**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по учебной практике**

Тема: Разработка визуализатора алгоритма ЯПД

|  |  |  |
| --- | --- | --- |
| Студент гр. 8383 |  | Костарев К.В. |
| Студент гр. 8383 |  | Мололкин К.А. |
| Студент гр. 8383 |  | Федоров И.И. |
| Руководитель |  | Жангиров Т.Р. |

Санкт-Петербург

2020

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студент Костарев К.В. группы 8383 | | |
| Студент Мололкин К.А. группы 8383 | | |
| Студент(ка) Федоров И.И. группы 8383  Тема практики: наименование темы | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.  Алгоритм: Ярника-Прима-Дейкстры. | | |
| Сроки прохождения практики: 29.06.2020 – 12.07.2020 | | |
| Дата сдачи отчета: 00.07.2020 | | |
| Дата защиты отчета: 00.07.2020 | | |
|  | | |
| Студент |  | Костарев К.В. |
| Студент |  | Мололкин К.А. |
| Студент |  | Федоров И.И. |
| Руководитель |  | Жангиров Т.Р. |

**Аннотация**

В данной учебной практической работе были освоены методы создания приложений с графическим пользовательским интерфейсом, реализация изучение алгоритма Ярника-Прима-Дейкстры, который используется для нахождения минимального остовного дерева, а также реализация взаимодействия графического интерфейса и модели алгоритма на языке программирования высокого уровня Java. Так же в данной работе был освоен навык работы в команде, каждый член которой выполнял свое задание.

**Summary**

In this training practical work, we have mastered the methods of creating applications with a graphical user interface, the implementation of the Jarnik-Prima-Dijkstra algorithm, which is used to find the minimum spanning tree, as well as the implementation of the interaction of the graphical interface and the algorithm model in the high-level programming language Java. Also in this work, the skill of working in a team was mastered, each member of which performed its own task.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Требования к программе | 6 |
| 1.1. | Исходные требования к программе\* | 6 |
| 1.3. | Уточнение требований после сдачи 1-ой версии | 10 |
| 2. | План разработки и распределение ролей в бригаде | 11 |
| 2.1. | План разработки | 11 |
| 2.2. | Распределение ролей в бригаде | 11 |
| 3. | Особенности реализации | 12 |
| 3.1. | Структуры данных | 12 |
| 3.2. | Основные методы | 16 |
| 4. | Тестирование | 21 |
|  | Заключение | 25 |
|  | Список использованных источников | 26 |
|  | Приложения. Исходный код – только в электронном виде | 27 |

**введение**

**Целью** данной учебной практики является освоение навыков создание приложения с графическим интерфейсом, а также реализация алгоритма Ярника‑Прима‑Дейкстры, по поиску минимального остовного дерева в графе.

**Задачей** данной практики является, создание классов графического интерфейса, классов, решающих задачу поиска минимального остовного дерева, а также классов, тестирующих программу, на языке программирования Java.

Алгоритм Ярника-Прима-Дейкстры – алгоритм по нахождению минимального остовного дерева в графе. Данный алгоритм применим к взвешенному не ориентированному связному графу.

Описание алгоритма

1. Произвольно выбирается первая вершина.
2. Выбирается ребро минимального веса, исходящее из данной вершины, и добавляется в остовное дерево.
3. Затем в цикле пока остов не будет содержать N - 1 ребро, где N – количество вершин в графе, добавляется ребро которое имеет один конец в остовном дереве, а второй, не содержащийся в остове.

**1. требования к программе**

* 1. **Исходные Требования к программе**

**1.1.1. Требования к вводу исходных данных**

Граф, для которого необходимо применить алгоритм, должен быть представлен как массив ребер, где каждое ребро представляет собой строку формата *<наименование вершины 1> <наименование вершины 2> <вес ребра>.* Отметим, что вводимый граф должен быть неориентированный, т.е. не должно быть двух строк с одинаковыми вершинами. В противном случае программа выведет сообщение об ошибке. В программе предусмотрен ввод данных как из интерфейса так и из стороннего файла формата TXT.

**1.1.2. Графический интерфейс**

Интерфейс программы предлагает возможность пользователю ввести данные как самостоятельно, так и с помощью загрузки из стороннего TXT файла. В функционал графического интерфейса также входит и пошаговое выполнение алгоритма Ярника-Прима-Дейкстры, т.е. пользователь имеет возможность контролировать выполнение алгоритма кнопками «вперед» (вперед на один шаг) и «результат» (в конец работы алгоритма).

На рис. 1 представлен прототип интерфейса стартового окна прототипа.

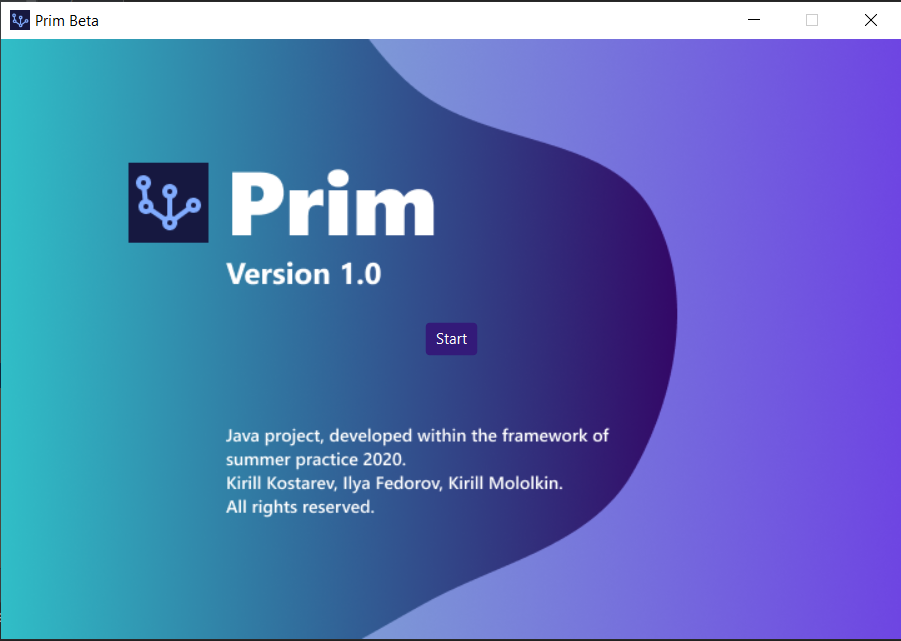


Рисунок 1 – Прототип стартового окна

На рис. 2 представлен прототип интерфейса главного окна прототипа.

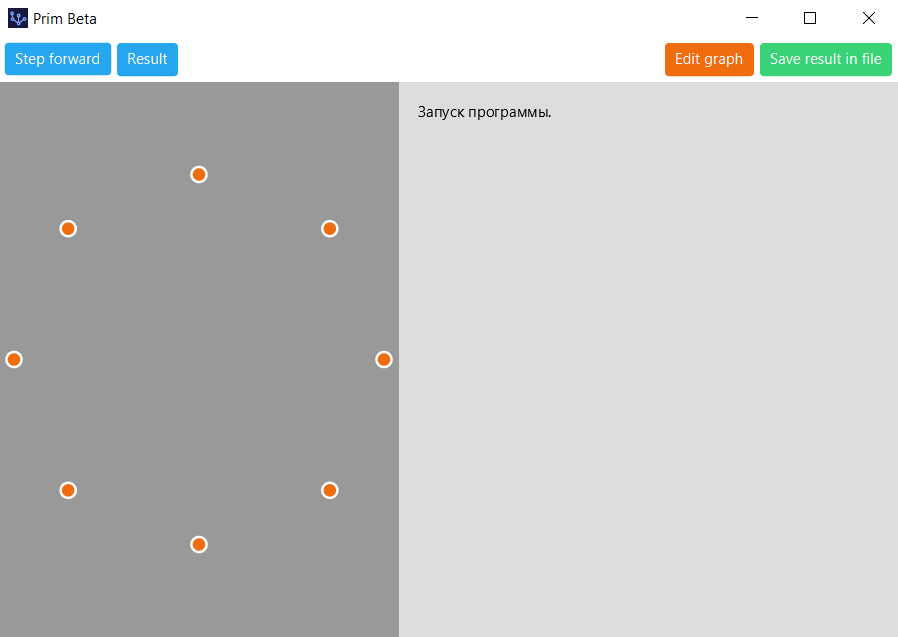


Рисунок 2 – Прототип главного окна

Главное окно логически разделено на три подраздела: верхний – раздел управления с кнопками редактирования графа, управления алгоритмом («вперед», «результат») и сохранения результата файла; нижний левый – графическое представление графа, нижний правый – логирование событий. Такой интерфейс является удобным и интуитивным для пользователя, так как отделяет логические составляющие программы.

Кнопки «вперед» и «результат» инициализируют выполнение алгоритма. В окне визуализации графа до начала инициализации алгоритма будет выводится граф согласно входным данным, далее на каждом шаге – ребра и вершины остовного дерева графа, при этом ребра и вершины, не входящие в остовное дерево на определенном шаге, будут изображены более тусклым цветом. В окне текстовой демонстрации алгоритма будут выводится сообщения о добавленных ребрах в остовное дерево, успешном его выполнении, результате работы и сохранении результата в файл.

Кнопка «Редактировать граф» вызывает открытие окна редактирования графа, прототип которого представлен на рис. 3.

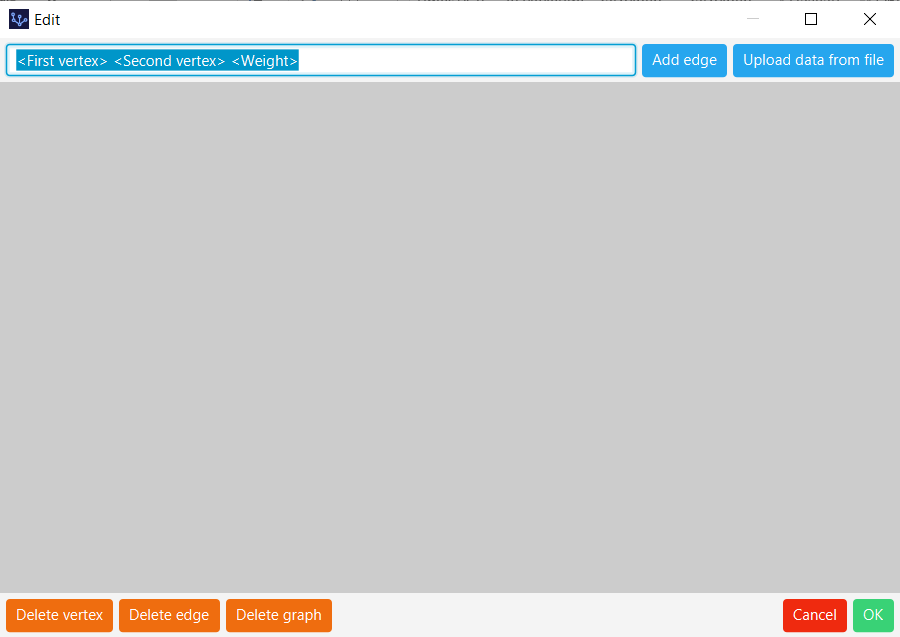


Рисунок 3 – Прототип окна редактирования графа

На рис. 4, 5, 6 представлены UML диаграммы классов, реализующих графический интерфейс, работу алгоритма, а также диаграмма взаимодействия пользователя с программой.

