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

ОТЧЕТ

по учебной практике

Тема: Визуализация алгоритмов на языке Java

Студент гр. 8304 _____ Сергеев А.Д.

Студент гр. 8304 _____ Алтухов А.Д.

Студентка гр. 8381 _____ Звегинцева Е.Н.

Руководитель _____ Жангиров Т.Р.

Санкт-Петербург
2020

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Сергеев А.Д. группы 8304

Студент Алтухов А.Д. группы 8304

Студентка Звегинцева Е.Н. группы 8381

Тема практики: визуализация алгоритмов на языке Java

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: ЯПД.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: ??.07.2020

Дата защиты отчета: ??.07.2020

Студент	_____	Сергеев А.Д.
Студент	_____	Алтухов А.Д.
Студентка	_____	Звегинцева Е.Н.
Руководитель	_____	Жангиров Т.Р.

АННОТАЦИЯ

В ходе выполнения задания учебной практики была реализована программа, предназначенная для нахождения минимального остовного дерева. Поиск минимального остовного дерева осуществляется с использованием алгоритма Прима.

Разработанная программа детально показывает этапы работы алгоритма при построении минимального остовного дерева. Программа была написана на языке программирования Java, в среде разработки IntelliJ Idea.

SUMMARY

In the course of the assignment, a program was developed designed to find the minimum spanning tree. Search using the Prim algorithm. The developed program shows in detail the stages of the algorithm when building the minimum spanning tree. The program was written in the Java, in the IntelliJ Idea development environment.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Разработка программы будет вестись с использованием языка программирования Java и его возможностей по созданию GUI. Для написания программы используется интегрированная среда разработки IntelliJ IDEA.

Цели выполнения учебной практики:

1. Освоение среды программирования Java, особенностей и синтаксиса данного языка.

2. Изучение средств для реализации графического интерфейса

3. Получение таких навыков как:

- разработка программ на языке Java;
- работа с системой контроля версий и репозиториями;
- командная работа.

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Формулировка задания

Разработать программу, которая визуализирует алгоритм Прима на языке программирования Java.

1.2. Формальное определение и сложность алгоритма

Алгоритм Прима - алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Алгоритм впервые был открыт в 1930 году чешским математиком Войцехом Ярником, позже переоткрыт Робертом Примом в 1957 году, и, независимо от них, Э. Дейкстрой в 1959 году.

На вход алгоритма подаётся связный неориентированный граф. Для каждого ребра задаётся его стоимость.

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Результатом работы алгоритма является остовное дерево минимальной стоимости.

Сложность алгоритма Прима зависит от способа нахождения вершины v, а также способа хранения множества непосещённых вершин и способа обновления меток. В простейшем случае, когда для хранения величин d используется массив, время работы алгоритма есть O(V^2).

1.3. Реализуемый алгоритм

Каждой вершине из V сопоставим приоритет — минимальное известное расстояние от этой вершины до a(произвольной). Алгоритм работает пошагово

— на каждом шаге он «посещает» одну вершину и пытается уменьшать приоритеты. Работа алгоритма завершается, когда все вершины посещены.

Инициализация.

Метка самой вершины a полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от a до других вершин пока неизвестны. Все вершины графа помечаются как непосещённые.

Шаг алгоритма.

Если все вершины посещены, алгоритм завершается.

В противном случае, выбирается вершина с минимальным приоритетом из еще не посещенных.

Мы рассматриваем всевозможные маршруты, в которых u является предпоследним пунктом. Вершины, в которые ведут рёбра из u, назовём соседями этой вершины. Для каждого соседа вершины u, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки u и длины ребра, соединяющего u с этим соседом. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину u как посещённую и повторим шаг алгоритма.

1.4. Псевдокод

Обозначения:

- d[i] — расстояние от i-й вершины до построенного дерева
- p[i] — предок i-й вершины, то есть такая вершина u, (u,i) легчайшее из всех рёбер, соединяющее i с вершиной из построенного дерева.
- w(i,j) — вес ребра (i,j)
- Q — приоритетная очередь вершин графа, где ключ - d[i]
- T — множество ребер минимального остового дерева

$T \leftarrow \{\}$
Для каждой вершины $i \in V$
$d[i] \leftarrow \infty$
$p[i] \leftarrow nil$
$d[1] \leftarrow 0$

$Q \leftarrow V$
$v \leftarrow Extract.Min(Q)$

Пока Q не пуста
  Для каждой вершины u смежной с v
    Если $u \in Q$ и $w(v,u) < d[u]$
      $d[u] \leftarrow w(v,u)$
      $p[u] \leftarrow v$
  $v \leftarrow Extract.Min(Q)$
 $T \leftarrow T + (p[v], v)$

2. СПЕЦИФИКАЦИЯ

2.1. Требования к интерфейсу пользователя

Пользовательский интерфейс должен представлять собой диалоговое окно, содержащее набор кнопок, предназначенных для управления состоянием программы. Набросок GUI представлен на рис.1, выполнен в редакторе Figma.

Основное диалоговое окно должно состоять из:

- Рабочей области для построения графа.
- Области для вывода логов
- Кнопки «Справка», показывающей пользователю информацию о работе алгоритма и вводе данных.
- Кнопки «Сохранить», которая должна позволять пользователю сохранить созданный в рабочей области граф.
- Кнопки «Открыть», которая должна позволять пользователю загрузить граф из файла.
- Кнопки «Запустить алгоритм», которая применяет алгоритм к графу из рабочей области
- Кнопки «Вперед», которая должна отобразить следующую итерацию алгоритма.
- Кнопки «Назад», которая должна отобразить предыдущую итерацию алгоритма

Всплывающее диалоговое окно:

- Кнопки «Создать вершину», которая должна создать вершину графа в рабочей области.
- Кнопки «Соединить вершины», которая должна создать направленное ребро между двумя вершинами и задать его вес.

- Кнопки «Удалить», которая должна удалить выбранный пользователем элемент графа.

