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 гр. 7383 _____ Зуев Д.В.

Студентка гр. 7383 _____ Маркова А.В.

Студент гр. 7383 _____ 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 гр. 7383	_____	Зуев Д.В.
Студентка гр. 7383	_____	Маркова А.В.
Студент гр. 7383	_____	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.

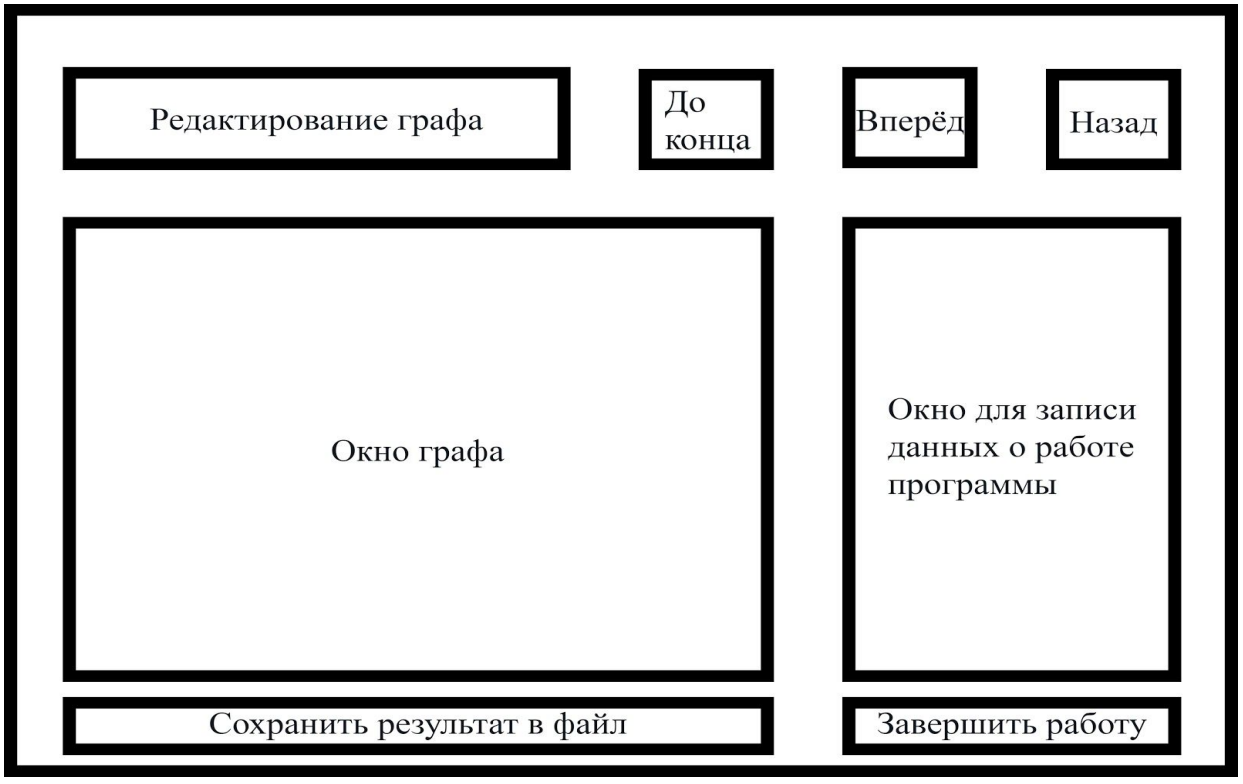

Рисунок 1 – Главное окно

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.