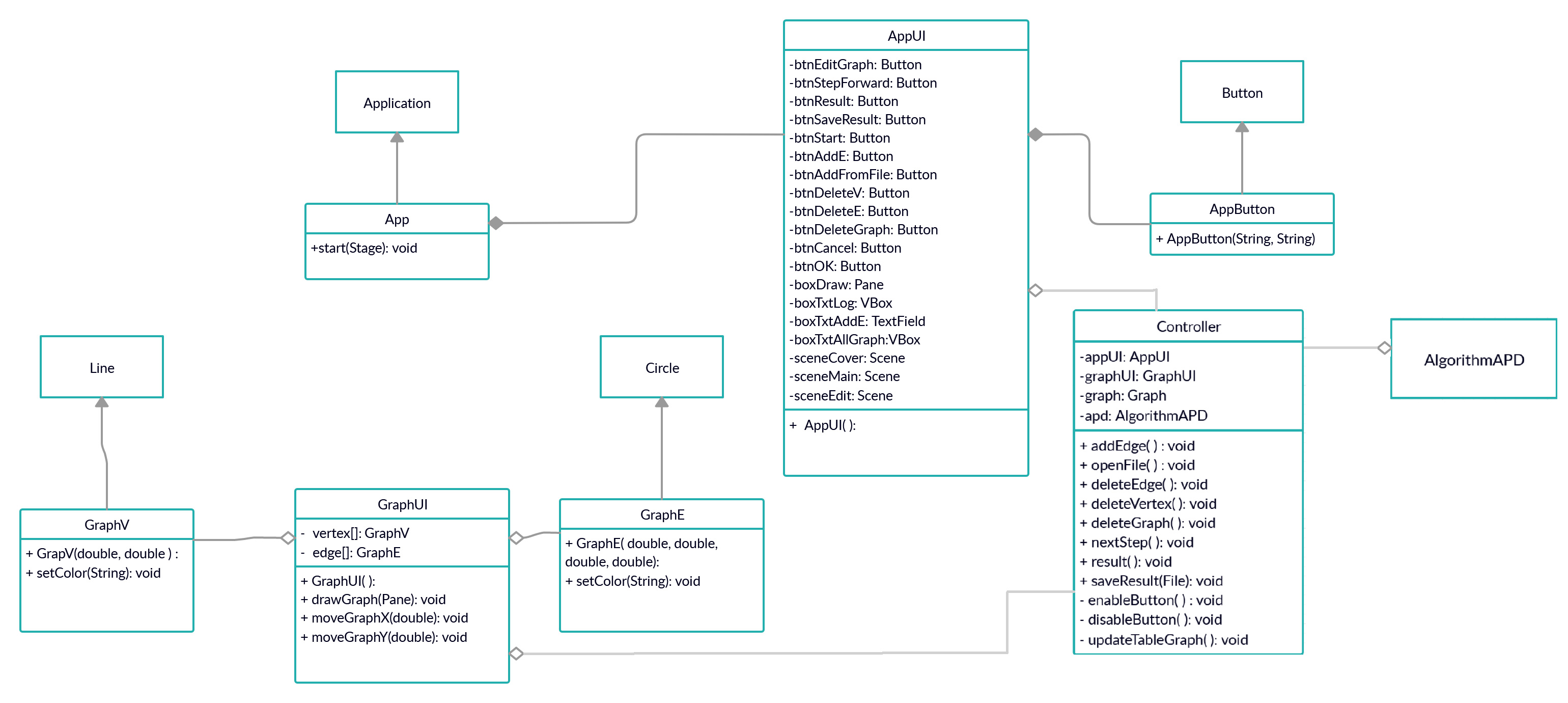


Рисунок 4 – диаграмма классов графического интерфейса

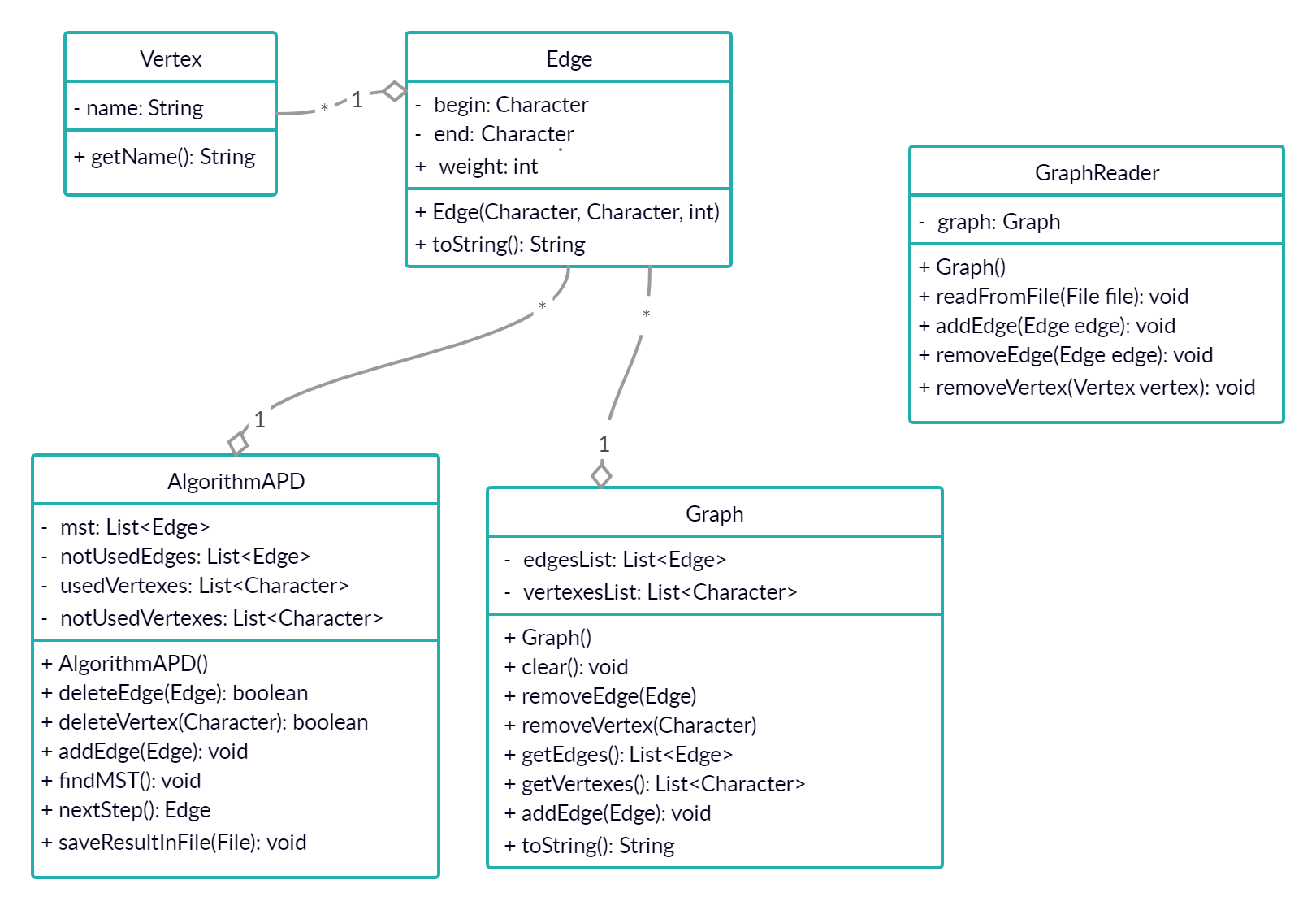


Рисунок 5 – Диаграмма классов алгоритма

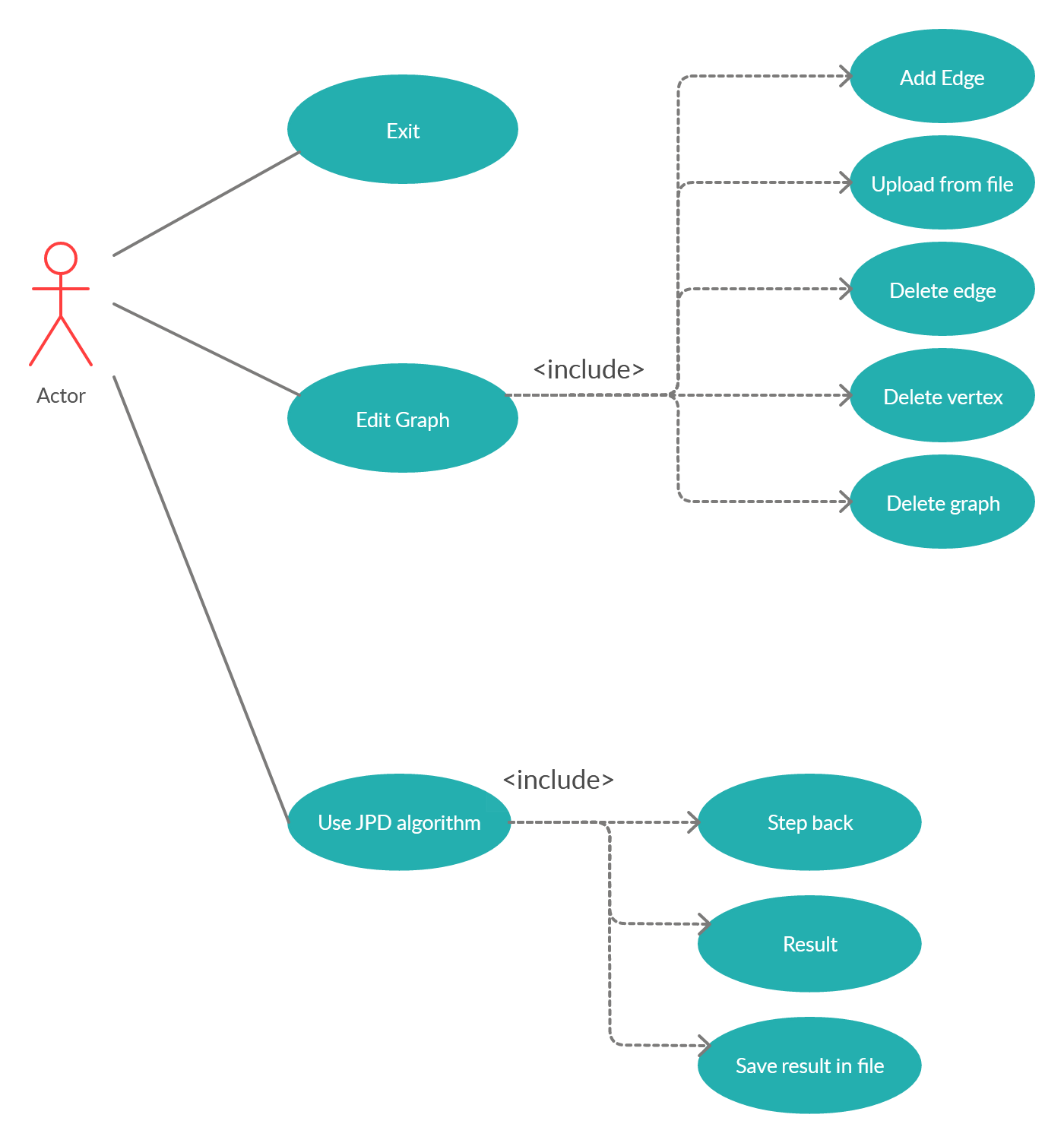


Рисунок 6 – Диаграмма взаимодействия пользователя с интерфейсом

* 1. **Уточнение требований после сдачи 1-ой версии**

После сдачи первой версии были получены следующие требования: добавить возможность включать и отключать логирование, вынести вершины графа в отдельный класс, а также сделать отдельный класс для чтения графа.