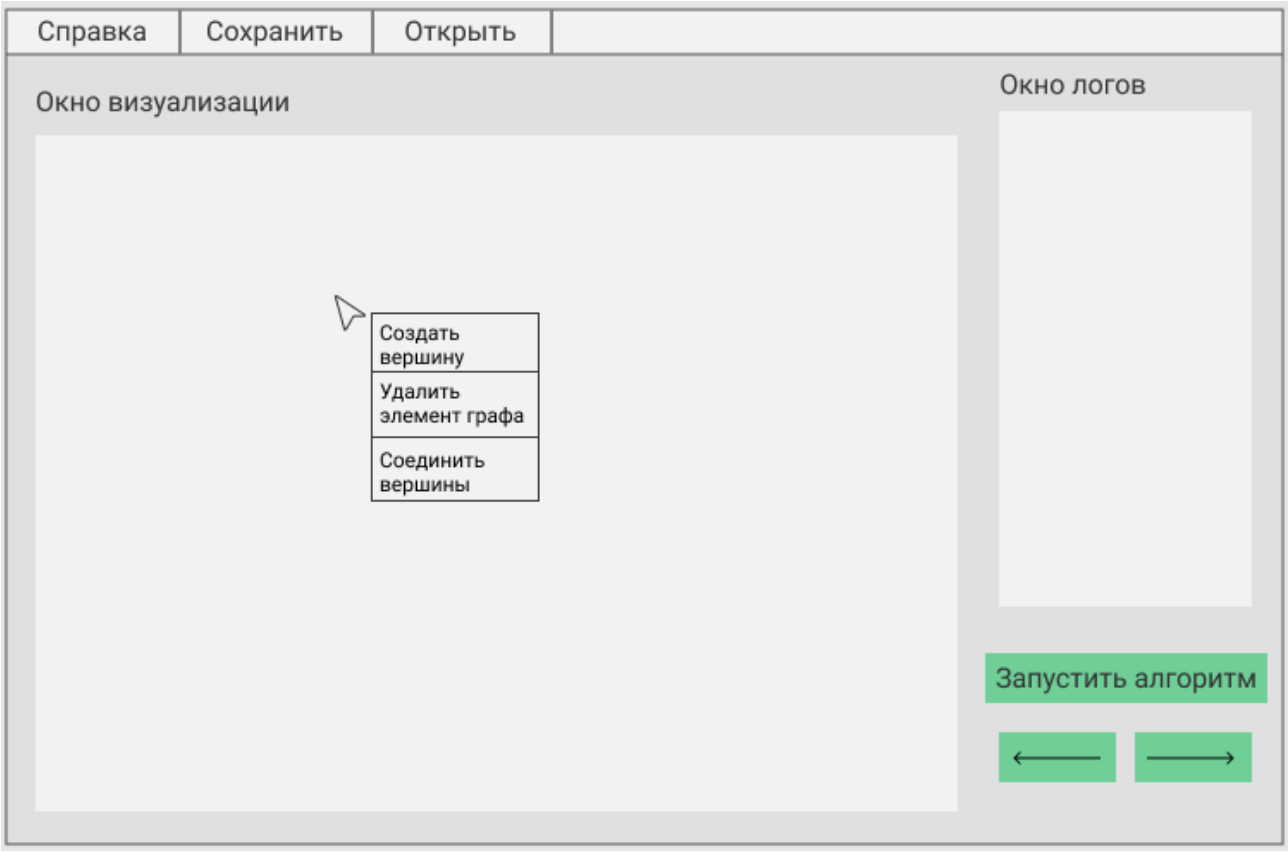

Рисунок 1 — Макет программы

2.2. Требования к вводу исходных данных

На вход алгоритму должен подаваться взвешенный неориентированный граф. Именем вершины могут быть числа. Область создания графа предоставляет возможность ввести данные с файла или сгенерировать. При ручном вводе происходит взаимодействие с графическим представлением графа с помощью мыши. Двойным кликом по области создается вершина с указанным именем, а через контекстное меню вершины можно будет удалить ее или провести ребро к другой вершине. При считывании с файла надо нажать соответствующую кнопку.

3. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

3.1. План разработки

План разработки программы разбит на следующие шаги:

1. Создание каркаса программы (04.07): к этому шагу будут созданы:
   - начальный этап отчета и модель GUI(Звегинцева);
   - алгоритм и uml-диаграмма(Сергеев и Алтухов);
2. Первая версия (07.07):
   - реализация классов алгоритма(Алтухов):
   - реализация классов GUI(Сергеев);
   - реализовано взаимодействие с вершинами для ручного ввода(Сергеев);
   - написание тестов к проекту(Звегинцева);
   - дополнение отчета (Звегинцева);
3. Вторая (финальная) версия (10.07):
   - реализация пошаговой работы программы(Алтухов);
   - реализация вывода дополнительного информационного текста при шагах алгоритма(Сергеев);
   - Исправление ошибок и недочетов проекта в коде, визуализации и отчете(команда);
   - Итоговое тестирование проекта и реализация автоматического тестирования(Звегинцева).

3.2. Распределение ролей в бригаде

Роли в бригаде распределены следующим образом:

- Сергеев Александр: лидер, фронтенд;
- Алтухов Александр: алгоритмист;
- Звегинцева Елизавета: тестировщик, документация;

4. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРТИМА

4.1 Структуры данных

Для создания графа были реализованы следующие классы: Ark, Node и соответственно Graph.

Класс Node реализует объект узла(вершины) графа. Он содержит поля:

- private static final long serialVersionUID = 3L - указание версии сериализованных данных;
- private String name – наименование узла;
- private LinkedList<Ark> arks = new LinkedList<Ark>() – двунаправленный список ребер;
- private boolean hidden – состояние узла в графе(при прохождении алгоритмом, может как остаться видимым, так и стать скрытым);

Класс Ark реализует объект ребра графа и содержит поля:

- private static final long serialVersionUID = 2L - указание версии сериализованных данных;
- private final Node start, end – вершины, лежащие на концах ребра;
- private final double weight – вес ребра;
- private boolean hidden - состояние ребра в графе(при прохождении алгоритмом, может как остаться видимым, так и стать скрытым);

Все приведенные выше данные можно получить, черед методы, реализованные в данных классах.

Класс Graph реализует объект самого графа и соответственно содержит поля:

- private HashMap<String, Node> nodes = new HashMap<String, Node>() – для хранения узлов графа;

- private LinkedList<Ark> arks = new LinkedList<Ark>() – для хранения ребер графа;

Для работы с ними были реализованы методы:

- addNode(Point2D position, String name) – добавление узла;
- deleteNode(String name) – удаление узла;
- addArk(Node start, Node end, int weight) – добавление ребра;
- deleteArk(Node start, Node end) – удаление ребра;
- isEmpty() – проверка на пустоту графа;
- getNodes() – вывод данных об узлах;
- getArks() – вывод данных о ребрах;

Сам алгоритм Ярника Прима Дейкстры реализован через методы класса PrimaAlgorithm, содержащего поля:

- private Graph graph – непосредственно сам граф для работы алгоритма;
- private ArrayList<Node> nodesForSearch = new ArrayList<Node>() – список непосещенных вершин графа;

Для удобства реализации дополнительных функций алгоритма были созданы классы History(осуществляет сохранение изменений в графе и помогает в реализации контроля шага алгоритма) и NodePlus.

- private Point2D position поле в классе NodePlus для сохранения координат узла, чтобы в дальнейшем можно было сохранять и открывать граф в той же позиции, которой он был задан.