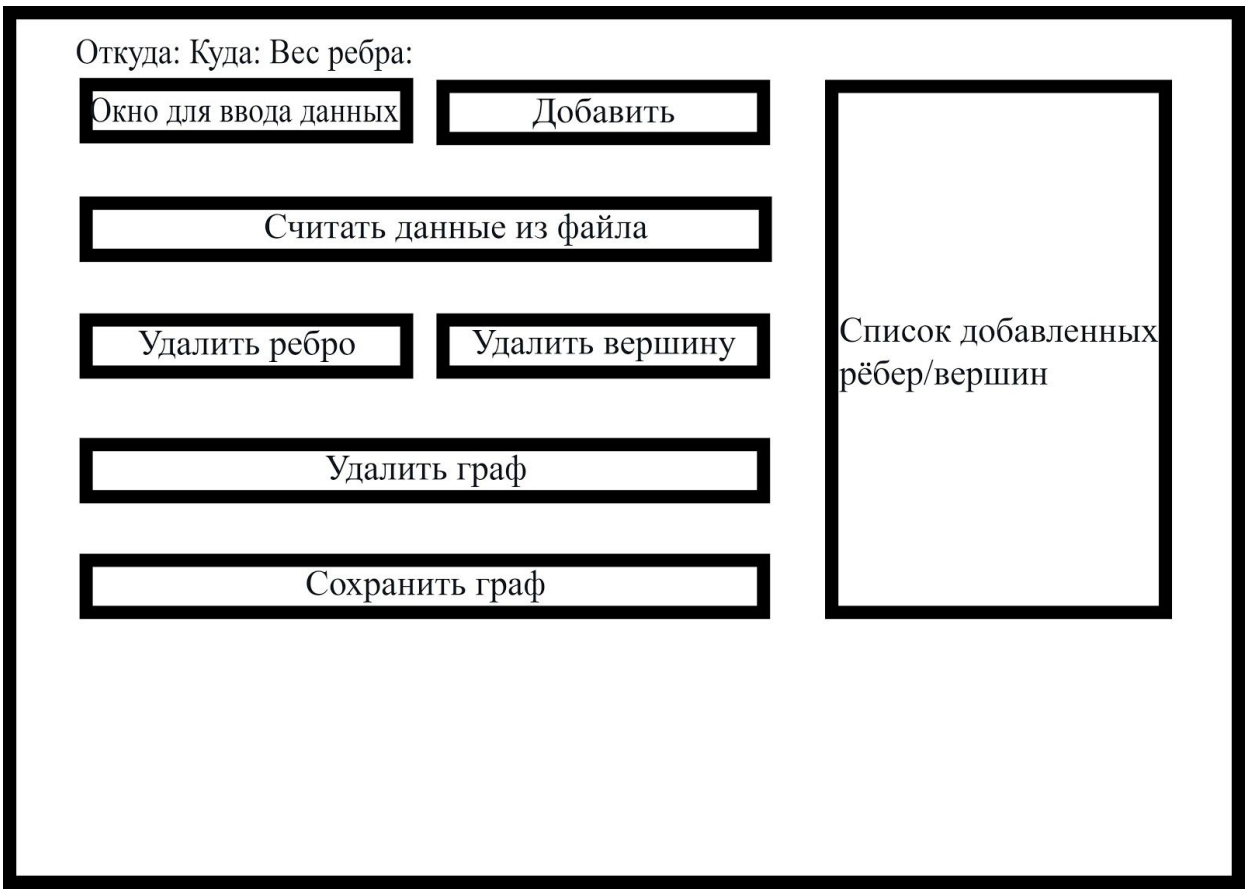

Рисунок 2 – Окно редактирования графа

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.