**2. План разработки и распределение ролей в бригаде**

**2.1. План разработки**

04.07.2020 - Согласование спецификации и плана разработки приложения.

04.07.2020 - Предоставление прототипа, демонстрирующего часть интерфейса пользователя без использования основной функции визуализации алгоритма.

07.07.2020 - Сдача 1-ой версии приложения, имеющего пошаговый вывод работы алгоритма. В данной версии пользователь имеет возможность подать на вход граф только из файла.

10.07.2020 - Сдача 2-ой версии приложения.

**2.2. Распределение ролей в бригаде**

Мололкин Кирилл – архитектор, ответственный за распределение задач, реализацию алгоритма и логирование.

Костарев Кирилл – фронтед-разработчик, ответственный за графический интерфейс.

Федоров Илья – ответственный за связку работы алгоритма и интерфейса, тестирование.

**3. Особенности реализации**

* 1. **Использованные структуры данных**

**public class Vertex –** класса вершин графа.

Поля:

* private String name – имя вершины;

**public class Edge** – класс ребер.

Поля:

* private Vertex begin, end – вершины ребра.
* private int weight – вес ребра.

**public class Graph –** класс хранящий граф.

Поля:

* private List<Edge> edgesList – список ребер графа.
* private List<Vertex> vertexesList – список вершин графа.
* private static final Logger LOGGER – логгер.

**public class GraphReader** – класса для чтения графа.

Поля:

* private Graph graph.
* private static final Logger LOGGER – логгер.

public class AlgorithmApd – класс алгоритма

Поля:

* private List<Edge> notUsedEdges – ребра вне остовного дерева.
* private List<Vertex> usedVertexes – вершины остовного дерева.
* private List<Vertex> notUsedVertexes – вершны вне дерева.
* private List<Edge> minimumSpanningTree – остовное дерево.
* private boolean isInitializedNotUsedLists - переменна для определения была ли выполнена инициализация полей.
* private boolean isConnectivityChecked – переменная для определения была ли выполнена проверка на связанность графа.

**public interface Observer** – интерфейс наблюдателя используется для изменения логирования.

**public class ObserverManager –** класс наблюдателя, используется для рассылки изменений.

Public class AppUI – класс, хранящий в себе все элементы интерфейса программы (модальные окна, сцены, кнопки, текстовые поля, слои, изображения и цвета). Конструктор класса AppUI() инициализирует нижеописанные поля и устанавливает им внешний вид. Поля класса:

* Private final class AppButton extends Button – внутренний класс, необходимый для создания кнопок в едином для приложения дизайне. Содержит два конструктора AppButton(String txt, String color) и AppButton(String txt, String color, Boolean disable), где первый по умолчанию делает кнопку активной, а во втором этот параметр передается непосредственно.
* Public final Image imgAppIcon, imgAppCover – иконка приложения и заставка стартового экрана соответственно.
* Public static final String colorRed, colorOrange, colorBlue, colorCyan, colorGreen – цвета в 16-ричном формате, определенные для всего приложения.
* Public final Button btnEditGraph – кнопка, вызывающая окно редактирования графа.
* Public final Button btnStepForward – кнопка вызова следующего шага работы алгоритма.
* Public final Button btnResult – кнопка, показывающая результат работы алгоритма, пропуская промежуточные шаги.
* Public final Button btnSaveResult – кнопка, вызывающая окно выбора директории в проводнике для сохранения результата работы алгоритма в формате PNG.
* Public final Button btnStart – кнопка на стартовом экране приложения.
* Public final Button btnAddE – кнопка добавления ребра в граф.
* Public final Button btnFromFile – кнопка добавления графа из TXT файла, где описано множество ребер этого графа, вызывает окно выбора файла в проводнике.
* Public final Button btnDeleteV – кнопка удаления вершины.
* Public final Button btnDeleteE – кнопка удаления ребра.
* Public final Button btnClear – кнопка сброса исходных данных для ввода новых.
* Public final Button btnOK – кнопка закрытия окна редактирования графа с сохранением данных.
* Public final CheckBox checkLog – чекбокс включения или отключения логирования работы приложения.
* Public final Pane boxDraw – слой, «холст», на котором визуализируется граф и работа алгоритма.
* Public final TextArea boxTxtLog – не редактируемое многострочное текстовое поле, в котором отображается логирование работы приложения.
* Public final TextField boxTxtAddE – однострочное текстовое поле для ввода ребра, которое необходимо добавить в граф по нажатии кнопки btnAddE или вершин, которые необходимо удалить из графа по нажатию кнопки btnDeleteV.
* Public final TableView boxTableAllGraph – табличное представление графа в виде ребер, т.е. таблица из 3 колонок: Vertex 1, Vertex 2, Weight. При нажатии на одну из строк таблицы, представляющую ребро графа, это ребро можно удалить по нажатию кнопки btnDeleteE.
* Public final Scene sceneCover – сцена стартового экрана приложения, где указываются разработчики, название, номер версии и визуальное оформление. Содержит только кнопку btnStart начала работы.
* Public final Scene sceneMain – сцена основного экрана приложения, где демонстрируется визуализация графа и алгоритма в частности, логирование и кнопки управления алгоритмом, редактирования графа и сохранения результата.
* Public final Scene sceneEdit – сцена окна редактирования графа, которая содержит табличное представление графа и кнопки редактирования.
* Public final Stage windowEdit – окно редактирования графа, содержащее сцену редактирования графа.
* Public final DirectoryChooser windowSaveResult – окно проводника для выбора директории, в которой необходимо сохранить результат.
* Public final FileChooser windowAddFromFile – окно проводника для выбора TXT файла, из которого необходимо считать граф.
* Public final Alert windowError – окно уведомления об ошибке.

Public class GraphUI – класс для графического представления графа. Имеет поля:

* Private class GraphV extends Circle – внутренний класс для представления вершины графа. Имеет поле private final Text name, представляющее собой название вершины.
* Private class GraphE extends Line – внутренний класс для представления ребра графа. Имеет следующие поля:
  + Private final GraphV v1, v2 – вершины, которые ребро соединяет.
  + Private final Text weight – вес ребра.
* Private final LinkedList<GraphV> graphVertixes – список вершин графа в их графическом представлении.
* Private final ArrayList<GraphE> graphEdges – список ребер графа в их графическом представлении.
* Private final Pane boxDraw – слой, на котором визуализируется граф, то есть boxDraw из AppUI, для простоты обращения к «холсту».
* Private double centerX – координата Х центра boxDraw.
* Private double centerY – координата Y центра boxDraw.
* Private double radius – максимальная длина радиуса окружности, которую можно вписать в boxDraw. Необходимо, так как граф рисуется как множество вершин правильного многоугольника.

Public class Controller – класс, организующий основную внутреннюю логику работы приложения и взаимодействия между ее элементами. Реализует паттерн МВЦ. Имеет следующие поля:

* Private final AppUI appUI – элементы интерфейса.
* Private final GraphUI – графическое представление графа.
* Private final Graph graph – техническое представление графа.
* Private final GraphReader graphReader – считывание графа.
* Private final AlgorithmAPD apd – алгоритм.

Public class App extends Application – класс приложения, где и происходит его запуск. Собственных полей не имеет.

**3.2. Основные методы**

1. public class Vertex:

* public Vertex(String name) – конструктор.
* public String getName() – геттер поля name.
* public boolean equals(Object o) – сравнение ребер.
* public String toString() – переводит вершину в строку.

2. public class Edge:

* public Edge(Vertex start, Vertex stop, int weight) – конструктор.
* public Vertex getBegin() – геттер вершины.
* public Vertex getEnd() – геттер вершины.
* public int getWeight() – геттер веса.
* public String toString() – перевод в строку.
* public boolean equals(Object o) – сравнение ребер.

3. public class Graph:

* public Graph(TextArea textArea) – конструктор, так же добавляет хендлер логера.
* public boolean addEdge(Edge edge) – добавление вершины графа.
* private boolean checkEdge(Edge edge) – провекра существет ли ребро в графе.
* public void clear() – очистка графа.
* public void removeEdge(Edge edge) – удаление ребра графа.
* public void removeVertex(Vertex vertex) – удаление вершины графа.
* public List<Edge> getEdgesList() – геттер списка ребер.
* public List<Vertex> getVertexesList() – геттер списка вершин.
* public void updateNotify(boolean isOn) – изменения состояния логирования.
* public String toString() – вывод списка ребер графа.

4. public class GraphReader:

* public GraphReader(Graph graph, TextArea textArea) – конструктор, так же добавляет хендлер логера.
* public void addEdge(Edge edge) – добавление ребра.
* public void readGraphFromFile(File file) – считывание графа из файла по реберно.
* public void removeEdge(Edge edge) – удаление ребра.
* public void removeVertex(Vertex vertex) – удаление вершины.
* public void updateNotify(boolean isOn) – изменения состояния логера.

5. public class AlgorithmAPD:

* public AlgorithmAPD(Graph graph, TextArea textArea) – конструктор, так же добавляет хендлер логера.
* private void initialize() – инициализация полей класса.
* private boolean graphConnectivityCheck() – проверка графа на связанность

Функция создает стек вершин, куда кладется одна вершина, затем в цикле пока стек не пустой, снимем вершину со стека и добавляем все ей смежные, при вытаскивании вершины со стека она помечается пройденной, если прошли все вершины, значит граф связан.

* public void clear() – очищает все поля графа.
* public Edge nextEdgeAtMst() – функция выполняет шаг алгоритма

Функция проходится по списку неиспользованных вершин, ищет минимальное ребро одни конец которого входит в остов, а другой нет, затем добавляет его в остов удаляя новую вершину остова из списка неиспользованных вершин, а так же полученное ребро удаляет из неиспользованных.

* public List<Edge> result() – вызывает функцию след шага, до тех пор пока все вершины не будут включены в остов.

6. public class ObserverManager:

* public void addObserver(Observer observer) – добавление наблюдателя.
* public void notify(boolean isLoggerOn) – оповещение.

1. Public class GraphUI:
   * Public void graphToUI(Graph g) – перерводит технический граф g в его графическое представление.
   * Public void addToSpanning(Edge e) – видоизменяет внешний вид ребра e, которое было добавлено в остовное дерево (изменяет цвет ребра и его вершин).
   * Public void drawGraph() - рисует граф, представленный списками графических ребер и вершин в boxDraw.
   * Public void moveGraph() – изменяет размеры графа, если был изменен размер холста boxDraw, так, чтобы он занимал максимально возможную площадь. Например, при изменении размера окна приложения.
   * Private void updateCoordinates() – метод для изменения координат ребер и вершин, используется moveGraph().
   * Private void update() – метод для обновления полей centerX, centerY и radius, используется moveGraph().
   * Private void clear() – метод «очистки» холста boxDraw и удаления графа. Используется graphToUI.
2. Public class Controller:
   1. Public void opeenFile(File file) – метод считывания данных графа из file.
   2. Public void addEdge() – метод считывания и добавления ребра в граф.
   3. Public void deleteEdge() – метод удаления ребра.
   4. Public void deleteVertex() – метод удаления вершины.
   5. Public void clear() – метод сброса исходных данных.
   6. Public void nextStep() – метод инициализации следующего шага алгоритма.
   7. Public void result() – метод инициализации результата алгоритма.
   8. Public void saveResult(File dir) – метод сохранения результата в папку dir.
   9. Private void disableBtn() – метод отключения функциональных кнопок для работы с редактированием графа и алгоритмом, используется clear, deleteEdge, deleteVertex.
   10. Private void enableBtn() – метод включения функциональных кнопок, используется openFile, addEdge.
   11. Private void update() – метод обновления данных в графическом и табличном представлениях графа. Используется всеми редактирующими граф методами класса.
   12. Private void updateTableGraph() – метод обновления табличного представления графа. Используется update.
   13. Private void updateBoxDraw() – метод обновления графического представления графа. Используется update.
3. Public class App extends Application:
   1. Public void start(Stage stage) – инициализирует запуск приложения и определяет обработчики событий для элементов интерфейса приложения, собирая необходимые данные для controller.
4. **ТЕСТИРОВАНИЕ**

**4.1. Описание тестирования**

Тестирование кода было осуществлено с помощью JUnit – фреймворка для модульного тестирования Java программ (т.е. тестирования отдельных участков кода, например, методов или классов). Объектами тестирования являются разработанный командой алгоритм Ярника-Прима-Дейкстры, а также структуры данных, классы и их методы, реализованные для работы алгоритма.