4.2 Основные методы

Основные методы, необходимые для эффективной работы алгоритма Ярника Прима Дейкстры, были написаны в классе PrimaAlgorithm.

solve(Graph graph) – метод-каркас алгоритма, который принимает на вход изначальный граф и, вызывая последующие методы, проводит алгоритм. Для работы с графом его следует подготовить, т.е. скрыть все существующие ребра и вершины, оставив лишь вершину(которая выбирается случайно), для начала работы алгоритма, что и происходит в методе prepareGraph(Graph graph). Идея алгоритма ЯПД заключается в том, что мы проходим все вершины, через минимальные смежные ребра, проверяя все пути. Метод solveStep(Graph graph) работает пока остаются непосещенные вершины. Он находит ребро с минимальным весом, в еще не посещенную вершину и переходит туда, в случае если из этой вершины больше нет ребер, или они ведут в уже посещенные вершины и менее выгодны, чем существующие, то мы возвращаемся в предыдущую вершину с учетом посещенных.

5. ОСОБЕННОСТИ РЕАЛИЗАЦИИ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА

5.1 Структуры данных

При создании GUI была использована библиотека Swing.

Для визуализации графа был создан класс GraphShape, в котором хранится граф, список узлов, список ребер, позиция двигающейся мышки и двигающийся узел(при инициализации координаты (0;0), т.е. позиция верхнего угла).

Класс NodeShape был создан для визуализации узла. В нем хранится информация об узле и радиус окружности для рисования. При выводе в окно визуализации рисуется две окружности с единым центром(для реализации краев окружности). Для вывода надписи, в окружность вписывается квадрат, края которого и служат границами для нее.

Класс ArkShape отвечает за визуализацию ребер. Ребра строятся по 4м точкам и имеют форму ромба. Вычисления координат этих точек можно посмотреть непосредственно в коде.

Класс SavedGraph создан для реализации сохранения графа на устройстве.

Класс Filer создан для работы с файлами графов. В нем существуют данные поля-константы:

- public static final String PROPERTIES_FILE_EXTENSION = "properties" – расширение для библиотеки констант и локализации языков;
- public static final String GRAPH_FILE_EXTENSION = "sv" – расширения файла для записи графа
- public static final String SAMPLE_GRAPH = "GraphSample.sv" – название файла, содержащего граф, предоставляемый пользователю для примера алгоритма;

- public static final String BIPARTITE_GRAPH = "GraphBipartite.sv"– название файла, содержащего двудольный граф, предоставляемый пользователю для примера алгоритма;
- public static final String PETERSON_GRAPH = "GraphPeterson.sv" – название файла, содержащего граф Петерсена, предоставляемый пользователю для примера алгоритма;

Класс Log реализован для вывода логов пользователю. Для наглядности в него были добавлены методы атрибутов(вида шрифта) и цвета, метод public void say же существует для проверки лога для вывода в окно интерфейса GUI.

Что немаловажно, также был создан класс Settings, отражающий основные параметры визуализации графа(цвет узлов, диаметр узлов, цвет шрифта и т.д.), локализацию, предоставляемую пользователю, и путь к директории, где можно сохранять изменения. Все параметры могут быть изменены пользователем и будут сохранены при следующем запуске программы.

Для визуализации диалоговых окон и самой программы также использовалась исходная платформо-независимая оконная библиотека графического интерфейса языка Java - Abstract Window Toolkit(AWT), что можно увидеть в директории dial.

6. ТЕСТИРОВАНИЕ

6.1 Мануальное тестирование

При запуске пробной версии программы всплывает диалоговое окно, представленное на рис.2

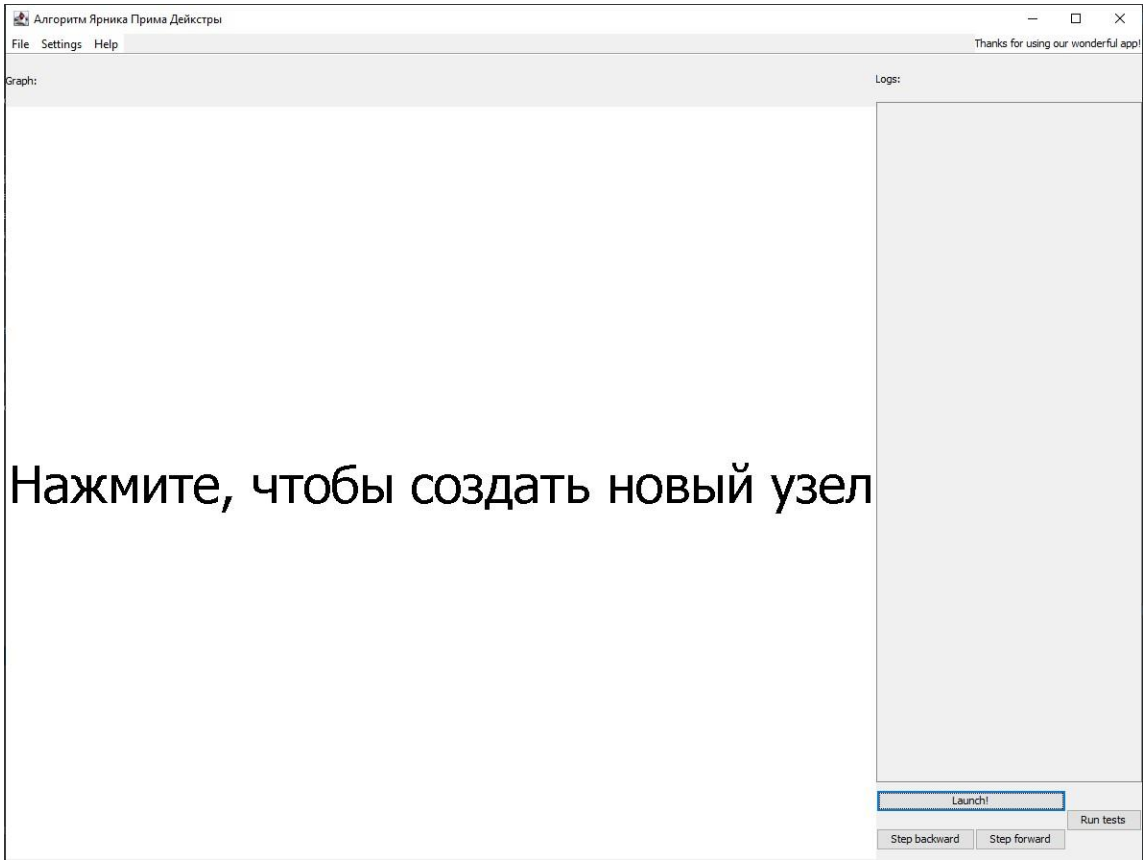

Рисунок 2 – начальный экран программы

При нажатии правой кнопкой мыши происходит вывод диалогового окна с предложением Создать узел, см. на рис.3. Далее происходит вывод диалогового окна с предложением назвать узел(рис.4) и после этого узел создается(в том месте, на которое вы нажали правой кнопкой мыши)

