- `private boolean readTo()` – метод, реализующий считывание с поля ввода “Куда”, а также обеспечивающий проверку правильности введенных данных.
- `private boolean readWeight()` – метод, реализующий считывание с поля ввода “Вес ребра”, а также обеспечивающий проверку правильности введенных данных.

Методы класса `GraphVizualizer`:

- `public GraphVizualizer()` – конструктор класса, задающий основные параметры окна, служащего для визуализации графа.
- `public void setGraph(Graph)` – метод, получающий на вход абстрактный граф, который нужно конвертировать в плоскостное представление. Сохраняет данные о графе, задаёт расположение вершин и рёбер на плоскости, делает проверку корректности, перерисовывает старый граф.
- `public void paint(Graphics)` – метод для отрисовки графа, который также проверяет наличие рёбер в остовном дереве и если такие существуют, то перекрашивает их в другой цвет для наглядности работы алгоритма Борувки.
- `private void arrangement()` – метод, задающий расположение графа на плоскости.
- `public void setMSTEdges(LinkedList<Edge>)` – метод отрисовки рёбер, которые вошли в остовное дерево по ходу работы алгоритма Борувки.
- `private Integer find(Point2D)` – метод, который ищет вершину по заданной координате курсора на экране.

Методы класса `Coordinates`:

- `public Coordinates(int, int)` – конструктор, принимающий пару чисел, являющихся координатами вершин графа.
- `public Coordinates(Coordinates)` – конструктор, принимающий экземпляр класса, чтобы инициализировать координаты вершин графа.

3.3. Порядок работы:

1. Открывается главное окно, описанное ранее. При отсутствии графа алгоритм не может работать и, соответственно, кнопки, отвечающие за выполнение алгоритма, никаких действий не совершают.
2. Для ввода графа есть окно редактирования графа, которое открывается с помощью кнопки на главном экране. В окне редактирования можно

считывать граф из файла, добавлять и удалять ребра, удалять вершины, удалять сам граф и записывать граф в файл.

3. После ввода графа можно выполнять алгоритм и смотреть по итерационную работу алгоритма с помощью кнопок управления выполнением алгоритма.
4. После выполнения алгоритма можно сохранить результат выполнения в файл.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

Графический интерфейс был протестирован вручную. В ходе тестирования окончательной версии программы ошибок в работе графического интерфейса выявлено не было.

4.2. Тестирование кода алгоритма

Для тестирования алгоритма была подключена библиотека Junit и написаны соответствующие тесты.

При тестировании классов `AlgorithBoruvki`, `Edge` и `UnionFind` ошибок в работе выявлено не было.

При тестировании класса `Graph` была выявлена ошибка работы метода добавления ребра `addEdge`. При добавлении ребра, добавленного ранее, счетчик ребер увеличивается на 1. Для исправления добавлена проверка ребра на существование и при добавлении существующего ребра метод не увеличивает счетчик ребе и перезаписывает ребро.

Выявлена ошибка работы метода удаления ребра `removeEdge`: при удалении несуществующего ребра бросается исключение `NullPointerException`. Для исправления добавлена проверка ребра на существование и при добавлении несуществующего ребра метод завершает свое выполнение. Та же ошибка исправлена для метода `removeVertex` удаления вершины.

Так же работа алгоритма протестирована с помощью графического интерфейса. В ходе тестирования недочетов в работе алгоритма не выявлено.

ЗАКЛЮЧЕНИЕ

В ходе данной учебной практики были изучены основы программирования на языке Java [1], были получены навыки работы с библиотеками визуализации Swing [2] и AWT [3], также навыки программирования графа для работы алгоритма Борувки. Была разработана программа, основой которой является графический интерфейс, взаимодействуя с которым пользователь может увидеть работу алгоритма Борувки по итерациям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Java Базовый курс // Stepik.URL: https://stepik.org/course/187/syllabus
2. Г.Шилдт. Swing руководство для начинающих. М: Вильямс, 2007. 705 с.
3. The Java Tutorials // Oracle.URL: https://docs.oracle.com/javase/tutorial/index.html

ПРИЛОЖЕНИЕ А

Исходный код

Main.java

```
public class Main {
    public static Graph graph = new Graph();
    public static AlgorithmBoruvki algorithm = null;
    public static void main(String[] args) {
        Main_window app = new Main_window();
        app.setVisible(true);
    }
}
```

Main_window.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.LinkedList;
import java.util.TimerTask;
import java.util.Timer;
import javax.swing.*;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.io.*;

public class Main_window extends JFrame{
    private JButton editButton = new JButton("Редактирование графа");
//кнопка
    private JButton moveForward = new JButton("Вперёд");
    private JButton moveBackward = new JButton("Назад");
    private JButton moveToEnd = new JButton("До конца");
    private JButton saveButton = new JButton("Сохранить результат в файл");
    private JButton exitButton = new JButton("Завершить работу");
    private JButton start = new JButton("Запуск по таймеру");
    private JButton stop = new JButton("Пауза");
    private JTextArea logging = new JTextArea(); //вывод логов
    private JPanel rootPanel = new JPanel(); //основное рабочее окно
    private GraphVizualizer graphPanel = new GraphVizualizer(); //окно
построения графа
    private Timer timer = new Timer();


    public static final double WIDTH = 1230;
    public static final double HEIGHT = 925;
    public static double coeff;

    public Main_window () {
        super("Главное окно");
        JButton[] arrayButtons = {editButton, moveForward, moveBackward,
                moveToEnd, saveButton, exitButton, start, stop};
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        coeff =  0.8*screenSize.height / HEIGHT;
        setBounds((int)((screenSize.width - WIDTH*coeff)/2),
                (int)((screenSize.height - HEIGHT*coeff)/2),
```

```
                (int)(WIDTH*coeff),(int)(HEIGHT*coeff)); //размер окна
        setResizable(false); //запрет на изменения размера окна
        rootPanel.setLayout(null);
        add(rootPanel);
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        logging.setEditable(false); //запрет на редактирование, ввод текста
        logging.setLineWrap(true);
        //добавление скроллбара
        final JScrollPane scrollPane = new JScrollPane(logging,
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        Font fontButton = new Font(null, 3, 16); //выбор шрифта
        Font fontLogging = new Font(null, 1, 14);
        for(JButton current : arrayButtons) {
            current.setFont(fontButton);
            //цвет
            current.setBackground(Color.cyan);
            //убрать выделение вокруг текста кнопки
            current.setFocusPainted(false);
            //установка рамок
            current.setBorder(BorderFactory.createLineBorder(Color.gray,2));
        }
        logging.setFont(fontLogging);
        //размеры кнопок

editButton.setBounds((int)(10*coeff),(int)(5*coeff),(int)(410*coeff),(int)(50
*coeff));

moveToEnd.setBounds((int)(460*coeff),(int)(5*coeff),(int)(220*coeff),(int)(50
*coeff));

moveForward.setBounds((int)(720*coeff),(int)(5*coeff),(int)(220*coeff),(int)(
50*coeff));

moveBackward.setBounds((int)(980*coeff),(int)(5*coeff),(int)(220*coeff),(int)
(50*coeff));

saveButton.setBounds((int)(10*coeff),(int)(820*coeff),(int)(670*coeff),(int)(
50*coeff));

exitButton.setBounds((int)(720*coeff),(int)(820*coeff),(int)(480*coeff),(int)
(50*coeff));

scrollPane.setBounds((int)(720*coeff),(int)(125*coeff),(int)(480*coeff),(int)
(685*coeff));

graphPanel.setBounds((int)(10*coeff),(int)(65*coeff),(int)(670*coeff),(int)(7
45*coeff));

start.setBounds((int)(720*coeff),(int)(65*coeff),(int)(220*coeff),(int)(50*co
eff));

stop.setBounds((int)(980*coeff),(int)(65*coeff),(int)(220*coeff),(int)(50*coe
ff));
        //Цвета полей
        rootPanel.setBackground(Color.darkGray);
```