**4.2. План тестирования**

* Тестирование класса Graph для хранения графа:
* Добавление корректного ребра в граф. Метод должен добавить ребро в граф и вернуть true как корректное завершение.
* Добавление идентичного ребра (граф уже содержит подобное ребро) в граф. В этом случае метод должен вернуть false как сообщение об ошибке и не добавлять переданное ребро.
* Удаление вершины графа.
* Удаление ребра графа.
* Очистка графа. Метод должен очистить список вершин и ребер.
* Тестирование класса AlgorithmAPD, реализующего алгоритм:
* Определение связности переданного графа (алгоритм Прима работает только со связными графами). Метод должен выбросить исключение, если переданный граф является несвязным, иначе он должен закончиться без ошибок.
* Вычисление минимального остова. Алгоритм должен правильно найти минимальный остов графа. Проверка происходит на наборе.
* Тестирование класса GraphReader:
* Считывание графа из корректного файла. Метод должен считать граф из файла без каких либо ошибок и исключений.
* Считывание графа из некорректного файла. Метод должен выбросить исключение.
* Добавление ребра. Метод на основе результата работы аналогичного метода класса Graph должен либо выполнится без ошибок, либо выбросить исключение.

**4.3. Краткое описание классов тестирования**

Класс GraphTest тестирует методы структуры данных для хранения графа:

* void addEdge() – тестовый метод добавления корректного ребра в граф.
* void addSameEdges() – тестовый метод добавления двух одинаковых ребер в граф. Метод класса должен вернуть false в качестве сообщения об ошибке.
* void removeVertex() – тестовый метод удаления вершины из графа. Метод класса должен корректно удалить вершину, включая смежные с ней ребра.
* void clearTest() – тестовый метод очистки графа. Метод должен очистить список вершин и список ребер.

Класс AlgorithmAPDTest тестирует часть методов алгоритма, реализованного в классе AlgorithmAPD. Тестовый классы содержит методы и поля:

* void addEdge() – тестовый метод добавления корректного ребра в граф.
* void addSameEdge() – тестовый метод добавления двух одинаковых ребер в граф. Метод класс должен выбросить исключение и не добавлять второе идентичное ребро.

Класс AlgorithmAPDParametriseTest тестирует на основе набора данных часть методов класса AlgorithmAPD. Методы и поля:

* static Iterable<Object[]> dataForTest() – метод с аннотацией @Parameterized.Parametres, возвращает Iterable, каждая запись которого представляет собой массив объектов. Одна запись – один набор данных для теста. В данном случае это три файла и один параметр типа int.
* AlgorithmAPDParametriseTest(File, File, File, int) – конструктор тестового класса. Принимает на вход набор тестовых данных.
* void corConnectivityGraphCheck() – тестовый метод проверки связности графа. Метод должен завершиться успешно, т.к. ему передаются связные графы.
* void badConnectivityGraphCheck() – тестовый метод проверки связности графа. Метод класса должен выбрасывать исключения, т.к. ему передаются только несвязные графы.
* void resultTest() - тестовый метод нахождения минимального остова. Проверяет правильно ли найден остов.

Класс GraphReaderTest – класс для тестирования методов класс GraphReader, предназначенного для считывания графа.

* void readGraphFromExistFile() – тестовый метод считывания графа из корректного файла. Метод должен закончиться без ошибок и исключений.
* void readGraphFromNotExistFile() – тестовый метод считывания графа из некорректного файла. Метод класс должен выбросить исключение.
* void addEdge() – тестовый метод добавления корректного ребра в граф. На основе результата аналогичного метода класса Graph должен закончиться без ошибок и исключений.
* void addSameEdge() – тестовый метод добавления двух одинаковых ребер в граф. Метод класса на основе результата работы аналогичного метода в классе Graph (вернет boolean значение) должен выбросить исключение и не добавлять второе идентичное ребро.

Класс AutoTestSuit – класс для объединения всех тестовых классов и их запуска из одного метода. Объявлен с аннотациями @RunWith(Suite.class) и @Suite.SuiteClasses({}), с перечислением тестовых классов для запуска.

Результаты тестов представлены в папке TestReports в формате pdf.

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения учебной практики были изучены основы программирования на языке Java, были получены навыки работы с библиотекой для реализации GUI JavaFX и библиотекой JUnit для модульного тестирования. Было создано приложение с графическим интерфейсом, реализующее пошаговую демонстрацию выполнения алгоритма Ярника-Прима-Дейкстры. Был получен опыт командной разработки проекта, распределения обязанностей, а также совместной работы с общей системой контроля версий.

**список использованных источников**

1. Г. Шилдт. Java. Полное руководство, 10-е издание, 2018. 1408 с.

2. Java Базовый курс //Stepik.URL: <https://stepik.org/course/187/syllabus>

3. Учебное пособие по программированию на языке JAVA / Герасимова Т.В. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2006. 77 с.

4. Java documentation // docs.oracle.com. URL: https://docs.oracle.com/en/java/javase/11/ /index.html.

**ПРИЛОЖЕНИЕ**

**Код классов реализующих алгоритм**

**package** org.apd;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.util.logging.LogManager;

**public** **class** Main {

**public** **static** void main(**String**[] args) {

App.main(args);

}

}

**package** org.apd.algorithm;

**import** java.util.Objects;

**public** **class** Edge {

**private** Vertex begin, end;

**private** int weight;

**public** Edge(Vertex start, Vertex stop, int weight) {

**this**.begin = start;

**this**.end = stop;

**this**.weight = weight;

}

**public** Vertex getBegin() {

**return** begin;

}

**public** Vertex getEnd() {

**return** end;

}

**public** int getWeight() {

**return** weight;

}

**@Override**

**public** **String** toString() {

**return** begin + " " + end + " " + weight;

}

**@Override**

**public** boolean equals(**Object** o) {

**if** (**this** == o) **return** **true**;

**if** (!(o **instanceof** Edge)) **return** **false**;

Edge edge = (Edge) o;

**return** ((**this**.getBegin().equals(edge.getBegin())) && (**this**.getEnd().equals(edge.getEnd()))) ||

((**this**.getEnd().equals(edge.getBegin())) && (**this**.getBegin().equals(edge.getEnd())));

}

**@Override**

**public** int hashCode() {

**return** **Objects**.hash(getBegin(), getEnd(), getWeight());

}

}

**package** org.apd.algorithm;

**import** java.util.Objects;

**public** **class** Vertex {

**public** **String** name;

**public** Vertex(**String** name) {

**this**.name = name;

}

**public** **String** getName() {

**return** name;

}

**@Override**

**public** **String** toString() {

**return** name;

}

**@Override**

**public** boolean equals(**Object** o) {

**if** (**this** == o) **return** **true**;

**if** (!(o **instanceof** Vertex)) **return** **false**;

Vertex vertex = (Vertex) o;

**return** getName().equals(vertex.getName());

}

**@Override**

**public** int hashCode() {

**return** **Objects**.hash(getName());

}

}

**package** org.apd.algorithm;

**import** javafx.scene.control.TextArea;

**import** org.apd.ApplicationHandler;

**import** java.util.ArrayList;

**import** java.util.LinkedList;

**import** java.util.List;

**import** java.util.logging.**\***;

**public** **class** Graph **implements** Observer {

**private** **static** **final** **Logger** LOGGER = **Logger**.getLogger(Graph.class.getName());

**private** **List**<Edge> edgesList;

**private** **List**<Vertex> vertexesList;

**public** Graph() {

LOGGER.info("Create new graph");

edgesList = **new** **ArrayList**<>();

vertexesList = **new** **LinkedList**<>();

LOGGER.removeHandler(**new** **ConsoleHandler**());

}

**public** Graph(TextArea textArea){

LOGGER.info("Create new graph");

edgesList = **new** **ArrayList**<>();

vertexesList = **new** **LinkedList**<>();

LOGGER.setLevel(**Level**.INFO);

**Handler** handler = **new** ApplicationHandler(textArea);

LOGGER.addHandler(handler);

}

**public** boolean addEdge(Edge edge) {

LOGGER.log(**Level**.INFO,"Try to add new edge: " + edge.toString());

**if**(!checkEdge(edge)) **return** **false**;

LOGGER.log(**Level**.INFO,"Edge: " + edge.toString() + " doesn't exist in the graph");

edgesList.add(edge);

**if** (!vertexesList.contains(edge.getBegin())) {

LOGGER.log(**Level**.INFO, "Vertex: " + edge.getBegin().toString() + " is new in graph, add it to vertexes list");

vertexesList.add(edge.getBegin());

}

**if** (!vertexesList.contains(edge.getEnd())){

LOGGER.log(**Level**.INFO, "Vertex: " + edge.getEnd().toString() + " is new in graph, add it to vertexes list");

vertexesList.add(edge.getEnd());

}

**return** **true**;

}

**private** boolean checkEdge(Edge edge) {

**for** (Edge curEdge : edgesList) {

**if** (curEdge.equals(edge)) {

**return** **false**;

}

}

**return** **true**;

}

**public** void clear() {

LOGGER.log(**Level**.INFO, "Clear graph");

edgesList.clear();

vertexesList.clear();

}

**public** void removeEdge(Edge edge) {

LOGGER.log(**Level**.INFO, "Remove edge: " + edge.toString() + ", from graph");

edgesList.remove(edge);

boolean isContainFirstVertex = **false**;

boolean isContainSecondVertex = **false**;

**for** (Edge curEdge : edgesList) {

**if** (curEdge.getBegin().equals(edge.getBegin()) || curEdge.getEnd().equals(edge.getBegin()))

isContainFirstVertex = **true**;

**if** (curEdge.getBegin().equals(edge.getEnd()) || curEdge.getEnd().equals(edge.getEnd()))

isContainSecondVertex = **true**;

}

**if** (!isContainFirstVertex) {

vertexesList.remove(edge.getBegin());

LOGGER.log(**Level**.CONFIG, "No edges, connected to vertex: " + edge.getBegin());

}

**if** (!isContainSecondVertex) {

LOGGER.log(**Level**.CONFIG, "No edges, connected to vertex: " + edge.getEnd());

vertexesList.remove(edge.getEnd());

}

}

**public** void removeVertex(Vertex vertex) {

LOGGER.log(**Level**.CONFIG, "Remove vertex: " + vertex + " from graph");

vertexesList.remove(vertex);

edgesList.removeIf(edge -> edge.getBegin().equals(vertex) || edge.getEnd().equals(vertex));

}

**public** **List**<Edge> getEdgesList() {

**return** edgesList;

}

**public** **List**<Vertex> getVertexesList() {

**return** vertexesList;

}

**@Override**

**public** **String** toString() {

var sb = **new** **StringBuilder**();

**for** (Edge edge : edgesList) {

sb.append(edge.toString()).append("\n");