Рисунок 3 – вывод при нажатии правой кнопкой мыши

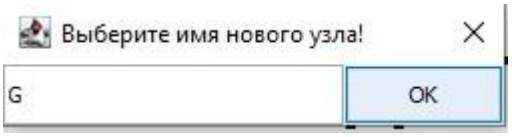

Рисунок 4 – диалоговое окно наименования узла

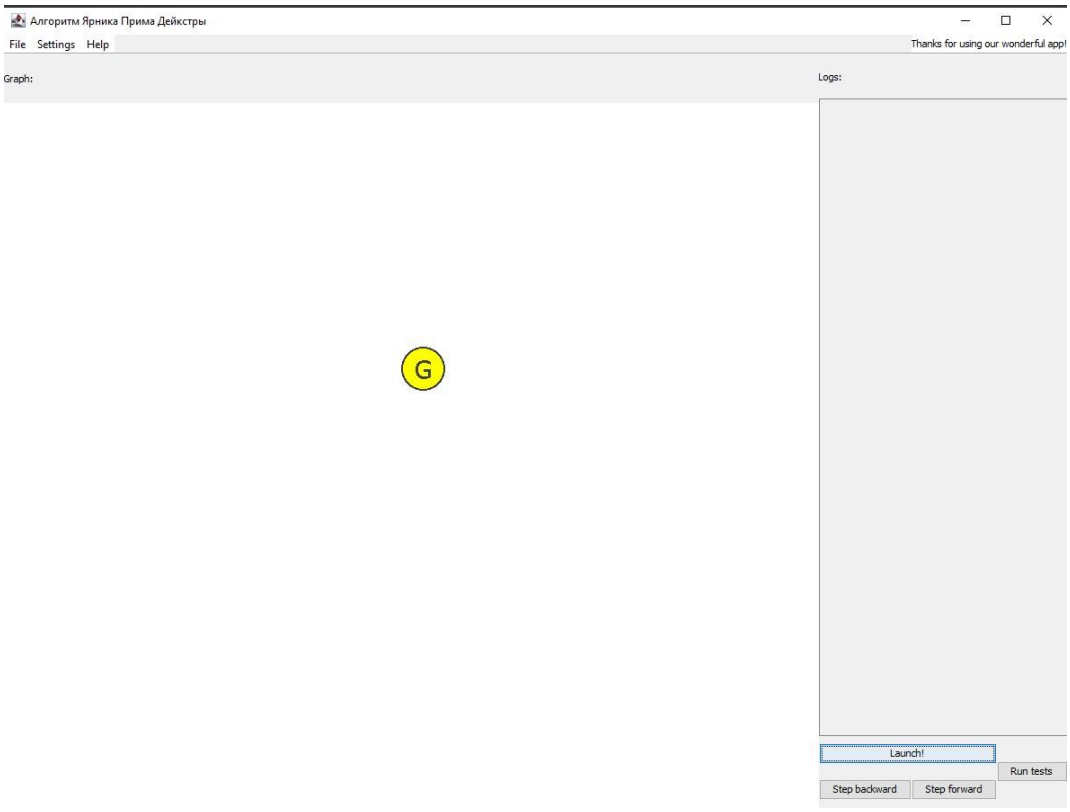

Рисунок 5 – Создание узла

Также проверяем создание ребер. При нажатии правой кнопкой мыши на узел мы видим рис.6. После которого, как показано на рис.7, нам предлагают назначить вес ребра.