```
        graphPanel.setBackground(Color.gray);
        logging.setBackground(Color.gray);
        logging.setForeground(Color.white);
        //добавление объектов на панель
        rootPanel.add(editButton);
        rootPanel.add(moveForward);
        rootPanel.add(moveBackward);
        rootPanel.add(moveToEnd);
        rootPanel.add(saveButton);
        rootPanel.add(exitButton);
        rootPanel.add(scrollPane);
        rootPanel.add(graphPanel);
        rootPanel.add(start);
        rootPanel.add(stop);
        //устоновка таймера

        //установка неактивной кнопки назад
        stop.setEnabled(false);
        moveBackward.setEnabled(false);
        //добавление действий
        editButton.addActionListener(e->{
            Edit_window editPanel = new Edit_window(rootPanel, Main.graph,
graphPanel);
            editPanel.setVisible(true);
            Main.algorithm = null;
            moveToEnd.setEnabled(true);
            moveBackward.setEnabled(false);
            moveForward.setEnabled(true);
            stop.setEnabled(false);
            start.setEnabled(true);
            logging.setText("");
            timer.cancel();
            timer.purge();
        });
        moveForward.addActionListener(e ->{
            if(Main.graph.V() != 0) {
                if (Main.algorithm == null) {
                    Main.algorithm = new AlgorithmBoruvki(Main.graph);
                }
                LinkedList<Edge> tmp = Main.algorithm.nextStep();
                graphPanel.setMSTEdges(tmp);
                if (tmp.size() == Main.graph.V() - 1) {
                    start.setEnabled(false);
                    moveForward.setEnabled(false);
                    moveToEnd.setEnabled(false);
                }
                moveBackward.setEnabled(true);
                logging.append("Итерация алгоритма Борувки №" +
Main.algorithm.getIteration() + '\n'
                        + "Добавлены следующие ребра:\n");
                for (Edge edge : Main.algorithm.deletedEdges()) {
                    logging.append(edge.toString() + '\n');
                }
                logging.setCaretPosition(logging.getDocument().getLength());
            }
        });
```

```
        moveBackward.addActionListener(e-> {
            logging.append("Откат до итерации алгоритма Борувки №" +
(Main.algorithm.getIteration() - 1) + '\n'
                    + "Удалены следующие ребра:\n");
            for (Edge edge : Main.algorithm.deletedEdges()) {
                logging.append(edge.toString() + '\n');
            }
            logging.setCaretPosition(logging.getDocument().getLength());
            LinkedList<Edge> tmp = Main.algorithm.previousStep();
            graphPanel.setMSTEdges(tmp);
            if (Main.algorithm.getIteration() == 0) {
                moveBackward.setEnabled(false);
            }
            start.setEnabled(true);
            moveForward.setEnabled(true);
            moveToEnd.setEnabled(true);
        });
        moveToEnd.addActionListener(e -> {
            if(Main.graph.V() != 0) {
                if (Main.algorithm == null) {
                    Main.algorithm = new AlgorithmBoruvki(Main.graph);
                }
                LinkedList<Edge> tmp = Main.algorithm.lastStep();
                graphPanel.setMSTEdges(tmp);
                moveForward.setEnabled(false);
                logging.append("Итерация алгоритма Борувки №" +
Main.algorithm.getIteration() + '\n'
                        + "Добавлены следующие ребра:\n");
                for (Edge edge : tmp) {
                    logging.append(edge.toString() + '\n');
                }
                JScrollBar vertical = scrollPane.getVerticalScrollBar();
                vertical.setValue( vertical.getMaximum() );
                start.setEnabled(false);
                moveBackward.setEnabled(true);
                moveToEnd.setEnabled(false);
            }
        });
        saveButton.addActionListener(e -> {
            saveAction();
        });
        exitButton.addActionListener(e -> {
            closeAction();
        });
        start.addActionListener(e -> {
            if(Main.graph.V() != 0) {
                moveForward.setEnabled(false);
                moveBackward.setEnabled(false);
                moveToEnd.setEnabled(false);
                TimerTask timerTask = new TimerTask() {
                    @Override
                    public void run() {
                        if (Main.algorithm == null) {
                            Main.algorithm = new
AlgorithmBoruvki(Main.graph);
                        }
```

```
                        LinkedList<Edge> tmp = Main.algorithm.nextStep();
                        graphPanel.setMSTEdges(tmp);
                        logging.append("Итерация алгоритма Борувки №" +
Main.algorithm.getIteration() + '\n'
                                + "Добавлены следующие ребра:\n");
                        for (Edge edge : Main.algorithm.deletedEdges()) {
                            logging.append(edge.toString() + '\n');
                        }

logging.setCaretPosition(logging.getDocument().getLength());
                        if(tmp.size() == Main.graph.V() - 1) {
                            moveBackward.setEnabled(true);
                            start.setEnabled(false);
                            stop.setEnabled(false);
                            timer.cancel();
                            timer.purge();
                        }
                    }
                };
                timer = new Timer();
                timer.schedule(timerTask, 500, 500);
                stop.setEnabled(true);
            }
        });
        stop.addActionListener(e ->{
            if (Main.algorithm != null && Main.algorithm.edges().size() ==
Main.graph.V() - 1) {
            } else {
                moveForward.setEnabled(true);
                moveToEnd.setEnabled(true);
            }
            moveBackward.setEnabled(true);
            stop.setEnabled(false);
            timer.cancel();
            timer.purge();
        });
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent we) {
                closeAction();
            }
        });
    }
    private void closeAction(){
        timer.cancel();
        timer.purge();
        String title = "Окно подтверждения";
        UIManager.put("OptionPane.yesButtonText", "Да");
        UIManager.put("OptionPane.noButtonText", "Нет");
        UIManager.put("OptionPane.cancelButtonText", "Отмена");
        int check = JOptionPane.showConfirmDialog(null,"Сохранить результат
перед закрытием?", title, JOptionPane.YES_NO_CANCEL_OPTION);
        if (check == JOptionPane.YES_OPTION){
            if(!saveAction()) return;
            dispose();
            System.exit(0);
```