}

**return** sb.toString();

}

**@Override**

**public** void updateNotify(boolean isOn) {

LOGGER.setLevel(**Level**.OFF);

}

}

**package** org.apd.algorithm;

**import** javafx.scene.control.TextArea;

**import** org.apd.ApplicationHandler;

**import** java.io.File;

**import** java.util.Scanner;

**import** java.util.logging.Handler;

**import** java.util.logging.Level;

**import** java.util.logging.Logger;

**public** **class** GraphReader **implements** Observer {

**private** **static** **final** **Logger** LOGGER = **Logger**.getLogger(AlgorithmAPD.class.getName());

**private** Graph graph;

**public** GraphReader(Graph graph, TextArea textArea) {

**this**.graph = graph;

LOGGER.setLevel(**Level**.INFO);

**Handler** handler = **new** ApplicationHandler(textArea);

LOGGER.addHandler(handler);

}

**public** GraphReader(Graph graph) {

**this**.graph = graph;

LOGGER.setLevel(**Level**.INFO);

}

**public** void addEdge(Edge edge) **throws** **Exception** {

**if** (!graph.addEdge(edge)) {

LOGGER.log(**Level**.WARNING, "Try to add edge: " + edge.toString());

**throw** **new** **Exception**("edge already exist");

}

}

**public** void readGraphFromFile(**File** file) **throws** **Exception** {

LOGGER.log(**Level**.INFO, "Read graph from file: " + file.getName());

var scanner = **new** **Scanner**(file).useDelimiter(**System**.getProperty("line.separator"));

**while** (scanner.hasNext()) {

**String**[] curLine = scanner.nextLine().split(" ");

Edge edge = **new** Edge(**new** Vertex(curLine[0]), **new** Vertex(curLine[1]), **Integer**.parseInt(curLine[2]));

addEdge(edge);

}

}

**public** void removeEdge(Edge edge) {

graph.removeEdge(edge);

}

**public** void removeVertex(Vertex vertex) {

graph.removeVertex(vertex);

}

**@Override**

**public** void updateNotify(boolean isOn) {

LOGGER.setLevel(**Level**.OFF);

}

}

**package** org.apd.algorithm;

**import** javafx.scene.control.TextArea;

**import** org.apd.ApplicationHandler;

**import** java.awt.**\***;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.**\***;

**import** java.util.List;

**import** java.util.logging.Handler;

**import** java.util.logging.Level;

**import** java.util.logging.Logger;

**public** **class** AlgorithmAPD **implements** Observer{

**private** **static** **final** **Logger** LOGGER = **Logger**.getLogger(AlgorithmAPD.class.getName());

**private** Graph graph;

**private** **List**<Edge> notUsedEdges = **new** **ArrayList**<>();

**private** **List**<Vertex> usedVertexes = **new** **ArrayList**<>();

**private** **List**<Vertex> notUsedVertexes = **new** **ArrayList**<>();

**private** **List**<Edge> minimumSpanningTree = **new** **ArrayList**<>();

**private** boolean isInitializedNotUsedLists = **false**;

**private** boolean isConnectivityChecked = **false**;

**public** AlgorithmAPD(Graph graph, TextArea textArea) {

**this**.graph = graph;

LOGGER.setLevel(**Level**.INFO);

**Handler** handler = **new** ApplicationHandler(textArea);

LOGGER.addHandler(handler);

}

**public** AlgorithmAPD(Graph graph) {

**this**.graph = graph;

}

**private** void initialize() {

**if** (!isInitializedNotUsedLists) {

LOGGER.log(**Level**.INFO, "Initialize arrays before algorithm");

notUsedEdges = **new** **ArrayList**<>(graph.getEdgesList());

notUsedEdges.removeIf(edge -> edge.getBegin().equals(edge.getEnd()));

notUsedVertexes = **new** **ArrayList**<>(graph.getVertexesList());

usedVertexes.add(notUsedVertexes.remove(0));

isInitializedNotUsedLists = **true**;

}

}

**private** boolean graphConnectivityCheck() {

**if** (isConnectivityChecked) **return** **true**;

LOGGER.log(**Level**.INFO, "Check graph connectivity");

var checkedVertexesList = **new** boolean[graph.getVertexesList().size()];

**List**<Edge> edgesList = graph.getEdgesList();

**List**<Vertex> vertexesList = graph.getVertexesList();

**Stack**<Vertex> stackForDFS = **new** **Stack**<>();

stackForDFS.push(vertexesList.get(0));

LOGGER.log(**Level**.CONFIG, "Start DFS, checking graph connectivity.");

**while** (!stackForDFS.empty()) {

Vertex curVertex = stackForDFS.pop();

checkedVertexesList[vertexesList.indexOf(curVertex)] = **true**;

LOGGER.log(**Level**.CONFIG, "Check vertex: " + curVertex.toString() + " neighbours");

**for** (Edge edge : edgesList) {

**if** ((edge.getBegin().equals(curVertex)) && (!checkedVertexesList[vertexesList.indexOf(edge.getEnd())])) {

LOGGER.log(**Level**.CONFIG, "Add vertex: " + edge.getEnd().toString() + " to stack");

stackForDFS.push(edge.getEnd());

}

**if** ((edge.getEnd().equals(curVertex)) && (!checkedVertexesList[vertexesList.indexOf(edge.getBegin())])) {

LOGGER.log(**Level**.CONFIG, "Add vertex: " + edge.getBegin().toString() + " to stack");

stackForDFS.push(edge.getBegin());

}

}

}

**for** (**Boolean** bool : checkedVertexesList) {

**if** (!bool) {

LOGGER.log(**Level**.INFO, "Graph isn't connected");

**return** **false**;

}

}

isConnectivityChecked = **true**;

LOGGER.log(**Level**.INFO, "Graph is connected");

**return** **true**;

}

**public** void clear() {

LOGGER.log(**Level**.INFO, "Clear graph and all fields");

notUsedEdges.clear();

usedVertexes.clear();

notUsedVertexes.clear();

minimumSpanningTree.clear();

isConnectivityChecked = **false**;

isInitializedNotUsedLists = **false**;

graph.clear();

}

**public** void addEdge(Edge edge) **throws** **Exception** {

**if** (!graph.addEdge(edge)) {

LOGGER.log(**Level**.WARNING, "Try to add edge: " + edge.toString());

**throw** **new** **Exception**("edge already exist");

}

}

**public** **List**<Edge> result() **throws** **Exception** {

initialize();

**if** (!graphConnectivityCheck()) {

**throw** **new** **Exception**("graph isn't connected");

}

LOGGER.log(**Level**.SEVERE, "Called function result, to find mst");

**while** (notUsedVertexes.size() > 0) {

nextEdgeAtMst();

}

**return** minimumSpanningTree;

}

**public** Edge nextEdgeAtMst() **throws** **Exception** {

initialize();

LOGGER.log(**Level**.INFO, "Called function to find next minimum edge for mst");

**if** (!graphConnectivityCheck()){

LOGGER.log(**Level**.WARNING, "Graph isn't connected");

**throw** **new** **Exception**("graph isn't connected");

}

var result = **new** Edge(**new** Vertex("a"), **new** Vertex("a"), -1);

**if** (notUsedVertexes.size() > 0) {

int minE = -1;

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

**if** (((usedVertexes.indexOf(notUsedEdges.get(i).getBegin()) != -1) &&

(notUsedVertexes.indexOf(notUsedEdges.get(i).getEnd()) != -1)) ||

((usedVertexes.indexOf(notUsedEdges.get(i).getEnd()) != -1) &&

(notUsedVertexes.indexOf(notUsedEdges.get(i).getBegin()) != -1))) {

**if** (minE != -1) {

**if** (notUsedEdges.get(i).getWeight() < notUsedEdges.get(minE).getWeight())

minE = i;

} **else**

minE = i;

}

}

**if** (usedVertexes.indexOf(notUsedEdges.get(minE).getBegin()) != -1) {

usedVertexes.add(notUsedEdges.get(minE).getEnd());

notUsedVertexes.remove(notUsedEdges.get(minE).getEnd());

} **else** {

usedVertexes.add(notUsedEdges.get(minE).getBegin());

notUsedVertexes.remove(notUsedEdges.get(minE).getBegin());

}

minimumSpanningTree.add(notUsedEdges.get(minE));

result = notUsedEdges.get(minE);

LOGGER.log(**Level**.INFO, "Add edge: " + result.toString() + " to MST");

notUsedEdges.remove(minE);

} **else** {

**throw** **new** **Exception**("all vertexes added");

}

**return** result;

}

**@Override**

**public** void updateNotify(boolean isOn) {

LOGGER.setLevel(**Level**.OFF);

}

}

**package** org.apd.algorithm;

**public** **interface** Observer {

void updateNotify(boolean isLoggerOn);

}

**package** org.apd.algorithm;

**import** java.util.ArrayList;

**public** **class** ObserverManager {

**private** **ArrayList**<**Observer**> observers= **new** **ArrayList**<>();

**public** void addObserver(**Observer** observer){

observers.add(observer);

}

**public** void notify(boolean isLoggerOn){

**for**(**Observer** observer : observers){

observer.updateNotify(isLoggerOn);

}

}

**Код классов реализующих GUI**

**package** org.apd;

**import** javafx.application.Application;

**import** javafx.beans.value.ChangeListener;

**import** javafx.beans.value.ObservableValue;

**import** javafx.stage.Stage;

**import** org.apd.algorithm.AlgorithmAPD;

**import** org.apd.algorithm.Graph;

**import** org.apd.algorithm.GraphReader;

**import** org.apd.algorithm.ObserverManager;

**import** org.apd.ui.AppUI;

**import** org.apd.ui.GraphUI;

**import** java.util.logging.Logger;

/\*\*

**\*** JavaFX App

 \*/

**public** **class** App **extends** Application {

**@Override**

**public** void start(Stage stage) {

//var javaVersion = SystemInfo.javaVersion();

//var javafxVersion = SystemInfo.javafxVersion();

var appUI = **new** AppUI();

var graphUI = **new** GraphUI(appUI.boxDraw);

var graph = **new** Graph(appUI.boxTxtLog);

var graphReader = **new** GraphReader(graph, appUI.boxTxtLog);

var apd = **new** AlgorithmAPD(graph, appUI.boxTxtLog);

var controller = **new** Controller(appUI, graphUI, graphReader, graph, apd);

var obsManager = **new** ObserverManager();

obsManager.addObserver(graph);

obsManager.addObserver(apd);

obsManager.addObserver(graphReader);

appUI.btnStart.setOnAction(actionEvent -> {

stage.setScene(appUI.sceneMain);

stage.setResizable(**true**);

});

appUI.btnEditGraph.setOnAction(actionEvent -> appUI.windowEdit.show());

**Logger**.getLogger("log");

appUI.btnOK.setOnAction(actionEvent -> appUI.windowEdit.close());

appUI.btnAddFromFile.setOnAction(actionEvent -> {

var file = appUI.windowAddFromFile.showOpenDialog(appUI.windowEdit);

controller.openFile(file);

});

appUI.btnAddE.setOnAction(actionEvent -> controller.addEdge());

appUI.btnDeleteE.setOnAction(actionEvent -> controller.deleteEdge());

appUI.btnDeleteV.setOnAction(actionEvent -> controller.deleteVertex());

appUI.btnClear.setOnAction(actionEvent -> controller.clear());

appUI.btnStepForward.setOnAction(actionEvent -> controller.nextStep());

appUI.btnResult.setOnAction(actionEvent -> controller.result());

appUI.btnSaveResult.setOnAction(actionEvent -> {

var dir = appUI.windowSaveResult.showDialog(stage);

controller.saveResult(dir);