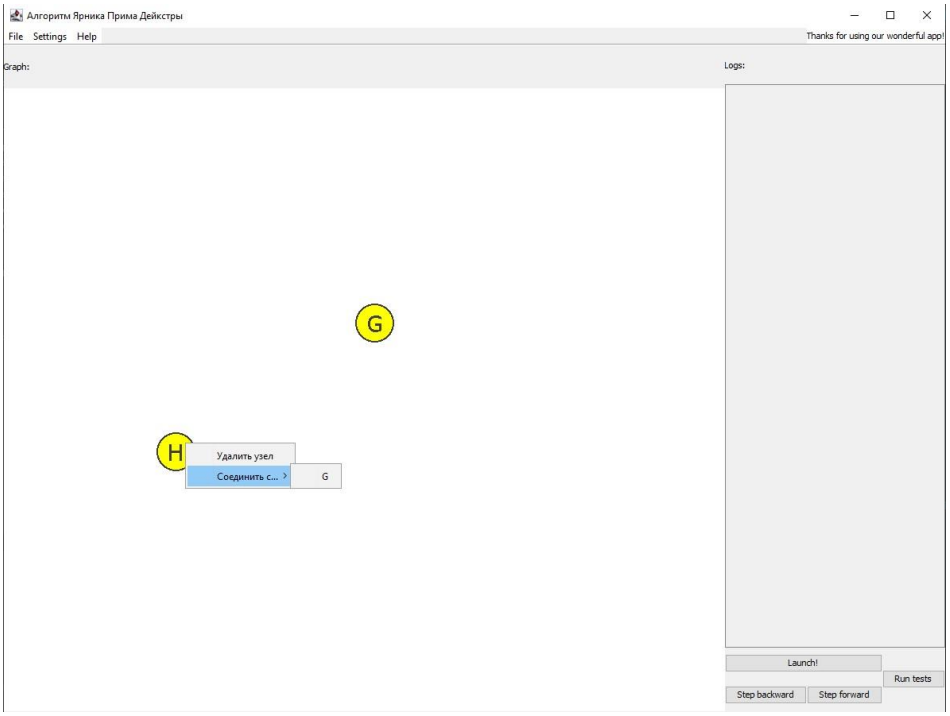

Рисунок 6 – Нажатие правой кнопкой мыши на узел

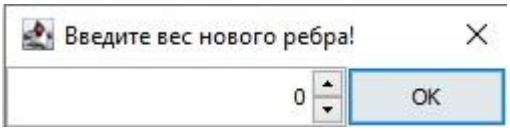

Рисунок 7 – Диалоговое окно взвешивания нового ребра

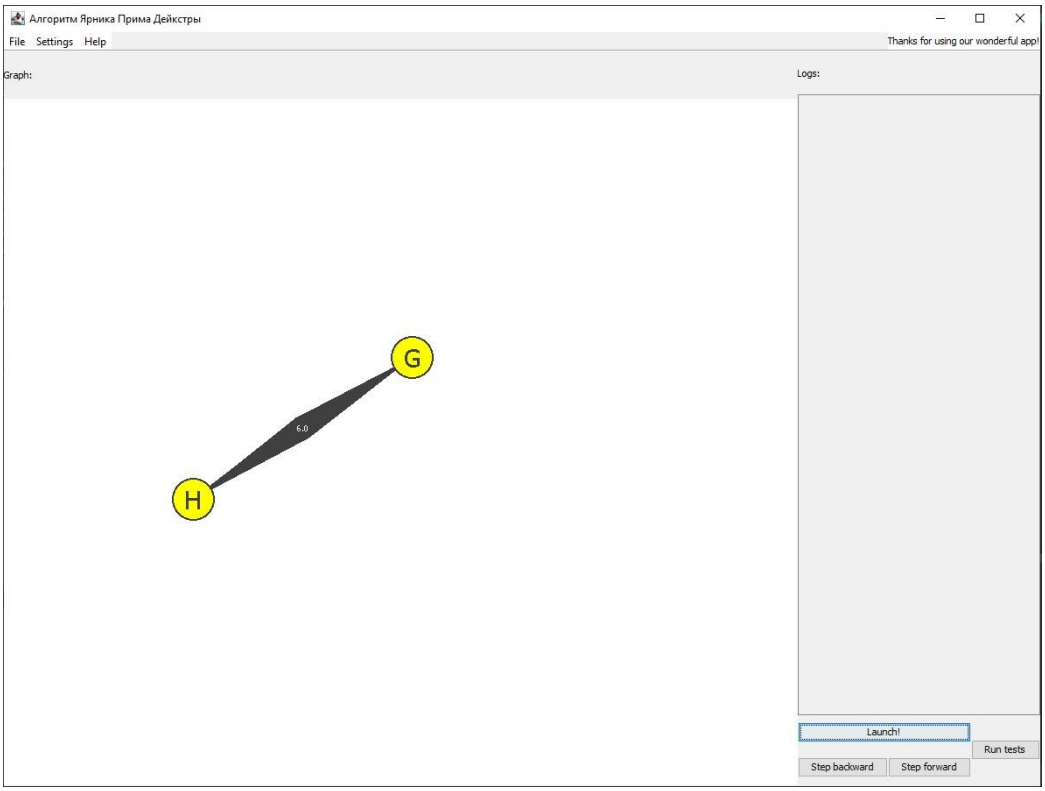

Рисунок 8 – Создание ребра

Также интерфейс позволяет нам в Настройках изменять параметры, например, как показано на рис.9-11, цвет шрифта узла. Также настройки предоставляют доступ к изменению локализации программы, что мы можем увидеть на рис.12-13.

Рисунок 9 – Диалоговое окно Настроек параметров

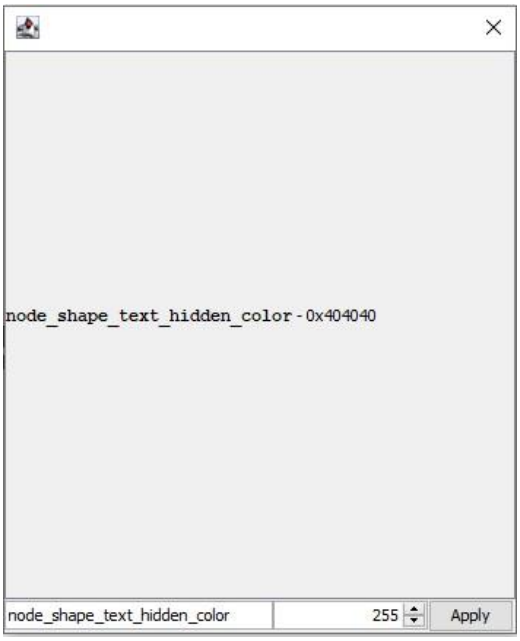

Рисунок 10 – Выбор параметра для замены(цвет шрифта на узлах)

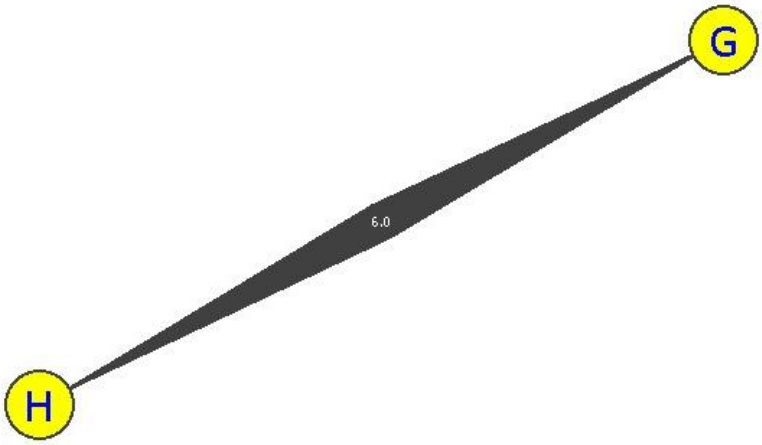

Рисунок 11 – Замена цвета шрифта на узлах

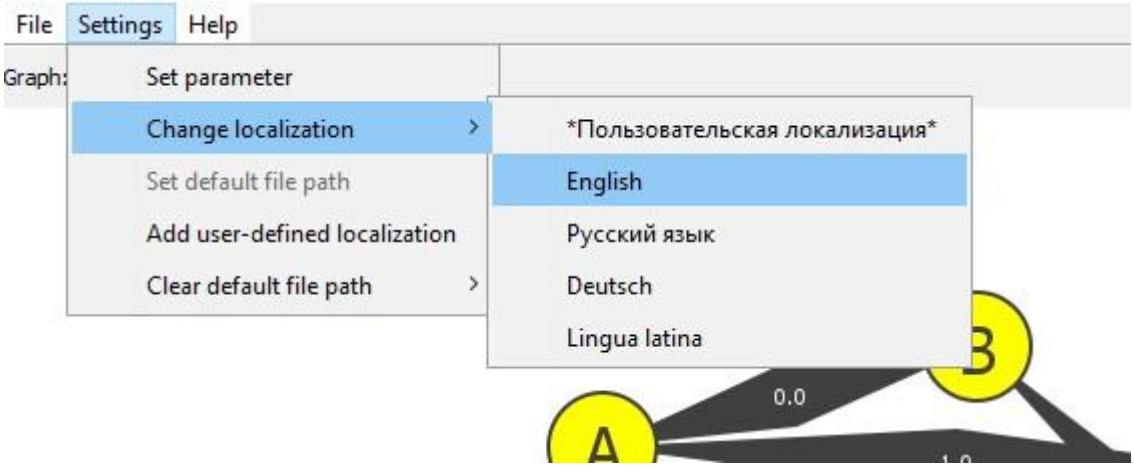

Рисунок 12 – Выбор языка программы

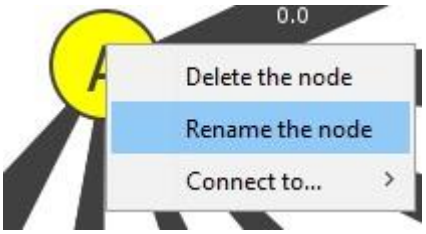

Рисунок 13 – Пример изменений языка программы на английский

Чтобы сохранять данные, туда, куда пользователю было бы удобнее, можно через Настройки, указать путь до нужной директории(см.рис.14-15).