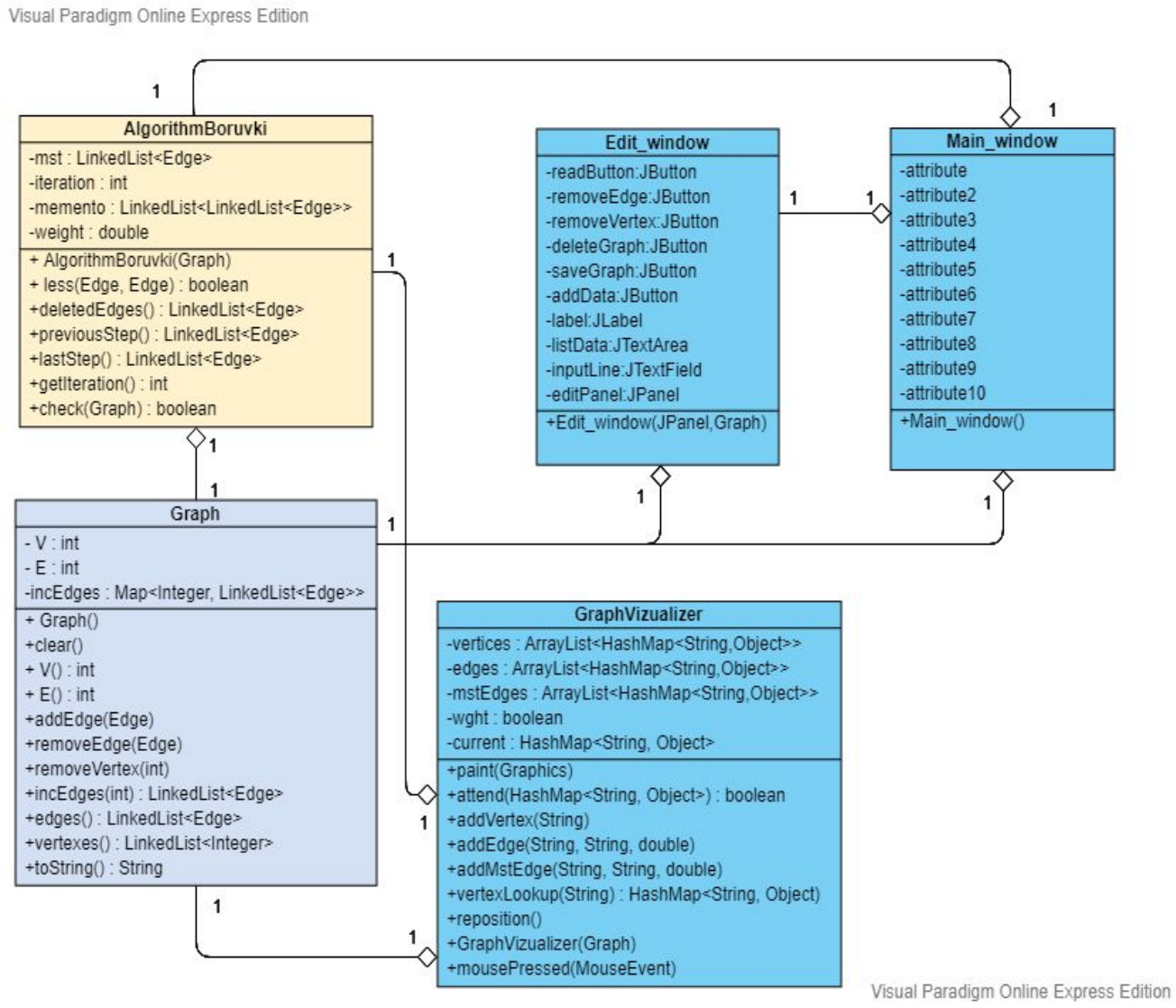

Рисунок 3 – UML диаграмма графического интерфейса

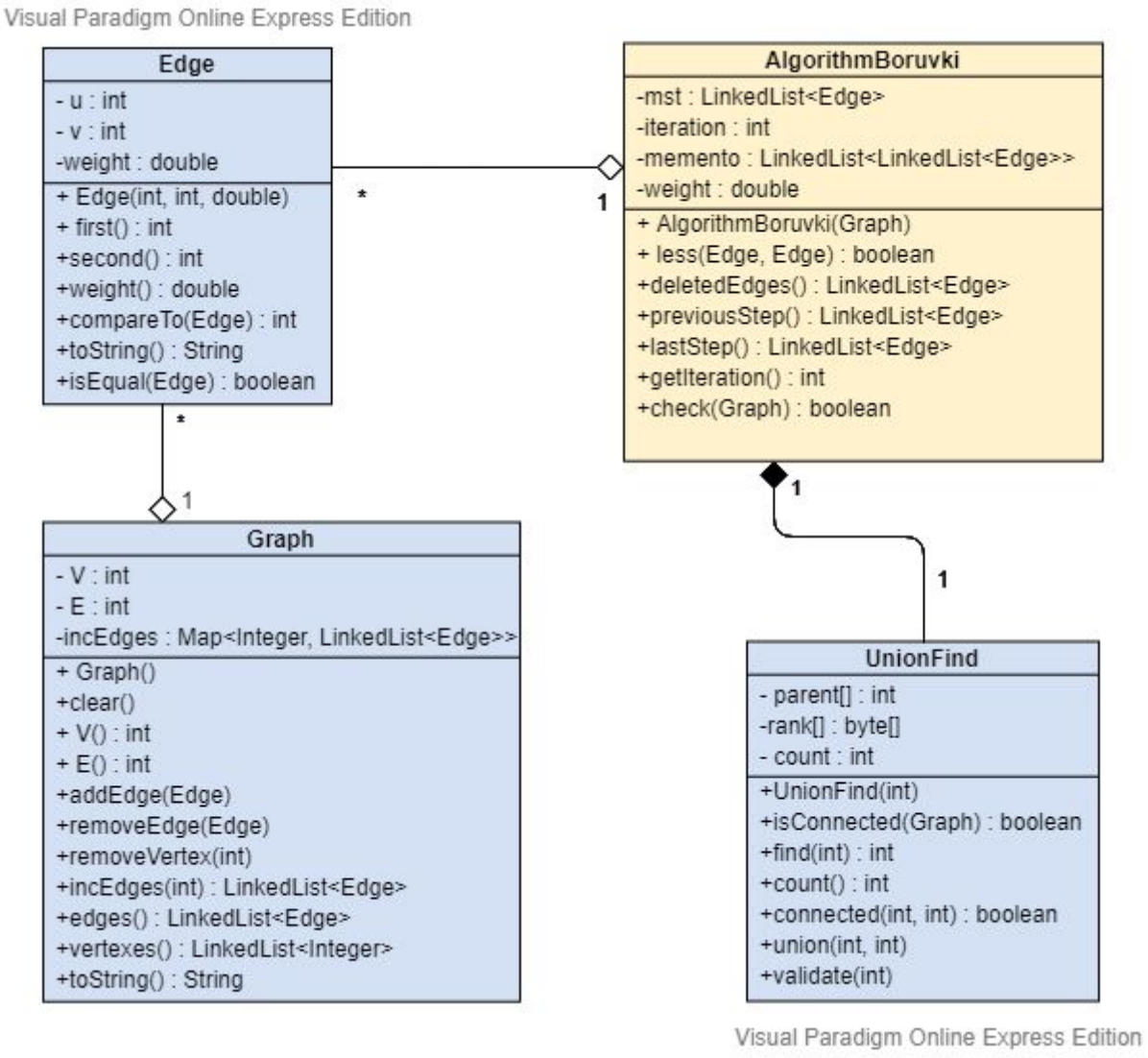

Рисунок 4 – UML диаграмма алгоритма Борувки

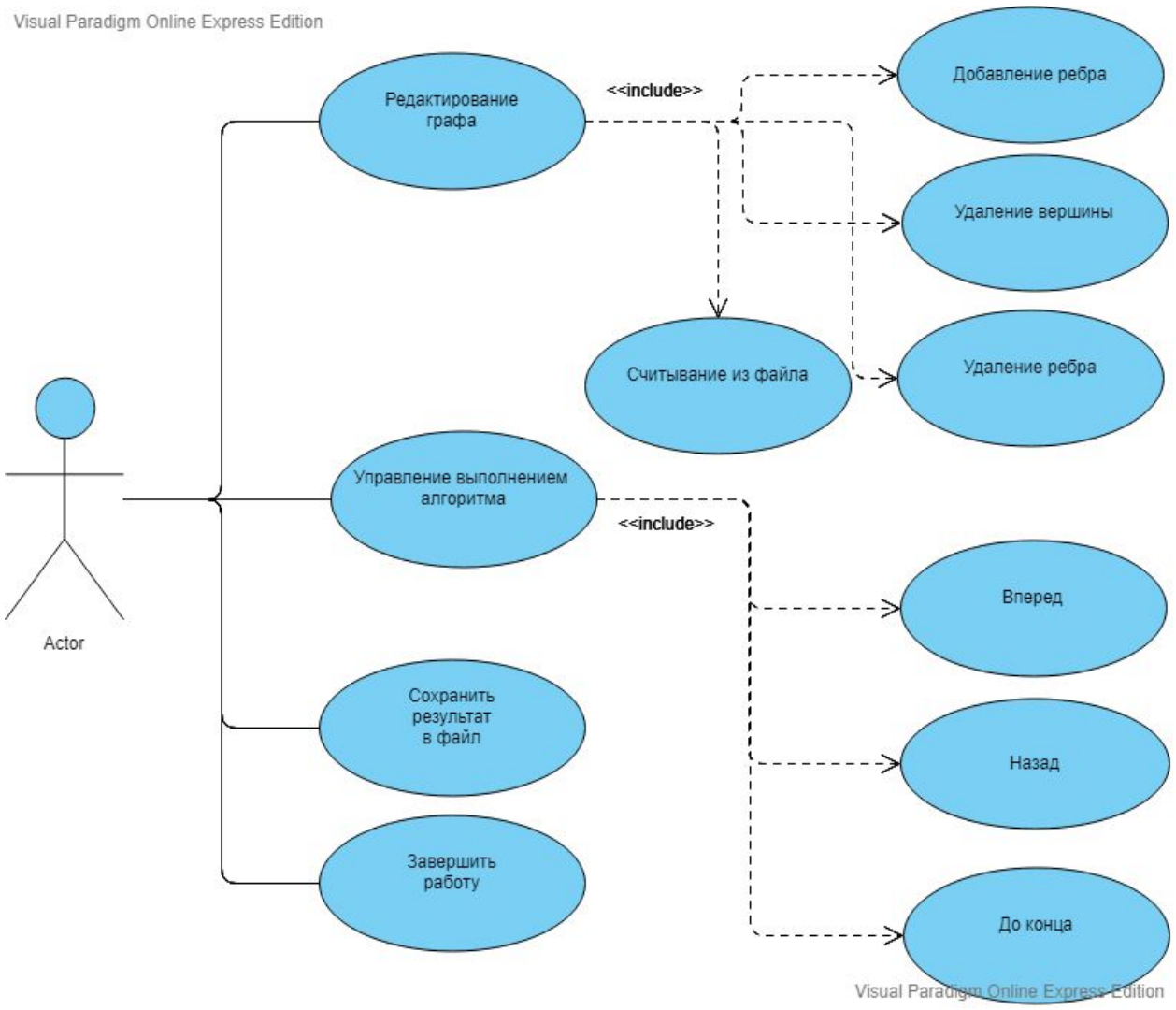

Рисунок 5 – UML диаграмма взаимодействия пользователя с программой

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

Прототип (06.07.2019):

Будут реализованы все окна программы. Все кнопки интерфейса будут реализовать только открытие или закрытие соответствующих им окон, остальные кнопки будут отключены.

Итерация 1 (08.07.2019):

Будет реализован ввод графа с клавиатуры. Будет реализовано окно логов, отображающее введенные ребра графа. Будет реализовано удаление ребер и вершин графа и удаление графа целиком.

Итерация 2 (10.07.2019):

Будет реализовано отображение графа и окно логов, которое будет отображать данные о работе программы. Будет реализован алгоритм Борувки с пошаговым выполнением. Будет реализован ввод и вывод графа из файла.

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.

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.

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.

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.

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.

`private class MyMouse` – класс, метод которого получает координаты места, куда нажимает курсор.

`private class MyMove` – класс, реализующий возможность пользователя перемещать вершины нарисованного графа.

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 удаленных на данном шаге ребер из предыдущего шага.
- 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 неудачи 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

ЗАКЛЮЧЕНИЕ

В ходе данной учебной практики были изучены основы программирования на языке Java, были получены навыки работы с библиотеками визуализации Swing и AWT, 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.URL: stepic.org/course/187/syllabus
2. The Java Tutorials.URL: https://docs.oracle.com/javase/tutorial/index.html
3. Г.Шилдт. Swing руководство для начинающих, 2007. – 705 с.

ПРИЛОЖЕНИЕ А

Код программы