});

appUI.checkLog.setOnAction(actionEvent -> obsManager.notify(appUI.checkLog.isSelected()));

appUI.boxDraw.widthProperty().addListener(**new** ChangeListener<**Number**>() {

**@Override**

**public** void changed(ObservableValue<? **extends** **Number**> observableValue, **Number** number, **Number** t1) {

graphUI.moveGraph();

}

});

appUI.boxDraw.heightProperty().addListener(**new** ChangeListener<**Number**>() {

**@Override**

**public** void changed(ObservableValue<? **extends** **Number**> observableValue, **Number** number, **Number** t1) {

graphUI.moveGraph();

}

});

stage.setTitle("Prim Beta");

stage.getIcons().add(appUI.imgAppIcon);

stage.setResizable(**false**);

stage.setScene(appUI.sceneCover);

stage.centerOnScreen();

stage.show();

}

**public** **static** void main(**String**[] args) {

launch();

}

}

**package** org.apd;

**import** javafx.scene.control.TextArea;

**import** java.util.logging.FileHandler;

**import** java.util.logging.Handler;

**import** java.util.logging.LogRecord;

**public** **class** ApplicationHandler **extends** Handler {

**private** TextArea textArea;

**public** ApplicationHandler(TextArea textArea) {

**this**.textArea = textArea;

}

**@Override**

**public** void publish(**LogRecord** logRecord) {

textArea.appendText(logRecord.getLoggerName() + " " + logRecord.getSourceMethodName() + "\n" + logRecord.getLevel() + ":" + logRecord.getMessage() + "\n");

}

**@Override**

**public** void flush() {

}

**@Override**

**public** void close() **throws** **SecurityException** {

}

}

**package** org.apd;

**import** javafx.collections.FXCollections;

**import** javafx.scene.SnapshotParameters;

**import** org.apd.algorithm.**\***;

**import** org.apd.ui.AppUI;

**import** org.apd.ui.GraphUI;

**import** java.io.File;

**import** java.util.List;

**import** java.util.Scanner;

**import** javax.imageio.ImageIO;

**import** javafx.embed.swing.SwingFXUtils;

**public** **class** Controller {

**private** **final** AppUI appUI;

**private** **final** GraphUI graphUI;

**private** **final** Graph graph;

**private** **final** GraphReader graphReader;

**private** **final** AlgorithmAPD apd;

Controller(AppUI aUI, GraphUI gUI, GraphReader gr, Graph g, AlgorithmAPD a){

appUI = aUI;

graphUI = gUI;

graphReader = gr;

graph = g;

apd = a;

}

**public** void openFile(**File** file){

**try** {

graphReader.readGraphFromFile(file);

update();

enableBtn();

} **catch** (**Exception** e) {

appUI.windowError.setHeaderText(e.getMessage());

appUI.windowError.show();

}

}

**public** void addEdge(){

**try** {

var scanner = **new** **Scanner**(appUI.boxTxtAddE.getText()).useDelimiter(**System**.getProperty("line.separator"));

**String**[] curLine = scanner.nextLine().split(" ");

Edge edge = **new** Edge(**new** Vertex(curLine[0]), **new** Vertex(curLine[1]), **Integer**.parseInt(curLine[2]));

graphReader.addEdge(edge);

update();

enableBtn();

} **catch** (**Exception** e){

appUI.windowError.setHeaderText(e.getMessage());

appUI.windowError.show();

}

}

**public** void deleteEdge(){

**try** {

Edge item = (Edge) appUI.boxTableAllGraph.getSelectionModel().getSelectedItem();

graphReader.removeEdge(item);

update();

**if** (graph.getVertexesList().size() == 0){

clear();

}

} **catch** (**Exception** e){

appUI.windowError.setHeaderText(e.getMessage());

appUI.windowError.show();

}

}

**public** void deleteVertex(){

**try** {

var scanner = **new** **Scanner**(appUI.boxTxtAddE.getText()).useDelimiter(**System**.getProperty("line.separator"));

**String**[] curLine = scanner.nextLine().split(" ");

**for** (var str: curLine){

graphReader.removeVertex(**new** Vertex(**Character**.toString(str.charAt(0))));

}

update();

**if** (graph.getVertexesList().size() == 0){

clear();

}

} **catch** (**Exception** e){

appUI.windowError.setHeaderText(e.getMessage());

appUI.windowError.show();

}

}

**public** void clear(){

apd.clear();

update();

disableBtn();

appUI.btnEditGraph.setDisable(**false**);

}

**public** void nextStep(){

**try** {

var e = apd.nextEdgeAtMst();

graphUI.addToSpanning(e);

appUI.btnEditGraph.setDisable(**true**);

} **catch** (**Exception** e){

appUI.btnStepForward.setDisable(**true**);

appUI.btnResult.setDisable(**true**);

appUI.btnSaveResult.setDisable(**false**);

}

}

**public** void result(){

**try** {

**List**<Edge> es = apd.result();

**for** (var e: es){

graphUI.addToSpanning(e);

}

appUI.btnStepForward.setDisable(**true**);

appUI.btnResult.setDisable(**true**);

appUI.btnEditGraph.setDisable(**true**);

appUI.btnSaveResult.setDisable(**false**);

} **catch** (**Exception** e){

appUI.windowError.setHeaderText(e.getMessage());

appUI.windowError.show();

}

}

**public** void saveResult(**File** dir){

**try** {

var snapshot = appUI.boxDraw.snapshot(**new** SnapshotParameters(), **null**);

var file = **new** **File**(dir.getPath() + "/result.png");

ImageIO.write(SwingFXUtils.fromFXImage(snapshot, **null**), "png", file);

} **catch** (**Exception** e){

appUI.windowError.setHeaderText(e.getMessage());

appUI.windowError.show();

}

}

**private** void enableBtn(){

**if** (graph.getEdgesList().size() != 0){

appUI.btnDeleteE.setDisable(**false**);

appUI.btnClear.setDisable(**false**);

appUI.btnDeleteV.setDisable(**false**);

appUI.btnStepForward.setDisable(**false**);

appUI.btnResult.setDisable(**false**);

}

}

**private** void disableBtn(){

**if** (graph.getEdgesList().size() == 0){

appUI.btnDeleteE.setDisable(**true**);

appUI.btnDeleteV.setDisable(**true**);

appUI.btnStepForward.setDisable(**true**);

appUI.btnSaveResult.setDisable(**true**);

appUI.btnResult.setDisable(**true**);

appUI.btnClear.setDisable(**true**);

}

}

**private** void updateTableGraph(){

var edges = graph.getEdgesList();

var list = FXCollections.observableArrayList(edges);

appUI.boxTableAllGraph.setItems(list);

}

**private** void updateBoxDraw(){

graphUI.graphToUI(graph);

graphUI.drawGraph();

}

**private** void update(){

updateTableGraph();

updateBoxDraw();

}

}

**package** org.apd.ui;

**import** javafx.scene.Scene;

**import** javafx.scene.control.**\***;

**import** javafx.scene.control.cell.PropertyValueFactory;

**import** javafx.scene.image.Image;

**import** javafx.scene.image.ImageView;

**import** javafx.scene.layout.**\***;

**import** javafx.stage.DirectoryChooser;

**import** javafx.stage.FileChooser;

**import** javafx.stage.Modality;

**import** javafx.stage.Stage;

**import** org.apd.algorithm.Edge;

**import** org.apd.algorithm.Graph;

**public** **class** AppUI {

**private** **static** **final** **class** AppButton **extends** Button{

AppButton(**String** txt, **String** color){

**super**(txt);

setStyle("-fx-border-radius: 0; -fx-text-fill: #fff; -fx-cursor: hand; -fx-font-weight: normal; -fx-background-color: " + color);

}

AppButton(**String** txt, **String** color, boolean disable){

**this**(txt, color);

setDisable(disable);

}

}

**public** **final** Image imgAppIcon;

**public** **final** Image imgAppCover;

**public** **static** **final** **String** colorRed = "#EF2A0F";

**public** **static** **final** **String** colorOrange = "#EF6D0F";

**public** **static** **final** **String** colorGreen = "#39D276";

**public** **static** **final** **String** colorCyan = "#26A6EE";

**public** **static** **final** **String** colorBlue = "#331A79";

**public** **final** Button btnEditGraph;

**public** **final** Button btnStepForward;

**public** **final** Button btnResult;

**public** **final** Button btnSaveResult;

**public** **final** Button btnStart;

**public** **final** Button btnAddE;

**public** **final** Button btnAddFromFile;

**public** **final** Button btnDeleteV;

**public** **final** Button btnDeleteE;

**public** **final** Button btnClear;

**public** **final** Button btnOK;

**public** **final** CheckBox checkLog;

**public** **final** Pane boxDraw;

**public** **final** TextArea boxTxtLog;

**public** **final** TextField boxTxtAddE;

**public** **final** TableView boxTableAllGraph;

**public** **final** Scene sceneCover;

**public** **final** Scene sceneMain;

**public** **final** Scene sceneEdit;

**public** **final** Stage windowEdit;

**public** **final** DirectoryChooser windowSaveResult;

**public** **final** FileChooser windowAddFromFile;

**public** **final** Alert windowError;

**public** AppUI(){

imgAppIcon = **new** Image("file:src/img/icon-64.png");

imgAppCover = **new** Image("file:src/img/cover.png");

btnEditGraph = **new** AppButton("Edit graph", colorOrange);

btnStepForward = **new** AppButton("Step forward", colorCyan, **true**);

btnResult = **new** AppButton("Result", colorCyan, **true**);

btnSaveResult = **new** AppButton("Save result in file", colorGreen, **true**);

btnStart = **new** AppButton("Start", colorBlue);

btnAddE = **new** AppButton("Add edge", colorCyan);

btnAddFromFile = **new** AppButton("Upload data from file", colorCyan);

btnDeleteE = **new** AppButton("Delete edge", colorOrange, **true**);

btnDeleteV = **new** AppButton("Delete vertex", colorOrange, **true**);

btnClear = **new** AppButton("Clear all", colorRed, **true**);

btnOK = **new** AppButton("OK", colorGreen);

checkLog = **new** CheckBox("Logging");

checkLog.setSelected(**true**);

boxDraw = **new** Pane();

boxDraw.setStyle("-fx-background-color: #999; -fx-max-width: 10000px; -fx-pref-width: 1px; -fx-pref-height: 1px");

boxTxtLog = **new** TextArea();

boxTxtLog.editableProperty().setValue(**false**);

boxTxtLog.setStyle("-fx-background-color: #ddd; -fx-text-fill: #000; -fx-max-width: 400px; -fx-highlight-fill: #ddd");

boxTxtAddE = **new** TextField("Add edge or delete vertexes");

boxTxtAddE.setMaxWidth(10000.0);

HBox.setHgrow(boxTxtAddE, Priority.ALWAYS);

boxTableAllGraph = **new** TableView<Graph>();

var vertex1 = **new** TableColumn<Edge, **Character**>("Vertex 1");

vertex1.setCellValueFactory(**new** PropertyValueFactory<>("begin"));

var vertex2 = **new** TableColumn<Edge, **Character**>("Vertex 2");

vertex2.setCellValueFactory(**new** PropertyValueFactory<>("end"));

var weight = **new** TableColumn<Edge, **Integer**>("Weight");

weight.setCellValueFactory(**new** PropertyValueFactory<>("weight"));

boxTableAllGraph.getColumns().addAll(vertex1, vertex2, weight);

boxTableAllGraph.setColumnResizePolicy(TableView.CONSTRAINED\_RESIZE\_POLICY);

boxTableAllGraph.setMaxWidth(10000.0);