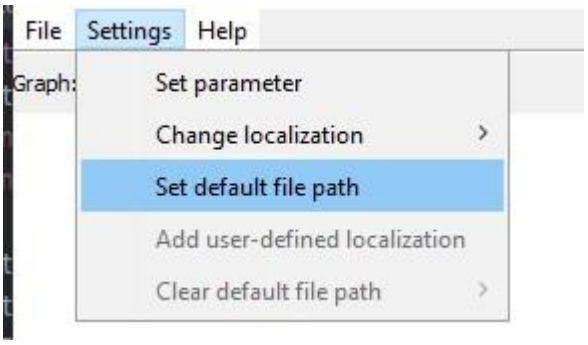

Рисунок 14 – Выбор Настроек пути в директорию для записи данных программы

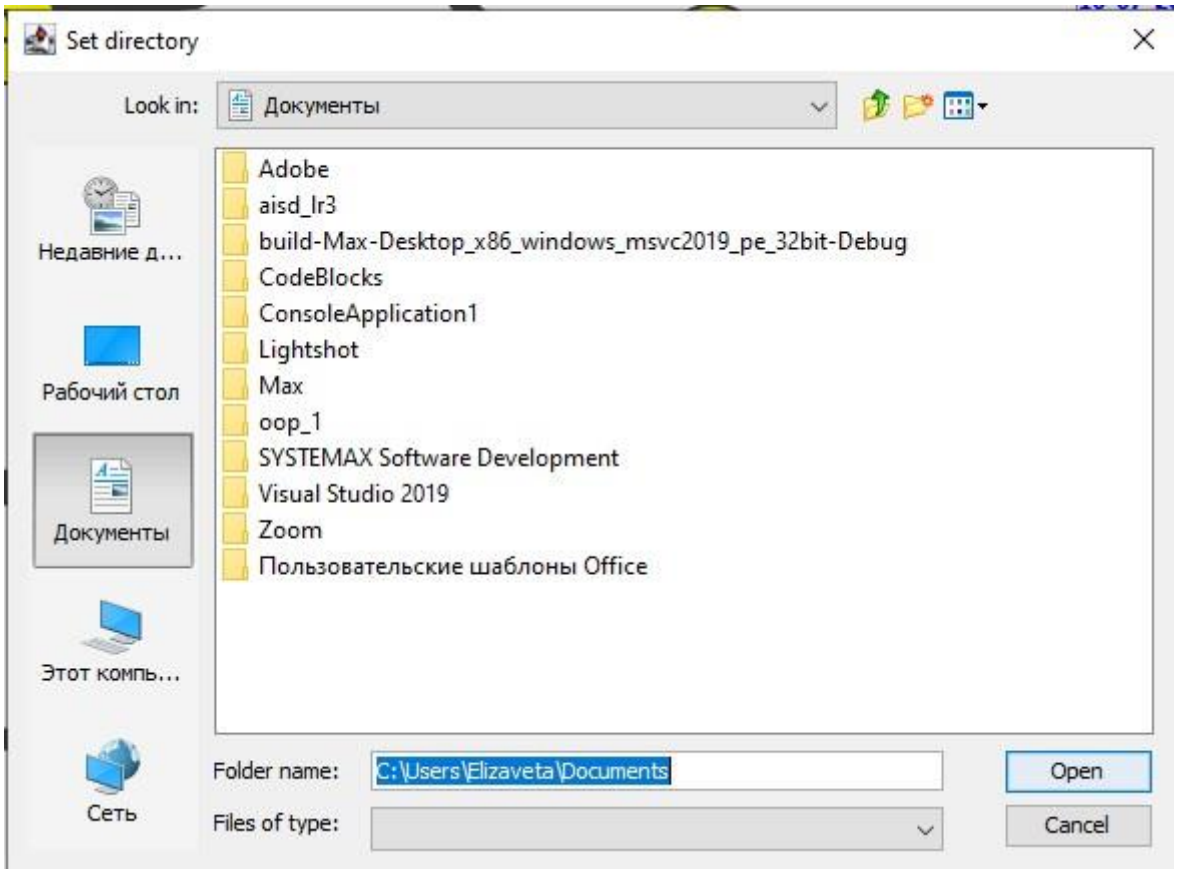

Рисунок 15 – Выбор пути в директорию для записи данных программы

Нами был создан граф, выбрана директория, следовательно нужно куда-то его сохранить, для этого выбираем File и Save as…(см.рис.16-17)