```
        }
        else if (check == JOptionPane.NO_OPTION) {
            dispose();
            System.exit(0);
        }
    }
    private boolean saveAction(){
        if(Main.algorithm == null){
            JOptionPane.showMessageDialog(Main_window.this,
                    "Нечего сохранять!" );
            return false;
        }
        if(Main.algorithm.edges().size() != Main.graph.V() - 1){
            JOptionPane.showMessageDialog(Main_window.this,
                    "Завершите алгоритм!" );
            return false;
        }
        try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch (Exception ex){
            JOptionPane.showMessageDialog(Main_window.this,
                    "Forbidden.");
            return false;
        }
        JFileChooser fd = new JFileChooser(new File(".").getAbsolutePath());
//дилоговое окно
        FileNameExtensionFilter filter = new FileNameExtensionFilter(".txt",
"txt", "text");
        fd.setFileFilter(filter);
        fd.setAcceptAllFileFilterUsed(false);

        int ret = fd.showDialog(null,"Выбор файла");
        if (ret == JFileChooser.APPROVE_OPTION) {
            File graphWrite = fd.getSelectedFile(); //получение выбранного
файла
            File save;
            StringBuilder str = new StringBuilder(graphWrite.getPath());
            String newStr = str.substring(0,
str.lastIndexOf(graphWrite.getName()));
            int index = graphWrite.getName().lastIndexOf(".");
            if(index == -1) {
                save = new File(newStr,
                        graphWrite.getName() + ".txt");
            } else {
                save = new File(newStr,
                        graphWrite.getName().substring(0, index) + ".txt");
            }
            if (graphWrite.exists()) {
                graphWrite.delete();
            }
            try {
                graphWrite.renameTo(save);
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(Main_window.this,
```

```
                        "Не удалось открыть файл!");
                return false;
            }
            try(BufferedWriter  writerResult = new BufferedWriter( new
FileWriter(save)))
            {
                writerResult.write("Для графа:");
                writerResult.newLine();
                for (Edge edge : Main.graph.edges()) {
                    writerResult.write(edge.toString());
                    writerResult.newLine();
                }
                writerResult.write("Построено минимальное остовное дерево с
помошью алгоритма Борувки:");
                writerResult.newLine();
                for (Edge edge : Main.algorithm.edges()) {
                    writerResult.write(edge.toString());
                    writerResult.newLine();
                }
                writerResult.flush();
            }
            catch(IOException ex){
                JOptionPane.showMessageDialog(Main_window.this,
                        "Не удалось записать в файл!" );
                return false;
            }
            try {

UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
            }
            catch (Exception ex){
                JOptionPane.showMessageDialog(Main_window.this,
                        "Forbidden.");
                return false;
            }
            JOptionPane.showMessageDialog(Main_window.this,
                    "Результат успешно сохранен в файл!" );
            return true;
        } else {
            try {

UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
            }
            catch (Exception ex){
                JOptionPane.showMessageDialog(Main_window.this,
                        "Forbidden.");
                return false;
            }
            JOptionPane.showMessageDialog(Main_window.this,
                    "Операция отменена.");
            return false;
        }
    }
}
```

Edit_window.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.JPanel;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Scanner;
import java.io.*;

public class Edit_window extends JFrame{
    private JButton readButton = new JButton("Считать данные из файла");
    JComboBox edgeList;//список ребер доступных для удаления
    JComboBox vertexList;//список вершин доступных для удаления
    private JButton removeEdge = new JButton("Удалить ребро");
    private JButton removeVertex = new JButton("Удалить вершину");
    private JButton deleteGraph = new JButton("Удалить граф");
    private JButton saveGraph = new JButton("Сохранить граф");
    private JButton addData = new JButton("Добавить");
    private JButton returnMainWindow = new JButton("Вернуться на главное
окно");
    private JLabel labelFrom = new JLabel("Откуда:");
    private JLabel labelTo = new JLabel("Куда:");
    private JLabel labelWeight = new JLabel("Вес ребра:");
    private JLabel labelPath = new JLabel("Путь до файла:");
    private JTextArea listData = new JTextArea(); //список добавленных вершин
и рёбер
    private JTextField inputLineFrom = new JTextField("",3);
    private JTextField inputLineTo = new JTextField("",3);
    private JTextField inputLineWeight = new JTextField("",3);
    private JTextField inputLinePath = new JTextField("",3);
    private JPanel editPanel = new JPanel(); //окно редактирования
    private int from = -1;
    private int to = -1;
    private double weight;

    public Edit_window(JPanel parent, Graph graph, GraphVizualizer
graphPanel) {
        super("Окно редактирования графа");
        parent.setVisible(false);
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((int)((screenSize.width -
Main_window.WIDTH*Main_window.coeff)/2),
                (int)((screenSize.height -
Main_window.HEIGHT*Main_window.coeff)/2) ,

(int)(Main_window.WIDTH*Main_window.coeff),(int)(Main_window.HEIGHT*Main_wind
ow.coeff)); //размер окна
        setResizable(false); //запрет на изменения размера окна
        editPanel.setLayout(null);
        add(editPanel);
        //установка действия на нажатие кнопки закрытия окна:
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        listData.setEditable(false); //запрет на редактирование, ввод текста
        listData.setLineWrap(true);
        //добавление скроллбара
```

```
        final JScrollPane scrollPane = new JScrollPane(listData,
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        inputLinePath.setEnabled(true);
        LinkedList<Edge> listOfEdges = graph.edges();//список ребер графа
        LinkedList<Integer> listOfVertexes = graph.vertexes();//список ребер
графа
        //соответствующие наборы строк
        String[] stringListOfEdges = new String[graph.E()];
        if(listOfEdges.size() != 0) {
            listData.append("В графе имеются следующие ребра:\n");
        }
        for (int i = 0; i < graph.E(); i++) {
            stringListOfEdges[i] = listOfEdges.get(i).toString();
            listData.append(listOfEdges.get(i).toString()+"\n");
        }
        String[] stringListOfVertexes = new String[graph.V()];
        for(int i = 0; i < graph.V(); i++){
            stringListOfVertexes[i] = listOfVertexes.get(i).toString();
        }
        JButton[] buttonList = {readButton, saveGraph, removeEdge,
removeVertex, deleteGraph, addData, returnMainWindow};
        edgeList = new JComboBox(stringListOfEdges);
        vertexList = new JComboBox(stringListOfVertexes);
        edgeList.setEditable(false);
        vertexList.setEditable(false);
        //стиль текста
        Font fontButton = new Font(null, 3, 16); //выбор шрифта
        Font fontList = new Font(null, 1, 14);
        for(JButton button : buttonList){
            button.setFont(fontButton);
            button.setBackground(Color.cyan);
            //установка рамок
            button.setBorder(BorderFactory.createLineBorder(Color.gray,2));
            //убрать выделение вокруг текста кнопки
            button.setFocusPainted(false);
            editPanel.add(button);
        }
        JLabel[] labelList = {labelFrom, labelPath, labelTo, labelWeight};
        for(JLabel label : labelList){
            label.setFont(fontButton);
            label.setBackground(Color.gray);
            label.setForeground(Color.white);
            editPanel.add(label);
        }
        JTextField[] textList = {inputLineFrom, inputLinePath, inputLineTo,
inputLineWeight};
        for(JTextField textField : textList){
            textField.setFont(fontList);
            textField.setBackground(Color.gray);
            textField.setForeground(Color.white);
            editPanel.add(textField);
        }
        edgeList.setFont(fontList);
        vertexList.setFont(fontList);
        listData.setFont(fontList);
```

```
        //размеры кнопок

readButton.setBounds((int)(10*Main_window.coeff),(int)(190*Main_window.coeff)
,(int)(740*Main_window.coeff),(int)(50*Main_window.coeff));

edgeList.setBounds((int)(10*Main_window.coeff),(int)(260*Main_window.coeff),(
int)(350*Main_window.coeff),(int)(50*Main_window.coeff));

vertexList.setBounds((int)(400*Main_window.coeff),(int)(260*Main_window.coeff
),(int)(350*Main_window.coeff),(int)(50*Main_window.coeff));

removeEdge.setBounds((int)(10*Main_window.coeff),(int)(330*Main_window.coeff)
,(int)(350*Main_window.coeff),(int)(50*Main_window.coeff));

removeVertex.setBounds((int)(400*Main_window.coeff),(int)(330*Main_window.coe
ff),(int)(350*Main_window.coeff),(int)(50*Main_window.coeff));

deleteGraph.setBounds((int)(10*Main_window.coeff),(int)(400*Main_window.coeff
),(int)(740*Main_window.coeff),(int)(50*Main_window.coeff));

saveGraph.setBounds((int)(10*Main_window.coeff),(int)(470*Main_window.coeff),
(int)(740*Main_window.coeff),(int)(50*Main_window.coeff));

addData.setBounds((int)(400*Main_window.coeff),(int)(50*Main_window.coeff),(i
nt)(350*Main_window.coeff),(int)(50*Main_window.coeff));

labelFrom.setBounds((int)(10*Main_window.coeff),(int)(5*Main_window.coeff),(i
nt)(110*Main_window.coeff),(int)(50*Main_window.coeff));

labelTo.setBounds((int)(130*Main_window.coeff),(int)(5*Main_window.coeff),(in
t)(110*Main_window.coeff),(int)(50*Main_window.coeff));

labelWeight.setBounds((int)(250*Main_window.coeff),(int)(5*Main_window.coeff)
,(int)(120*Main_window.coeff),(int)(50*Main_window.coeff));

labelPath.setBounds((int)(10*Main_window.coeff),(int)(120*Main_window.coeff),
(int)(200*Main_window.coeff),(int)(50*Main_window.coeff));

scrollPane.setBounds((int)(790*Main_window.coeff),(int)(50*Main_window.coeff)
,(int)(410*Main_window.coeff),(int)(540*Main_window.coeff));

inputLineFrom.setBounds((int)(10*Main_window.coeff),(int)(50*Main_window.coef
f),(int)(110*Main_window.coeff),(int)(50*Main_window.coeff));

inputLineTo.setBounds((int)(130*Main_window.coeff),(int)(50*Main_window.coeff
),(int)(110*Main_window.coeff),(int)(50*Main_window.coeff));

inputLineWeight.setBounds((int)(250*Main_window.coeff),(int)(50*Main_window.c
oeff),(int)(110*Main_window.coeff),(int)(50*Main_window.coeff));

inputLinePath.setBounds((int)(200*Main_window.coeff),(int)(120*Main_window.co
eff),(int)(550*Main_window.coeff),(int)(50*Main_window.coeff));

returnMainWindow.setBounds((int)(10*Main_window.coeff),(int)(540*Main_window.
coeff),(int)(740*Main_window.coeff),(int)(50*Main_window.coeff));
        //Цвета полей
```

```
        editPanel.setBackground(Color.darkGray);
        listData.setBackground(Color.gray);
        edgeList.setBackground(Color.gray);
        vertexList.setBackground(Color.gray);
        listData.setForeground(Color.white);
        //добавление объектов на панель
        editPanel.add(edgeList);
        editPanel.add(vertexList);
        editPanel.add(scrollPane);
        //добавление действий
        inputLinePath.addActionListener(e->{
            Graph tmp0 = new Graph();
            String path = inputLinePath.getText();
            ArrayList<String> edges = new ArrayList<String>();
            try{
                FileInputStream fstream = new FileInputStream(path);
                BufferedReader br = new BufferedReader(new
InputStreamReader(fstream));
                String str;
                while ((str = br.readLine()) != null){
                    edges.add(str);
                }
            }
            catch (IOException ioexc){
                JOptionPane.showMessageDialog(Edit_window.this,
                        "При считывании возникла ошибка." );
                return;
            }
            reloadGraph(edges, graph, listOfEdges, listOfVertexes);
        });
        readButton.addActionListener(e->{
            String path = inputLinePath.getText();
            ArrayList<String> edges = new ArrayList<String>();
            if(!path.isEmpty()) {
                try {
                    FileInputStream fstream = new FileInputStream(path);
                    BufferedReader br = new BufferedReader(new
InputStreamReader(fstream));
                    String str;
                    while ((str = br.readLine()) != null) {
                        edges.add(str);
                    }