VBox.setVgrow(boxTableAllGraph, Priority.ALWAYS);

//Components of Main window

var boxCover = **new** StackPane(**new** ImageView(imgAppCover), btnStart);

var boxForCheckLog = **new** BorderPane();

boxForCheckLog.setCenter(checkLog);

boxForCheckLog.heightProperty().add(btnResult.heightProperty().doubleValue());

var boxMainCommandsLeft = **new** HBox(btnStepForward, btnResult, boxForCheckLog);

boxMainCommandsLeft.setStyle("-fx-spacing: 5px");

var boxMainCommandsRight = **new** HBox(btnEditGraph, btnClear, btnSaveResult);

boxMainCommandsRight.setStyle("-fx-spacing: 5px");

var boxMainCommands = **new** BorderPane();

boxMainCommands.setLeft(boxMainCommandsLeft);

boxMainCommands.setRight(boxMainCommandsRight);

boxMainCommands.setStyle("-fx-pref-width: 720px; -fx-background-color: #fff; -fx-padding: 5px");

var boxMainDrawAndLog = **new** HBox(boxDraw, boxTxtLog);

HBox.setHgrow(boxDraw, Priority.ALWAYS);

HBox.setHgrow(boxTxtLog, Priority.ALWAYS);

var boxMain = **new** VBox(boxMainCommands, boxMainDrawAndLog);

boxMain.setStyle("-fx-min-height: 480px; -fx-min-width: 720px");

VBox.setVgrow(boxMainDrawAndLog, Priority.ALWAYS);

//Components of Edit Graph window

var boxEditCommandsTop = **new** HBox(boxTxtAddE, btnAddE, btnAddFromFile);

boxEditCommandsTop.setStyle("-fx-spacing: 5px; -fx-padding: 5px; -fx-background-color: #fff");

var boxEditCommandsBottomLeft = **new** HBox(btnDeleteV, btnDeleteE);

boxEditCommandsBottomLeft.setStyle("-fx-spacing: 5px");

var boxEditCommandsBottomRight = **new** HBox(btnOK);

boxEditCommandsBottomRight.setStyle("-fx-spacing: 5px");

var boxEditCommandsBottom = **new** BorderPane();

boxEditCommandsBottom.setStyle("-fx-padding: 5px; -fx-background-color: #fff");

boxEditCommandsBottom.setLeft(boxEditCommandsBottomLeft);

boxEditCommandsBottom.setRight(boxEditCommandsBottomRight);

var boxEdit = **new** VBox(boxEditCommandsTop, boxTableAllGraph, boxEditCommandsBottom);

boxEdit.setStyle("-fx-min-height: 480px; -fx-min-width: 720px");

sceneCover = **new** Scene(boxCover);

sceneMain = **new** Scene(boxMain);

sceneEdit = **new** Scene(boxEdit);

windowEdit = **new** Stage();

windowEdit.setTitle("Edit");

windowEdit.getIcons().add(imgAppIcon);

windowEdit.setScene(sceneEdit);

windowEdit.initModality(Modality.APPLICATION\_MODAL);

windowAddFromFile = **new** FileChooser();

windowAddFromFile.setTitle("Select Text file with edges");

windowAddFromFile.getExtensionFilters().add(**new** FileChooser.ExtensionFilter("Text", "\*.txt"));

windowSaveResult = **new** DirectoryChooser();

windowSaveResult.setTitle("Select directory to save result");

windowError = **new** Alert(Alert.AlertType.ERROR);

}

}

**package** org.apd.ui;

**import** javafx.scene.layout.Pane;

**import** javafx.scene.paint.Color;

**import** javafx.scene.shape.Circle;

**import** javafx.scene.shape.Line;

**import** javafx.scene.text.Text;

**import** org.apd.algorithm.Edge;

**import** org.apd.algorithm.Graph;

**import** org.apd.algorithm.Vertex;

**import** java.util.ArrayList;

**import** java.util.LinkedList;

**public** **class** GraphUI {

**private** **class** GraphV **extends** Circle{

**private** **final** **Text** name;

GraphV(double centerX, double centerY, Vertex name){

**super**(centerX, centerY,6.0);

**this**.name = **new** **Text**("" + name.toString());

**this**.name.yProperty().set(centerY - 10.0);

**this**.name.xProperty().set(centerX);

**this**.name.setStyle("-fx-font-weight: bold; -fx-text-fill: #fff");

setStyle("-fx-stroke: #fff; -fx-stroke-width: 2px; -fx-fill: " + AppUI.colorOrange);

}

}

**private** **class** GraphE **extends** Line {

**private** **final** GraphV v1;

**private** **final** GraphV v2;

**private** **final** **Text** weight;

GraphE(int weight, GraphV v1, GraphV v2){

**super**(v1.getCenterX(), v1.getCenterY(), v2.getCenterX(), v2.getCenterY());

**this**.weight = **new** **Text**("" + weight);

**this**.weight.yProperty().set((v1.getCenterY() + v2.getCenterY()) / 2.0);

**this**.weight.xProperty().set((v2.getCenterX() + v1.getCenterX()) / 2.0);

**this**.weight.setStyle("-fx-font-weight: bold");

**this**.v1 = v1;

**this**.v2 = v2;

setStyle("-fx-stroke: #fff; -fx-stroke-width: 2px");

}

}

**private** **final** **LinkedList**<GraphV> graphVertixes;

**private** **final** **ArrayList**<GraphE> graphEdges;

**private** **final** Pane boxDraw;

**private** double centerX;

**private** double centerY;

**private** double radius;

**public** GraphUI(Pane boxDraw){

graphEdges = **new** **ArrayList**<>();

graphVertixes = **new** **LinkedList**<>();

**this**.boxDraw = boxDraw;

update();

}

**public** void graphToUI(Graph g){

update();

clear();

var edges = g.getEdgesList();

var vs = g.getVertexesList();

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

double x = centerX + radius \* **Math**.cos(2 \* **Math**.PI \* i / vs.size());

double y = centerY + radius \* **Math**.sin(2 \* **Math**.PI \* i / vs.size());

var v = **new** GraphV(x, y, vs.get(i));

graphVertixes.add(v);

}

**for** (org.apd.algorithm.Edge edge : edges) {

int indexV0 = vs.indexOf(edge.getBegin());

int indexV1 = vs.indexOf(edge.getEnd());

var e = **new** GraphE(edge.getWeight(), graphVertixes.get(indexV0), graphVertixes.get(indexV1));

graphEdges.add(e);

}

}

**public** void addToSpanning(Edge e){

**for** (var edge: graphEdges){

**if** (e.getBegin().equals(**new** Vertex(**Character**.toString(edge.v1.name.getText().charAt(0)))) && e.getEnd().equals(**new** Vertex(**Character**.toString(edge.v2.name.getText().charAt(0))))){

edge.setStroke(Color.web(AppUI.colorGreen));

edge.v1.setFill(Color.web(AppUI.colorGreen));

edge.v2.setFill(Color.web(AppUI.colorGreen));

}

}

}

**public** void drawGraph(){

**for** (GraphE graphEdge : graphEdges) {

boxDraw.getChildren().addAll(graphEdge, graphEdge.weight);

}

**for** (GraphV graphVertix : graphVertixes) {

boxDraw.getChildren().addAll(graphVertix, graphVertix.name);

}

}

**public** void moveGraph(){

update();

updateCoordinates();

}

**private** void updateCoordinates(){

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

double x = centerX + radius \* **Math**.cos(2 \* **Math**.PI \* i / graphVertixes.size());

double y = centerY + radius \* **Math**.sin(2 \* **Math**.PI \* i / graphVertixes.size());

graphVertixes.get(i).setCenterX(x);

graphVertixes.get(i).setCenterY(y);

graphVertixes.get(i).name.xProperty().set(x);

graphVertixes.get(i).name.yProperty().set(y - 10.0);

}

**for** (var e: graphEdges){

e.setStartX(e.v1.getCenterX());

e.setEndX(e.v2.getCenterX());

e.setStartY(e.v1.getCenterY());

e.setEndY(e.v2.getCenterY());

e.weight.xProperty().set((e.v1.getCenterX() + e.v2.getCenterX()) / 2.0);

e.weight.yProperty().set((e.v1.getCenterY() + e.v2.getCenterY()) / 2.0);

}

}

**private** void update(){

centerX = boxDraw.getWidth() / 2.0;

centerY = boxDraw.getHeight() / 2.0;

radius = centerX < centerY ? centerX - 12 : centerY - 12;

}

**private** void clear(){

graphVertixes.clear();

graphEdges.clear();

boxDraw.getChildren().clear();

}

}

**Код классов тестирования**

**package** org.apd.algorithm;

**import** javafx.scene.control.TextArea;

**import** org.junit.After;

**import** org.junit.Assert;

**import** org.junit.Before;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Parameterized;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.Arrays;

**import** java.util.List;

**import** **static** org.junit.Assert.**\***;

**@RunWith(Parameterized.class)**

**public** **class** AlgorithmAPDParametriseTest {

**static** **final** **String** badConTestPath ="src/test/java/org/apd/algorithm/TestResources/AlgorithmAPDTests/BadConnectivity/test";

**static** **final** **String** correctConTestPath = "src/test/java/org/apd/algorithm/TestResources/AlgorithmAPDTests/CorrectConnectivity/test";

**static** **final** **String** graphsForApdPath = "src/test/java/org/apd/algorithm/TestResources/AlgorithmAPDTests/GraphsForApd/test";

**static** **final** **String** testType = ".txt";

**static** **final** int countTest = 10;

**private** **File** fileBadConnectivity;

**private** **File** fileCorrectConnectivity;

**private** **File** fileGraphAPD;

**private** AlgorithmAPD jpd;

**private** GraphReader graphReader;

**private** Graph graph;

**private** int resultAPDalgo;

/\*\*

**\*** Конструктор параметризованного теста класса **<code>**AlgorithmAPD**</code>**

**\*** **@param** fileBadCon **-** файл с несвязным графом.

**\*** **@param** fileCorrCon **-** файл со связным (корректным) графом.

**\*** **@param** fileGraph **-** файл с графом для вычисления остова.

**\*** **@param** resultAPD **-** результат остова для текущего теста.

     \*/

**public** AlgorithmAPDParametriseTest(**File** fileBadCon, **File** fileCorrCon, **File** fileGraph, int resultAPD){

fileBadConnectivity = fileBadCon;

fileCorrectConnectivity = fileCorrCon;

fileGraphAPD = fileGraph;

resultAPDalgo = resultAPD;

}

**@Parameterized.Parameters(name = "{index}:Test")**

**public** **static** **Iterable**<**Object**[]> dataForTest() {

int [] resultTests = **new** int[]{150, 9, 5, 75, 37, 9, 17, 172, 34, 25};

**Object**[][] data = **new** **Object**[countTest][];

**for** (int i = 0; i < countTest; i++){

data[i] = **new** **Object**[]{

**new** **File**(badConTestPath+(i+1)+testType),

**new** **File**(correctConTestPath + (i+1) + testType),

**new** **File**(graphsForApdPath + (i+1) + testType),

resultTests[i]

};

}

**return** **Arrays**.asList(data);

}

/\*\*

**\*** Метод выполняющийся перед каждым тестовым методом.

     \*/

**@Before**

**public** void setUp(){

graph = **new** Graph();

jpd = **new** AlgorithmAPD(graph);

graphReader = **new** GraphReader(graph);

}

/\*\*

**\*** Метод**,** очищающий данные после каждого тестового метода.

     \*/

**@After**

**public** void afterMethod() {

jpd.clear();

graph.clear();