Рисунок 16 – Выбор сохранения графа на устройстве

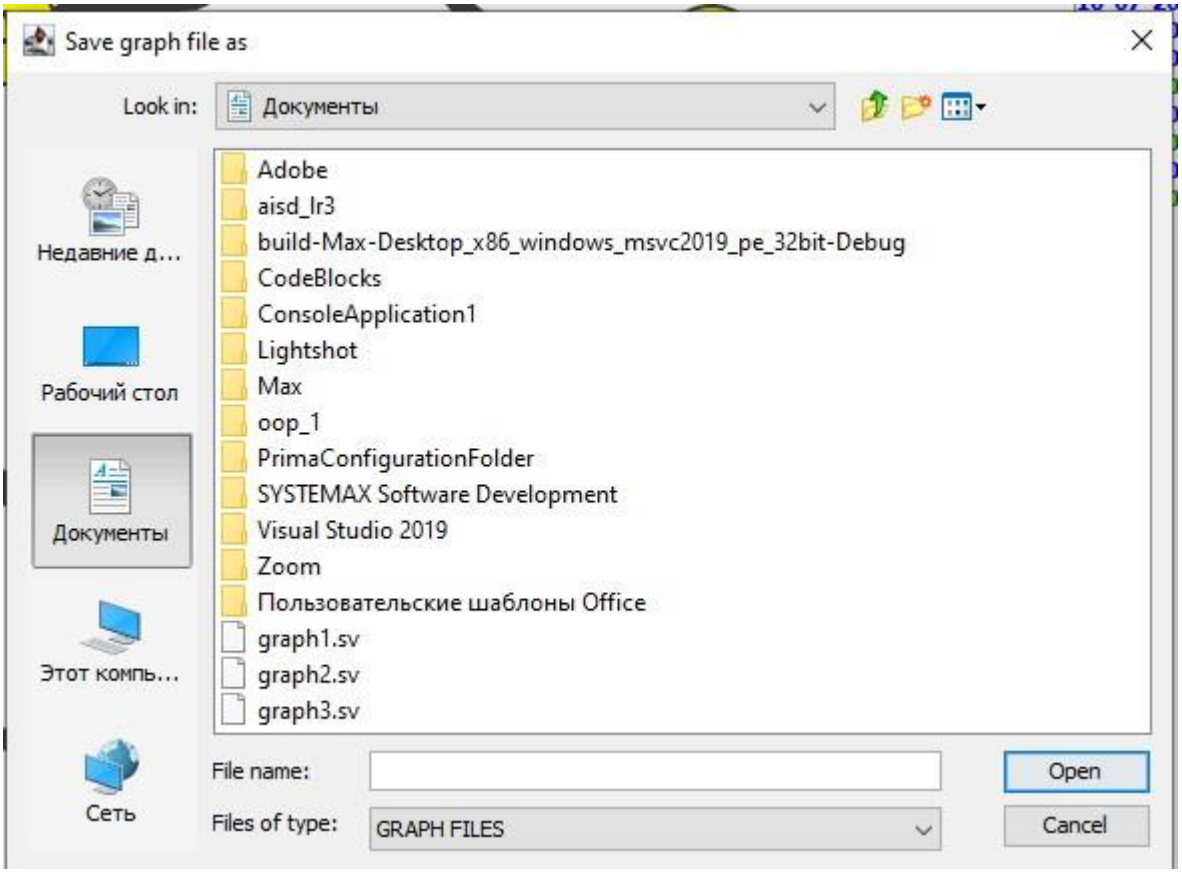

Рисунок 17 – Сохранение графа

Предположим, нам неинтересно создавать граф самим, тогда у нас есть возможность выбрать из трех предложенных, например граф Петерсена, рис.18. Для него можно запустить алгоритм Прима, рис.19