                }
                catch (IOException ioexc) {
                    JOptionPane.showMessageDialog(Edit_window.this,
                            "При считывании возникла ошибка.");
                    return;
                }
            } else {
                try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
                }
                catch (Exception ex){
                    JOptionPane.showMessageDialog(Edit_window.this,
                            "Forbidden.");
```

```
                    return;
                }

                JFileChooser fd = new JFileChooser(new
File(".").getAbsolutePath()); //дилоговое окно
                FileNameExtensionFilter filter = new
FileNameExtensionFilter(".txt", "txt", "text");
                fd.setFileFilter(filter);
                fd.setAcceptAllFileFilterUsed(false);

                int ret = fd.showDialog(null,"Выбор файла");
                if (ret == JFileChooser.APPROVE_OPTION) {
                    File file = fd.getSelectedFile(); //получение выбранного
файла
                    try(BufferedReader br = new BufferedReader(new
FileReader(file))) {
                        String str;
                        while ((str = br.readLine()) != null) {
                            edges.add(str);
                        }
                    }
                    catch(IOException ex){
                        JOptionPane.showMessageDialog(Edit_window.this,
                                "При считывании возникла ошибка.");
                        return;
                    }
                } else {
                    JOptionPane.showMessageDialog(Edit_window.this,
                            "Операция отменена.");
                    return;
                }
                try {

UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
                }
                catch (Exception ex){
                    JOptionPane.showMessageDialog(Edit_window.this,
                            "Forbidden.");
                    return;
                }
            }
            reloadGraph(edges, graph, listOfEdges, listOfVertexes);
        });
        removeEdge.addActionListener(e->{
            int index = edgeList.getSelectedIndex();
            if(index == -1){
                JOptionPane.showMessageDialog(Edit_window.this,
                        "Нет ребер для удаления" );
                return;
            }
            Edge tmp = listOfEdges.get(index);
            //непосредственно удаление ребра
            graph.removeEdge(listOfEdges.get(index));
            edgeList.removeItemAt(index);
            listData.append("Удалено ребро
"+listOfEdges.remove(index).toString()+'\n');
```

```
            //удаление вершин, исчезнувших после удаления ребра
            int first = tmp.first();
            boolean isFirstExists = false;
            int second = tmp.second(first);
            boolean isSecondExists = false;
            for(Edge edge : listOfEdges){
                if(first == edge.first() || first ==
edge.second(edge.first())){
                    isFirstExists = true;
                }
                if(second == edge.first() || second ==
edge.second(edge.first())){
                    isSecondExists = true;
                }
                if(isFirstExists && isSecondExists){
                    break;
                }
            }
            if(!isFirstExists){
                int indexFirst = listOfVertexes.indexOf(first);
                vertexList.removeItemAt(indexFirst);
                listData.append("Удалена вершина
"+listOfVertexes.remove(indexFirst).toString()+'\n');
            }
            if(!isSecondExists){
                int indexSecond = listOfVertexes.indexOf(second);
                vertexList.removeItemAt(indexSecond);
                listData.append("Удалена вершина
"+listOfVertexes.remove(indexSecond).toString()+'\n');
            }

        });
        removeVertex.addActionListener(e->{
            int index = vertexList.getSelectedIndex();
            if(index == -1){
                JOptionPane.showMessageDialog(Edit_window.this,
                        "Нет вершин для удаления" );
                return;
            }
            //удаление ребер инцедентных удаленной вершине
            for(Edge edge : graph.incEdges(listOfVertexes.get(index))){
                if(listOfEdges.indexOf(edge) != -1){
                    edgeList.removeItemAt(listOfEdges.indexOf(edge));
                    listData.append("Удалено ребро
"+listOfEdges.remove(listOfEdges.indexOf(edge)).toString()+'\n');
                    int second = edge.second(listOfVertexes.get(index));
                    boolean isSecondExists = false;
                    for(Edge edgeTmp : listOfEdges){
                        if(second == edgeTmp.first() || second ==
edgeTmp.second(edgeTmp.first())){
                            isSecondExists = true;
                        }
                        if(isSecondExists){
                            break;
                        }
                    }
```

```
                    if(!isSecondExists) {
                        int indexSecond =
listOfVertexes.indexOf(edge.second(listOfVertexes.get(index)));
                        vertexList.removeItemAt(indexSecond);
                        listData.append("Удалена вершина " +
listOfVertexes.remove(indexSecond) + '\n');
                        if (index > indexSecond) {
                            index--;
                        }
                    }
                }
            }
            graph.removeVertex(listOfVertexes.get(index));
            vertexList.removeItemAt(index);
            listData.append("Удалена вершина
"+listOfVertexes.remove(index).toString()+'\n');
        });
        deleteGraph.addActionListener(e->{
            graph.clear();
            edgeList.removeAllItems();
            listOfEdges.clear();
            vertexList.removeAllItems();
            listOfVertexes.clear();
            listData.append("Граф удален\n");
        });
        saveGraph.addActionListener(e -> {
            try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            }
            catch (Exception ex){
                JOptionPane.showMessageDialog(Edit_window.this,
                        "Forbidden.");
                return;
            }
            JFileChooser fd = new JFileChooser(new
File(".").getAbsolutePath()); //дилоговое окно
            FileNameExtensionFilter filter = new
FileNameExtensionFilter(".txt", "txt", "text");
            fd.setFileFilter(filter);
            fd.setAcceptAllFileFilterUsed(false);

            int ret = fd.showDialog(null,"Выбор файла");
            if (ret == JFileChooser.APPROVE_OPTION) {
                File graphWrite = fd.getSelectedFile(); //получение
выбранного файла
                File save;
                StringBuilder str = new StringBuilder(graphWrite.getPath());
                String newStr = str.substring(0,
str.lastIndexOf(graphWrite.getName()));
                int index = graphWrite.getName().lastIndexOf(".");
                if(index == -1) {
                    save = new File(newStr,
                            graphWrite.getName() + ".txt");
                } else {
                    save = new File(newStr,
```

```
                            graphWrite.getName().substring(0, index) +
".txt");
                }
                if (graphWrite.exists()) {
                    graphWrite.delete();
                }
                try {
                    graphWrite.renameTo(save);
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(Edit_window.this,
                            "Не удалось открыть файл!");
                    return;
                }
                try(BufferedWriter  writerGraph = new BufferedWriter( new
FileWriter(save)))
                {
                    for (Edge edge : listOfEdges) {
                        writerGraph.write(String.format("%d %d %.1f",
edge.first(), edge.second(edge.first()), edge.weight()));
                        writerGraph.newLine();
                    }
                    writerGraph.flush();
                }
                catch(IOException ex){
                    JOptionPane.showMessageDialog(Edit_window.this,
                            "Не удалось записать в файл!" );
                    return;
                }
                JOptionPane.showMessageDialog(Edit_window.this,
                        "Граф успешно сохранен в файл" );
            }

            try {

UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
            }
            catch (Exception ex){
                JOptionPane.showMessageDialog(Edit_window.this,
                        "Forbidden.");
                return;
            }
        });
        addData.addActionListener(e -> {
            if(!readFrom()) return;
            if(!readTo()) return;
            if(from == to){
                JOptionPane.showMessageDialog(Edit_window.this,
                        "Вершины ребра совпадают!");
                return;
            }
            if(!readWeight()) return;
            Edge tmp = new Edge(from, to, weight);
            boolean isExists = false;
            for(Edge edge : listOfEdges){
                if(edge.isEqual(tmp)){
                    isExists = true;
```

```
                break;
            }
        }
        if(isExists){
            JOptionPane.showMessageDialog(Edit_window.this,
                    "Ребро уже существует!" );