}

/\*\*

**\*** Проверка на связность корректных графов. **<br>**(Алгоритм Прима работает только со связанными графами).

**\*** **@**result Метод должен проверить все переданные файлы с корректными графами

**\*** и закончить работу без ошибок и исключений.

     \*/

**@Test**

**public** void corConnectivityGraphCheck() {

**try** {

graphReader.readGraphFromFile(fileCorrectConnectivity);

} **catch** (**Exception** e){

**if** (e **instanceof** **FileNotFoundException**)

**System**.out.println("Test correctConnectivity fall because of FileNotException.");

**System**.out.println("Test correctConnectivity for file " + fileCorrectConnectivity.getName() + " fall");

e.printStackTrace();

Assert.fail();

}

**try** {

jpd.result();

} **catch** (**Exception** e) {

Assert.fail("Error, graph correct, but method result another. File " + fileCorrectConnectivity.getName());

e.printStackTrace();

}

}

/\*\*

**\*** Проверка на связность некорректных графов. **<br>**(Алгоритм Прима работает только со связанными графами).

**\*** **@**result Метод должен для каждого файла возвращать исключение **<code>**Exception**</code>**.

**\*** **@throws** Exception

     \*/

**@Test(expected = Exception.class)**

**public** void badConnectivityGraphCheck() **throws** **Exception** {

**try** {

graphReader.readGraphFromFile(fileBadConnectivity);

} **catch** (**Exception** e) {

**if** (e **instanceof** **FileNotFoundException**)

**System**.out.println("Test badConnectivity fall because of FileNotException.");

**System**.out.println("Test badConnectivity for file " + fileBadConnectivity.getName() + " fall");

Assert.fail();

e.printStackTrace();

}

jpd.result();

}

/\*\*

**\*** 'ычисление минимального остова.

**\*** **@**result Метод проверяет работу метода **<code>**result()**</code>** для каждого тестового файла.

**\*** "олжен получить правильный результат и закончиться без исключений.

     \*/

**@Test**

**public** void resultTest(){

**File** file;

int actual = 0;

**try** {

graphReader.readGraphFromFile(fileGraphAPD);

} **catch** (**Exception** e) {

**if** (e **instanceof** **FileNotFoundException**)

**System**.out.println("Test resultTest fall because of FileNotException.");

**System**.out.println("Test resultTest for file" + fileGraphAPD.getName() + " fall.");

e.printStackTrace();

Assert.fail();

}

**try** {

**List**<Edge> list = jpd.result();

actual = 0;

**for** (Edge edge: list){

actual += edge.getWeight();

}

} **catch** (**Exception** e) {

**System**.out.println("Correct file, but resultTest for file" + fileGraphAPD.getName() + " fall.");

e.printStackTrace();

Assert.fail();

}

Assert.assertEquals("Неверный результат.",resultAPDalgo, actual);

}

}

**package** org.apd.algorithm;

**import** org.junit.**\***;

**import** **static** org.junit.Assert.**\***;

/\*\*

**\*** Класс для тестирования класса **<b>**AlgorithmAPD**</b>**

**\*** **@author** **Ilya**

 \*/

**public** **class** AlgorithmAPDTest {

**private** AlgorithmAPD jpd;

/\*\*

**\*** Метод инициализации**,** выполняющийся до любого метода.

     \*/

**@BeforeClass**

**public** **static** void globalSetUp(){

}

/\*\*

**\*** Метод для подготовки данных**,** выполняется перед каждым тестовым методом.

     \*/

**@Before**

**public** void setUp(){

jpd = **new** AlgorithmAPD(**new** Graph());

}

/\*\*

**\*** Метод**,** очищающий данные после каждого тестового метода.

     \*/

**@After**

**public** void afterMethod() {

jpd.clear();

}

/\*\*

**\*** обавление корректного ребра.

**\*** **@**result Ребро будет добавлено без каких либо ошибок и икслючений.

     \*/

**@Test**

**public** void addEdge() {

Edge edge = **new** Edge(**new** Vertex("s"), **new** Vertex("e"), 10);

**try** {

jpd.addEdge(edge);

} **catch** (**Exception** e) {

Assert.fail("Ошибка при корректном добавлении.");

e.printStackTrace();

}

}

/\*\*

**\*** обавление идентичных ребер в граф.

**\*** **@**result Метод добавления ребра **<code>**addEdge()**</code>** должен вернуть исключение **<code>**Exception**</code>**.

**\*** **@throws** Exception

     \*/

**@Test(expected = Exception.class)**

**public** void addSameEdge() **throws** **Exception** {

Edge edge\_first = **new** Edge(**new** Vertex("s"), **new** Vertex("e"), 10);

Edge edge\_second = **new** Edge(**new** Vertex("s"), **new** Vertex("e"), 1);

Edge edge\_third = **new** Edge(**new** Vertex("e"), **new** Vertex("s"), 23);

**try** {

jpd.addEdge(edge\_first);

} **catch** (**Exception** e) {

Assert.fail("Ошибка при корректном добавлении.");

e.printStackTrace();

}

jpd.addEdge(edge\_second);

jpd.addEdge(edge\_third);

}

/\*

    @Test

    public void removeVertex() {

    }

    \*/

}

**package** org.apd.algorithm;

**import** org.junit.After;

**import** org.junit.Assert;

**import** org.junit.Before;

**import** org.junit.Test;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** **static** org.junit.Assert.**\***;

**public** **class** GraphReaderTest {

**private** GraphReader graphReader;

**private** Graph graph;

**@Before**

**public** void setUp() **throws** **Exception** {

graph = **new** Graph();

graphReader = **new** GraphReader(graph);

}

**@After**

**public** void tearDown() **throws** **Exception** {

graph.clear();

}

/\*\*

**\*** обавление корректного ребра.

**\*** **@**result Ребро будет добавлено без каких либо ошибок и икслючений.

     \*/

**@Test**

**public** void addEdge() {

int expected = graph.getEdgesList().size() + 1;

Edge edge = **new** Edge(**new** Vertex("l"), **new** Vertex("v"), 10);

**try** {

graph.addEdge(edge);

} **catch** (**Exception** e) {

Assert.fail("Ошибка при корректном добавлении.");

e.printStackTrace();

}

Assert.assertEquals(expected, graph.getEdgesList().size());

}

/\*\*

**\*** обавление идентичных ребер в граф.

**\*** **@**result Метод добавления ребра **<code>**addEdge()**</code>** должен вернуть исключение **<code>**Exception**</code>**.

**\*** **@throws** Exception

     \*/

**@Test(expected = Exception.class)**

**public** void addSameEdge() **throws** **Exception** {

int expected = graph.getEdgesList().size() + 1;

Vertex start = **new** Vertex("st");

Vertex end = **new** Vertex("end");

Edge edge\_first = **new** Edge(start,end, 10);

Edge edge\_second = **new** Edge(start, end, 1);

Edge edge\_third = **new** Edge(end, start, 23);

**try** {

graphReader.addEdge(edge\_first);

} **catch** (**Exception** e) {

Assert.fail("Ошибка при корректном добавлении.");

e.printStackTrace();

}

graphReader.addEdge(edge\_second);

graphReader.addEdge(edge\_third);

Assert.assertEquals(expected, graph.getEdgesList().size());

}

/\*\*

**\*** Попытка считывания графа из несуществующего файла.

**\*** **@**result Метод должен для некорректного файла возвращать исключение **<code>**FileNotFoundException**</code>**.

**\*** **@throws** FileNotFoundException

     \*/

**@Test(expected = Exception.class)**

**public** void readGraphFromNotExistFile() **throws** **Exception** {

**File** file = **new** **File**("notexist.txt");

graphReader.readGraphFromFile(file);

}

/\*\*

**\*** Попытка считывания графа из корректного файла.

**\*** **@**result Метод должен завершить работу без ошибок и исключений.

     \*/

**@Test**

**public** void readGraphFromExistFile() {

**File** file = **new** **File**(AlgorithmAPDParametriseTest.graphsForApdPath+1+AlgorithmAPDParametriseTest.testType);

**try** {

graphReader.readGraphFromFile(file);

} **catch** (**Exception** e) {

Assert.fail("Ошибка открытия существующего файла.");

e.printStackTrace();

Assert.fail();

}

}

}

**package** org.apd.algorithm;

**import** org.junit.**\***;

**import** **static** org.junit.Assert.**\***;

**public** **class** GraphTest {

**private** Graph graph;

**@Before**

**public** void setUp() **throws** **Exception** {

graph = **new** Graph();

}

**@After**

**public** void tearDown() **throws** **Exception** {

graph.clear();

}

/\*\*

**\*** обавление корректного ребра в граф.

**\*** **@**result Метод должен завершить работу без ошибок и исключений.

     \*/

**@Test**

**public** void addEdge() {

Edge edge = **new** Edge(**new** Vertex("s"), **new** Vertex("e"), 10);

boolean actual;

actual = graph.addEdge(edge);

Assert.assertTrue("Error! Не добавлено корректное ребро.", actual);

}

/\*\*

**\*** обавление идентичных ребер в граф.

**\*** **@**result Метод должен закончить работу**,** вернув **<code>**false**</code>**.

     \*/

**@Test**

**public** void addSameEdges(){

Edge edge\_first = **new** Edge(**new** Vertex("s"), **new** Vertex("e"), 10);

Edge edge\_second = **new** Edge(**new** Vertex("s"), **new** Vertex("e"), 1);

Edge edge\_third = **new** Edge(**new** Vertex("e"), **new** Vertex("s"), 23);

boolean actual = graph.addEdge(edge\_first);

Assert.assertTrue("Ошибка при добавлении корректного ребра.", actual);

actual = graph.addEdge(edge\_second);

Assert.assertFalse("Ошибка! обавлено одинаковое ребро.", actual)

Assert.assertFalse("Ошибка! обавлено одинаковое ребро.", graph.addEdge(edge\_third))

**@Test**

**public** void clearTest(){

Vertex start1 = **new** Vertex("a"), start2 = **new** Vertex("b"),

end1 = **new** Vertex("b"), end2 = **new** Vertex("c");

Edge first = **new** Edge(start1, end1, 2);

Edge second = **new** Edge(start2, end2, 1);

graph.addEdge(first);

graph.addEdge(second);

graph.clear();

int actual = graph.getVertexesList().size();

Assert.assertEquals(0, actual);

actual = graph.getEdgesList().size();

Assert.assertEquals(0, actual);

}

/\*\*

**\*** Удаление вершины.

**\*** **@**result Метод должен удалить вершину и смежные ей ребра.

     \*/

**@Test**

**public** void removeVertex() **throws** **Exception** {

Vertex start1 = **new** Vertex("a"), start2 = **new** Vertex("b"),

end1 = **new** Vertex("b"), end2 = **new** Vertex("c");

Edge first = **new** Edge(start1, end1, 2);

Edge second = **new** Edge(start2, end2, 1);

graph.addEdge(first);

graph.addEdge(second);

int expectedVertex = graph.getVertexesList().size() - 1;

int expectedEdges = 0;

graph.removeVertex(start2);

int actual = graph.getVertexesList().size();

Assert.assertEquals(expectedVertex, actual);

actual = graph.getEdgesList().size();

Assert.assertEquals(expectedEdges, actual);

}

package org.apd.algorithm;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

**@RunWith(Suite.class)**

**@Suite.SuiteClasses({**

**AlgorithmAPDTest.class,**

**AlgorithmAPDParametriseTest.class,**

**GraphTest.class,**

**GraphReaderTest.class**

**})**

**public** **class** AutoTestSuite {

}