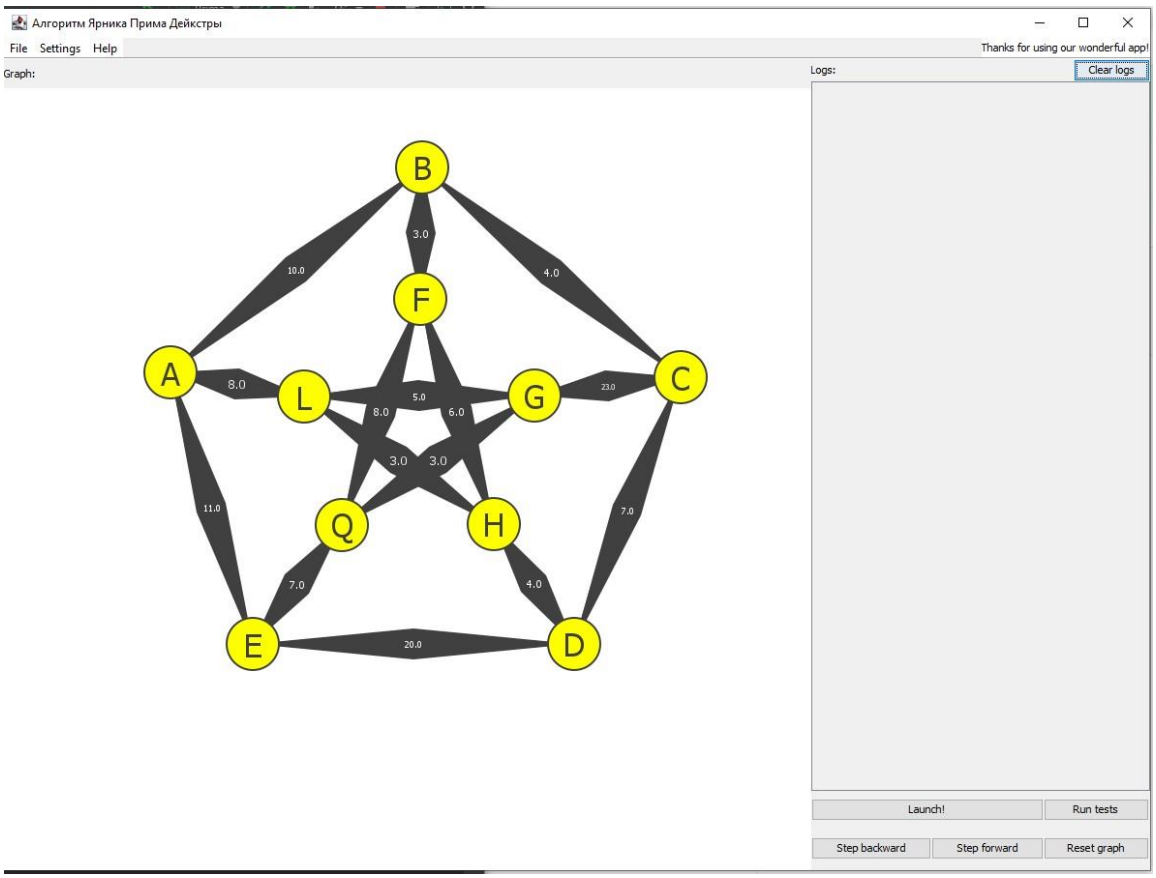

Рисунок 18 – Выбор графа Петерсена, для реализации алгоритма

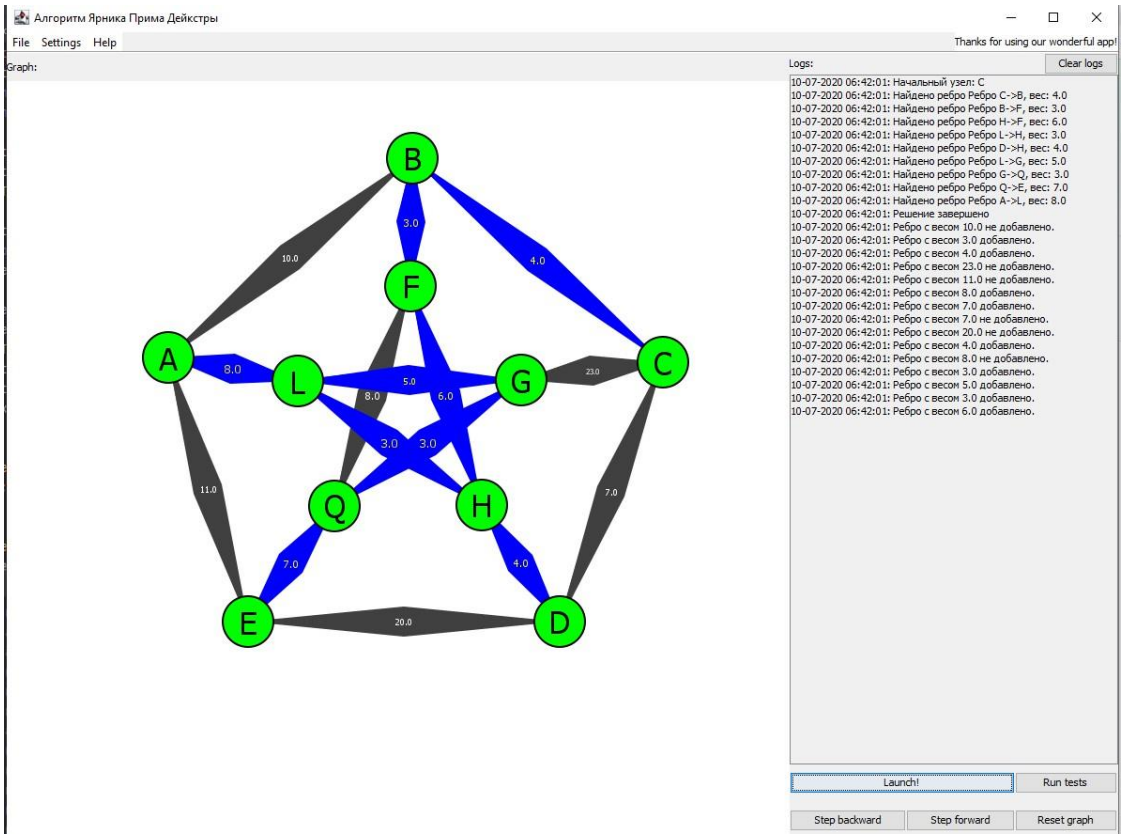

Рисунок 19 – Результат работы алгоритма Прима(синим – минимальное остовное дерево)

6.2 Unit тестирование программы

Для проведение автоматического тестирования программы использовалась библиотека junit 4. UNIT тесты были написаны с целью проверить основные моменты кода, а именно проверить верность нахождения минимального остовного дерева и корректность создания графа.

Для выполнения тестирования методов создания графа, был задан граф, представленный на рис.20. После запуска Unit тестирования корректные тесты для него будут выглядеть так, как на рис.21

```
graph = new Graph();
graph.addNode(new NodePlus(new Point2D.Double( x: 200,  y: 400),  name: "A"));
graph.addNode(new NodePlus(new Point2D.Double( x: 300,  y: 600),  name: "B"));
graph.addNode(new NodePlus(new Point2D.Double( x: 500,  y: 600),  name: "C"));

graph.addArk( strStart: "A",  strEnd: "B",  weight: 8);
graph.addArk( strStart: "B",  strEnd: "C",  weight: 9);
```

Рисунок 20 – Граф для Unit тестирования методов класса Graph

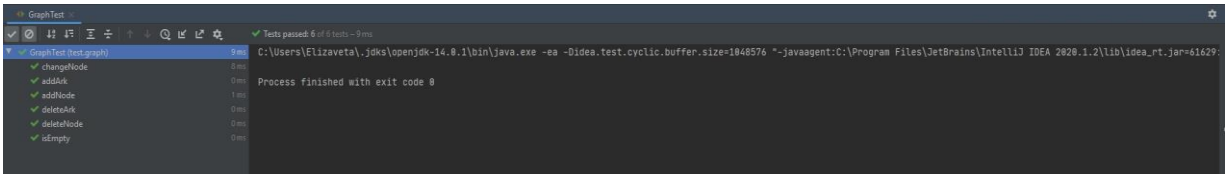

Рисунок 21 – Корректные тесты методов класса Graph

Unit тестирование методов PrimaAlgoritm происходило на графе, представленным на рис.22. Проверка методов происходит с помощью функции assertEquals(), которая сравнивает результат работы метода с сгенерированным нами, тем самым проверяя корректность алгоритма.

```
graph = new Graph();
graph.addNode(new NodePlus(new Point2D.Double( x: 200,  y: 400),  name: "A"));
graph.addNode(new NodePlus(new Point2D.Double( x: 300,  y: 600),  name: "B"));
graph.addNode(new NodePlus(new Point2D.Double( x: 500,  y: 600),  name: "C"));

graph.addArk( strStart: "A",  strEnd: "B",  weight: 8);
graph.addArk( strStart: "B",  strEnd: "C",  weight: 9);

alg = new PrimaAlgorithm();
```

Рисунок 22 - Граф для Unit тестирования методов алгоритма Прима

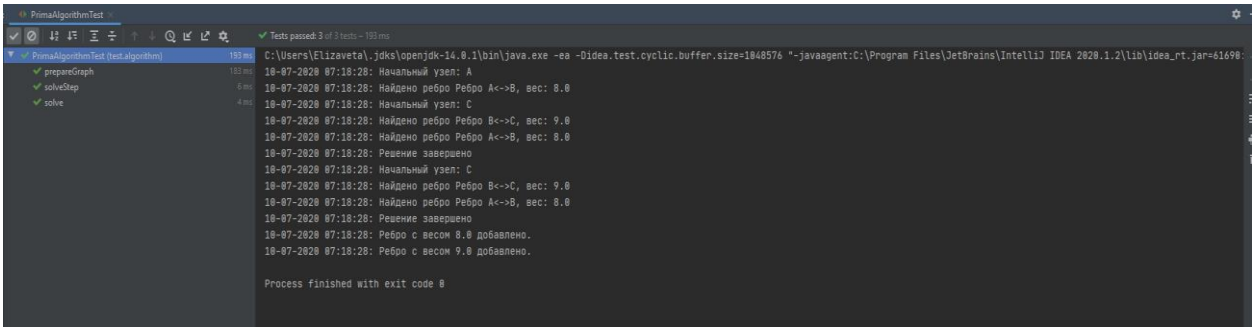

Рисунок 23 - Корректные тесты методов алгоритма Прима

6.3 Unit тестирование программы в GUI

Для Unit тестирования программы пользователем через графический интерфейс программы были созданы отдельные классы, запускающиеся с помощью кнопки 'Run Test'. Данные тесты были написаны с целью проверить основные моменты кода и правильность работы алгоритма. Все шаги выводятся в окно логов, а в случае ошибки, выводится в чем именно она произошла.

Класс GraphTestForGUI проверяет работу методов создания графа, а также добавления/удаления/переименования ребер и узлов.

Класс PrimaAlgorithmTestForGUI тестирует работу методов алгоритма ЯПД.

Для тестирования алгоритма мы берем три графа, из файлов GraphBipartite/GraphPetersen/GraphSample, окно логов будет выглядеть соответственно рис.24

Рисунок 24 – Окно логов с тестированием

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы мы ознакомились со средой разработки IntelliJ IDEA, а также языком программирования Java. Нами был успешно реализован алгоритм нахождения минимального остовного дерева, а также его визуализация и интерфейс самой программы. Все поставленные задачи были выполнены полностью и в срок. Также к основным требованиям нами были добавлены логирование, вывод/ввод из файла, изменение языка программы, изменение параметров визуализации, а также информация о создателях и самой программе.

Исходя из всего вышеперечисленного, можно сделать вывод, что поставленные задачи были выполнены успешно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Учебный курс по основам Java на Stepik: https://stepik.org/course/187/
2. Официальная документация к Java: https://docs.oracle.com/en/java/javase/
3. Java 8. Руководство для начинающих. Герберт Шилдт
4. https://ru.wikipedia.org/wiki/
5. https://habr.com/ru/
6. https://ru.stackoverflow.com/