        } else {
            graph.addEdge(tmp);
            edgeList.addItem(tmp.toString());
            listOfEdges.add(tmp);
            listData.append("Добавлено ребро " + tmp.toString() + '\n');
            if (listOfVertexes.indexOf(from) == -1) {
                listOfVertexes.add(from);
                vertexList.addItem("" + from);
            }
            if (listOfVertexes.indexOf(to) == -1) {
                listOfVertexes.add(to);
                vertexList.addItem("" + to);
            }
            inputLineFrom.setText("");
            inputLineTo.setText("");
            inputLineWeight.setText("");
        }
        from = to = -1;
    });
    inputLineFrom.addActionListener(e -> {
        // Отображение введенного текста
        if(!readFrom()) return;
        inputLineTo.requestFocusInWindow();
    });
    inputLineTo.addActionListener(e -> {
        // Отображение введенного текста
        if(from == -1) {
            if(!readFrom()) return;
        }
        if(!readTo()) return;
        if(from == to){
            JOptionPane.showMessageDialog(Edit_window.this,
                    "Вершины ребра совпадают!");
            return;
        }
        inputLineWeight.requestFocusInWindow();
    });
    inputLineWeight.addActionListener(e -> {
        // Отображение введенного текста
        if(from == -1) {
            if(!readFrom()) return;
        }
        if(to == -1) {
            if(!readTo()) return;
        }
        if(from == to){
            JOptionPane.showMessageDialog(Edit_window.this,
                    "Вершины ребра совпадают!");
            return;
        }
```

```
            if(!readWeight()) return;
            Edge tmp = new Edge(from, to, weight);
            boolean isExists = false;
            for(Edge edge : listOfEdges){
                if(edge.isEqual(tmp)){
                    isExists = true;
                    break;
                }
            }
            if(isExists){
                JOptionPane.showMessageDialog(Edit_window.this,
                        "Ребро уже существует!" );
            } else {
                graph.addEdge(tmp);
                edgeList.addItem(tmp.toString());
                listOfEdges.add(tmp);
                listData.append("Добавлено ребро " + tmp.toString() + '\n');
                if (listOfVertexes.indexOf(from) == -1) {
                    listOfVertexes.add(from);
                    vertexList.addItem("" + from);
                }
                if (listOfVertexes.indexOf(to) == -1) {
                    listOfVertexes.add(to);
                    vertexList.addItem("" + to);
                }
                inputLineFrom.setText("");
                inputLineTo.setText("");
                inputLineWeight.setText("");
                inputLineFrom.requestFocusInWindow();
                from = to = -1;
            }
            from = to = -1;
        });
        returnMainWindow.addActionListener(e->{
            UnionFind uf = new UnionFind(graph.V());
            if(uf.isConnected(graph)){
                String title = "Окно подтверждения";
                UIManager.put("OptionPane.yesButtonText", "Да");
                UIManager.put("OptionPane.noButtonText", "Нет");
                int check = JOptionPane.showConfirmDialog(null, "Выйти из
редактора?", title, JOptionPane.YES_NO_OPTION);
                if (check == JOptionPane.YES_OPTION) {
                    //действия по сохранению графа
                    graphPanel.setGraph(graph);
                    //graphPanel.setMST(graph);
                    parent.setVisible(true);
                    dispose();
                }
            } else {
                JOptionPane.showMessageDialog(Edit_window.this,
                        "Перед выходом сделайте граф связным" );
            }
        });
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent we) {
```

```
                UnionFind uf = new UnionFind(graph.V());
                if(uf.isConnected(graph)){
                    String title = "Окно подтверждения";
                    UIManager.put("OptionPane.yesButtonText", "Да");
                    UIManager.put("OptionPane.noButtonText", "Нет");
                    int check = JOptionPane.showConfirmDialog(null, "Выйти из
редактора?", title, JOptionPane.YES_NO_OPTION);
                    if (check == JOptionPane.YES_OPTION) {
                        //действия по сохранению графа
                        graphPanel.setGraph(graph);
                        //graphPanel.setMST(graph);
                        parent.setVisible(true);
                        dispose();
                    }
                } else {
                    JOptionPane.showMessageDialog(Edit_window.this,
                            "Перед выходом сделайте граф связным" );
                }
            }
        });
    }
    private boolean reloadGraph(ArrayList<String> edges, Graph graph,
LinkedList<Edge> listOfEdges, LinkedList<Integer> listOfVertexes){
        Graph tmp0 = new Graph();
        for (String k : edges) {
            try (Scanner s = new Scanner(k)) {
                int args[] = new int[2];
                double weight;
                for (int count = 0; count < 2; count++) {
                    if (s.hasNextInt()) {
                        args[count] = s.nextInt();
                        if (args[count] < 0) {
                            throw new Exception();
                        }
                    } else {
                        throw new Exception();
                    }
                }
                if (s.hasNextDouble()) {
                    weight = s.nextDouble();
                    if (weight <= 0) {
                        throw new Exception();
                    }
                } else {
                    throw new Exception();
                }
                if (s.hasNext()) {
                    throw new Exception();
                }
                Edge tmp = new Edge(args[0], args[1], weight);
                boolean isExists = false;
                for(Edge edge : tmp0.edges()){
                    if(edge.isEqual(tmp)){
                        isExists = true;
                        break;
```

```
                }
            }
            if(isExists) {
                JOptionPane.showMessageDialog(Edit_window.this,
                        "Встречены повторяющиеся ребра!");
                return false;
            }
            tmp0.addEdge(tmp);

        } catch (Exception exc) {
            JOptionPane.showMessageDialog(Edit_window.this,
                    "Проверьте правильность данных в файле");
            inputLinePath.setText("");
            return false;
        }
    }
    graph.clear();
    edgeList.removeAllItems();
    listOfEdges.clear();
    vertexList.removeAllItems();
    listOfVertexes.clear();
    listData.append("Исходный граф удален\n");
    for (Edge edge : tmp0.edges()) {
        graph.addEdge(edge);
        edgeList.addItem(edge.toString());
        listOfEdges.add(edge);
        listData.append("Добавлено ребро " + edge.toString() + '\n');
        if (listOfVertexes.indexOf(edge.first()) == -1) {
            listOfVertexes.add(edge.first());
            vertexList.addItem("" + edge.first());
        }
        if (listOfVertexes.indexOf(edge.second(edge.first())) == -1) {
            listOfVertexes.add(edge.second(edge.first()));
            vertexList.addItem("" + edge.second(edge.first()));
        }
    }
    JOptionPane.showMessageDialog(Edit_window.this,
            "Граф считан");
    inputLinePath.setText("");
    return true;
}
private boolean readFrom(){
    try (Scanner s = new Scanner(inputLineFrom.getText())) {
        if (s.hasNextInt()) {
            from = s.nextInt();
            if (from < 0) {
                throw new Exception();
            }
        } else {
            throw new Exception();
        }
        if (s.hasNext()) {
            throw new Exception();
        }
    }
    catch(Exception exc) {
```

```
            JOptionPane.showMessageDialog(Edit_window.this,
                    "Проверьте правильность данных поля Откуда!");
            return false;
        }
        return true;
    }
    private  boolean readTo(){
        try (Scanner s = new Scanner(inputLineTo.getText())) {
            if (s.hasNextInt()) {
                to = s.nextInt();
                if (to < 0) {
                    throw new Exception();
                }
            } else {
                throw new Exception();
            }
            if (s.hasNext()) {
                throw new Exception();
            }
        }
        catch(Exception exc) {
            JOptionPane.showMessageDialog(Edit_window.this,
                    "Проверьте правильность данных поля Куда!");
            return false;
        }
        return true;
    }
    private boolean readWeight(){
        try (Scanner s = new Scanner(inputLineWeight.getText())) {
            if (s.hasNextDouble()) {
                weight = s.nextDouble();
                if (weight <= 0) {
                    throw new Exception();
                }
            } else {
                throw new Exception();
            }
            if (s.hasNext()) {
                throw new Exception();
            }
        }
        catch(Exception exc) {
            JOptionPane.showMessageDialog(Edit_window.this,
                    "Проверьте правильность данных поля Вес ребра!");
            return false;
        }
        return true;
    }
}
```

GraphVizualizer.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
```

```
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.util.*;
import java.lang.Math;

//класс, конвертирующий абстрактный граф в его плоскостное представление
public class GraphVizualizer extends JPanel {

    private LinkedList<Edge> listOfEdges; //список ребер исходного графа
    private LinkedList<Edge> listOfEdgesOfMSTStep; //список ребер
минимального остовного дерева на данном шаге
    private LinkedList<Integer> listOfVerteces;//список вершин
    private ArrayList<Coordinates> coordsOfVerteces;//координаты вершин
    public Integer indexOfSelectedVertex;

    public GraphVizualizer(){
        setLayout(null); //абсолютное позиционирование - расположение файлов
задается точно
        Font fontLogging = new Font(null, 1, 14);
        setFont(fontLogging);
    }
    public void setGraph(Graph graph) {
        setBackground(Color.gray);
        addMouseListener(new MyMouse());
        addMouseMotionListener(new MyMove());

        listOfVerteces = graph.vertexes();
        listOfEdges = graph.edges();
        listOfEdgesOfMSTStep = new LinkedList<>();
        coordsOfVerteces = new ArrayList<Coordinates>(graph.V());
        arrangement(); //располложение
        revalidate();
        repaint();
    }
    public void paint(Graphics graph) { //отрисовка графа
        super.paint(graph);
        if(listOfVerteces == null){
            return;
        }
        Graphics2D graph_2d = (Graphics2D) graph;
        graph_2d.setRenderingHint (RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON); //размывание крайних пикселей линий для
гладкости изображения
        Color oldColor = Color.black;
        for (Edge e : listOfEdges) {
            //настройка пера
            if(listOfEdgesOfMSTStep != null &&
listOfEdgesOfMSTStep.indexOf(e) != -1){
                graph_2d.setStroke(new BasicStroke(3));
                graph_2d.setColor(Color.cyan);
            } else {
                graph_2d.setStroke(new BasicStroke());
                graph_2d.setColor(Color.black);
            }
            int indexFirst = listOfVerteces.indexOf(e.first());
```

```
            int indexSecond = listOfVerteces.indexOf(e.second(e.first()));
            Coordinates from = coordsOfVerteces.get(indexFirst);
            Coordinates to = coordsOfVerteces.get(indexSecond);
            graph_2d.draw(new Line2D.Double(from.x, from.y, to.x, to.y));
            //отображение веса ребер
            graph_2d.setColor(oldColor);
            graph_2d.drawString("" + e.weight(), (to.x - from.x) * 2 / 5 +
from.x, (to.y - from.y) * 2 / 5 + from.y);
        }
        //работа с цветом

        //отображение вершин
        for (int i = 0; i < listOfVerteces.size(); i++) { //проход по всем
вершинам
            Coordinates center = coordsOfVerteces.get(i);
            Ellipse2D.Double v = new Ellipse2D.Double(center.x - 17,center.y
- 13,35,35);
            graph_2d.draw(v);
            graph_2d.setColor(Color.cyan);
            graph_2d.fill(v);
            graph_2d.setColor(oldColor);
            graph_2d.drawString("" + listOfVerteces.get(i), center.x - 5,
center.y + 10);
        }
    }
    private void arrangement() { //расстановка вершин/рёбер графа
        for (int i = 0; i < listOfVerteces.size(); i++) { //проход по всем
вершинам
            //вид правильного n-угольника
            coordsOfVerteces.add(new Coordinates((int)(335 *
Main_window.coeff + 150 * Math.cos(6.28 / listOfVerteces.size() * i)),
(int)(372 * Main_window.coeff + 150 * Math.sin(6.28 / listOfVerteces.size() *
i))));
        }
    }
    public void setMSTEdges(LinkedList<Edge> edges){
        listOfEdgesOfMSTStep = edges;
        revalidate();
        repaint();
    }
    private Integer find(Point2D cursorPosition){
        for(Coordinates center : coordsOfVerteces) {
            Ellipse2D.Double tmp = new Ellipse2D.Double(center.x-17,center.y-
13,35,35);
            if (tmp.contains(cursorPosition)) return
coordsOfVerteces.indexOf(center);
        }
        return null;
    }
    private class MyMouse extends MouseAdapter{
        @Override
        public void mousePressed(MouseEvent e) {
            indexOfSelectedVertex = find(e.getPoint());
        }
    }
    private class MyMove implements MouseMotionListener {
```

```
        @Override
        public void mouseMoved(MouseEvent e) {
            if(find(e.getPoint()) != null)
                setCursor(Cursor.getPredefinedCursor(Cursor.MOVE_CURSOR)); //
            else
                setCursor(Cursor.getDefaultCursor());
        }
        @Override
        public void mouseDragged(MouseEvent e) {
            if(indexOfSelectedVertex != null){
                Coordinates crd = new Coordinates(e.getX()+10,e.getY());
                if(crd.x<20) crd.x = 20;
                if(crd.y<20) crd.y = 20;
                if(crd.x>(int)(650 * Main_window.coeff)) crd.x = (int)(650 *
Main_window.coeff);
                if(crd.y>(int)(725 * Main_window.coeff)) crd.y = (int)(725 *
Main_window.coeff);
                coordsOfVerteces.set(indexOfSelectedVertex, crd);
                repaint();
            }
        }
    }
}
```

Coordinates.java

```
public class Coordinates {
    public int x;
    public int y;

    public Coordinates(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public Coordinates(Coordinates old) {
        this.x = old.x;
        this.y = old.y;
    }
}
```

AlgorithmBoruvki.java

```
import java.util.LinkedList;

public class AlgorithmBoruvki {
    private LinkedList<Edge> mst = new LinkedList<>();
    private int iteration = 0;
    private LinkedList<LinkedList<Edge>> memento = new LinkedList<>();
    private double weight = 0.0;
    private Graph graph;
    private LinkedList<Integer> listOfVertexes;
    private UnionFind uf;

    public AlgorithmBoruvki(Graph graph) {
        this.graph = graph;
        LinkedList<Edge> zeroStep = new LinkedList<>();
        memento.addLast(zeroStep);
```

```
        uf = new UnionFind(graph.V());
        listOfVertexes = graph.vertexes();
    }

    private void step() {
        Edge[] closest = new Edge[graph.V()];
        for (Edge e : graph.edges()) {
            int v = e.first(), w = e.second(v);
            int i = uf.find(listOfVertexes.indexOf(v)), j =
uf.find(listOfVertexes.indexOf(w));
            if (i == j) continue;   // same tree
            if (closest[i] == null || less(e, closest[i])) closest[i] = e;
            if (closest[j] == null || less(e, closest[j])) closest[j] = e;
        }

        for (int i = 0; i < graph.V(); i++) {
            Edge e = closest[i];
            if (e != null) {
                int v = e.first(), w = e.second(v);
                if (!uf.connected(listOfVertexes.indexOf(v),
listOfVertexes.indexOf(w))) {
                    mst.add(e);
                    weight += e.weight();
                    uf.union(listOfVertexes.indexOf(v),
listOfVertexes.indexOf(w));
                }
            }
        }
        memento.addLast(new LinkedList<>(mst));
    }

    private static boolean less(Edge e, Edge f) {
        return e.weight() < f.weight();
    }

    public LinkedList<Edge> nextStep(){
        if(mst.size() >= graph.V() - 1){
            if(memento.size() == iteration+1){
                return memento.get(iteration);
            } else {
                return memento.get(++iteration);
            }
        }
        if(memento.size() <= iteration+1){
            step();
        }
        return memento.get(++iteration);
    }

    public LinkedList<Edge> deletedEdges() {
        LinkedList<Edge> deletedEdges = new LinkedList<>();
        for(int i = memento.get(iteration-
1).size();i<memento.get(iteration).size();i++){
            deletedEdges.add(memento.get(iteration).get(i));
        }
        return deletedEdges;
```

```
    }

    public LinkedList<Edge> previousStep() {
        if(iteration <= 0){
            return memento.get(0);
        }
        return memento.get(--iteration);
    }

    public LinkedList<Edge> lastStep() {
        iteration = memento.size();
        for (; iteration < graph.V() && mst.size() < graph.V() - 1; iteration
*=2) {
            step();
        }
        iteration = memento.size() - 1;
        assert check(graph);
        return memento.getLast();
    }

    public int getIteration(){
        return iteration;
    }

    private boolean check(Graph graph) {
        // check that it is acyclic
        UnionFind uf = new UnionFind(graph.V());
        for (Edge e : edges()) {
            int v = e.first(), w = e.second(v);
            if (uf.connected(v, w)) {
                System.err.println("Not a forest");
                return false;
            }
            uf.union(v, w);
        }

        // check that it is a spanning forest
        for (Edge e : graph.edges()) {
            int v = e.first(), w = e.second(v);
            if (!uf.connected(v, w)) {
                System.err.println("Not a spanning forest");
                return false;
            }
        }
        return true;
    }

    public LinkedList<Edge> edges() {
        return mst;
    }
}
```

Graph.java

```
import java.util.*;

public class Graph {
```

```
    private int V = 0;
    private int E = 0;
    private Map<Integer, LinkedList<Edge>> incEdges;

    public Graph(){
        incEdges = new HashMap<Integer, LinkedList<Edge>>();
    }
    public void clear(){
        V = 0;
        E = 0;
        incEdges = new HashMap<Integer, LinkedList<Edge>>();
    }
    public int V(){ return V; }
    public int E(){ return E; }
    public void addEdge(Edge e) {
        boolean isExists = false;
        for(Edge edge : edges()){
            if(edge.isEqual(e)){
                isExists = true;
                break;
            }
        }
        if(!isExists){
            E++;
        }
        if(incEdges.containsKey(e.first())) {
            incEdges.get(e.first()).add(e);
        }
        else {
            V++;
            incEdges.putIfAbsent(e.first(), new LinkedList<Edge>());
            incEdges.get(e.first()).add(e);
        }
        if(incEdges.containsKey(e.second(e.first()))) {
            incEdges.get(e.second(e.first())).add(e);
        }
        else {
            V++;
            incEdges.putIfAbsent(e.second(e.first()), new
LinkedList<Edge>());
            incEdges.get(e.second(e.first())).add(e);
        }
    }
    public void removeEdge(Edge e) {
        boolean isExists = false;
        for(Edge edge : edges()){
            if(edge.isEqual(e)){
                isExists = true;
                break;
            }
        }
        if(!isExists){
            return;
        }
        E--;
        incEdges.get(e.first()).remove(e);
```

```
    if(incEdges.get(e.first()).size() == 0){
        V--;
        incEdges.remove(e.first());
    }
    incEdges.get(e.second(e.first())).remove(e);
    if(incEdges.get(e.second(e.first())).size() == 0){
        V--;
        incEdges.remove(e.second(e.first()));
    }
}
public void removeVertex(int v){
    if(incEdges.get(v) == null){
        return;
    }
    int size = incEdges.get(v).size();
    if(size != 0) {
        for (int i = incEdges.get(v).size() - 1; i >= 0; i--) {
            removeEdge(incEdges.get(v).get(i));
        }
    } else{
        V--;
        incEdges.remove(v);
    }

}
public LinkedList<Edge> incEdges(int v){
    LinkedList<Edge> a = new LinkedList<Edge>();
    for(Edge e : incEdges.get(v)) {
        a.add(e);
    }
    return a;
}
public LinkedList<Edge> edges(){
    LinkedList<Edge> a = new LinkedList<Edge>();
    for(int v : incEdges.keySet()){
        for(Edge e : incEdges.get(v)) {
            if (e.second(v) > v) {
                a.add(e);
            }
        }
    }
    return a;
}
public LinkedList<Integer> vertexes() {
    LinkedList<Integer> a = new LinkedList<Integer>();
    for(int v : incEdges.keySet()){
        a.add(v);
    }
    return a;
}
public String toString(){
    String str = "";
    for(Edge e: edges()){
        str += e.toString()+'\n';
    }
    return str;
```

```
    }
}
```

Edge.java

```
public class Edge implements Comparable<Edge> {
    private final int u;
    private final int v;
    private final double weight;
    public Edge(int u, int v, double w) {
        this.u = u;
        this.v = v;
        this.weight = w;
    }
    public double weight() {
        return weight;
    }
    public int first() {
        return u;
    }
    public int second(int u) {
        if(u == this.u) return v;
        else if (u == v) return this.u;
        else throw new IllegalArgumentException("Illegal endpoint");
    }
    @Override
    public int compareTo(Edge that) {
        return Double.compare(this.weight, that.weight);
    }
    public String toString() {
        return String.format("%d-%d %.1f", u, v, weight);
    }
    public boolean isEqual(Edge other){
        return (other.v == this.v) && (other.u == this.u) || (other.v ==
this.u) && (other.u == this.v);
    }
}
```

UnionFind.java

```
import java.util.LinkedList;

public class UnionFind {
    private int[] parent;
    private byte[] rank;
    private int count;

    public UnionFind(int n) {
        if (n < 0) throw new IllegalArgumentException();
        count = n;
        parent = new int[n];
        rank = new byte[n];
        for (int i = 0; i < n; i++) {
            parent[i] = i;
            rank[i] = 0;
        }
    }
```

```
    public boolean isConnected(Graph graph) {
        LinkedList<Integer> listOfVertexes = graph.vertexes();
        for(Edge e : graph.edges()) {
            union(listOfVertexes.indexOf(e.first()),
listOfVertexes.indexOf(e.second(e.first())));
        }
        if(count() > 1){
            return false;
        } else {
            return true;
        }
    }

    public int find(int p) {
        validate(p);
        while (p != parent[p]) {
            parent[p] = parent[parent[p]];    // path compression by halving
            p = parent[p];
        }
        return p;
    }

    public int count() {
        return count;
    }

    public boolean connected(int p, int q) {
        return find(p) == find(q);
    }

    public void union(int p, int q) {
        int rootP = find(p);
        int rootQ = find(q);
        if (rootP == rootQ) return;

        // make root of smaller rank point to root of larger rank
        if      (rank[rootP] < rank[rootQ]) parent[rootP] = rootQ;
        else if (rank[rootP] > rank[rootQ]) parent[rootQ] = rootP;
        else {
            parent[rootQ] = rootP;
            rank[rootP]++;
        }
        count--;
    }

    private void validate(int p) {
        int n = parent.length;
        if (p < 0 || p >= n) {
            throw new IllegalArgumentException("index " + p + " is not
between 0 and " + (n-1));
        }
    }
}
```