

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

ОТЧЕТ
по учебной практике
Тема: Алгоритм Прима

Студентка гр. 9381	_____	Москаленко Е.М.
Студент гр. 9381	_____	Аухадиев А.А.
Студент гр. 9381	_____	Камакин Д.В.
Руководитель	_____	Жангиров Т.Р.

Санкт-Петербург

2021

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студентка Москаленко Е.М. группы 9381

Студент Аухадиев А.А. группы 9381

Студент Камакин Д.В. группы 9381

Тема практики: Алгоритм Прима

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: ЯПД (прим).

Сроки прохождения практики: 01.07.2021 – 14.07.2021

Дата сдачи отчета: 14.07.2021

Дата защиты отчета: 14.07.2021

Студентка	_____	Москаленко Е.М.
Студент	_____	Аухадиев А.А.
Студент	_____	Камакин Д.В.
Руководитель	_____	Жангиров Т.Р.

АННОТАЦИЯ

Цель данной практики заключается в реализации и визуализации алгоритма Прима. Для этого необходимо изучить язык программирования Java, принципы SOLID, паттерны проектирования. Кроме того, перед началом разработки, требуется написание спецификации и согласование архитектуры проекта. При разработке, для проверки корректности программы, выполняется тестирование алгоритма и графического интерфейса.

SUMMARY

The purpose of this practice is to implement and visualize the Prim algorithm. To do this, it's necessary to learn the Java programming language, the principles of SOLID, design patterns. In addition, before starting development, it is necessary to write a specification and agree on the project architecture. During development, to check the correctness of the program, the algorithm and the graphical interface are tested.

СОДЕРЖАНИЕ

	Введение	5
1.	Требования к программе	6
1.1.	Исходные требования к программе	6
1.1.1	Требования к визуализации	6
1.1.2	Требования к входным данным	6
1.1.3	Требования к выходным данным	6
1.2.	Уточнение требований после сдачи прототипа	6
2.	План разработки и распределение ролей в бригаде	7
2.1.	План разработки	7
2.2.	Распределение ролей в бригаде	7
3.	Особенности реализации	8
3.1.	Описание алгоритма	8
3.2.	Структуры данных	9
3.3.	Основные методы	10
3.4.	Особенности проектирования	11
3.5.	Описание модулей проекта через UML диаграммы классов	10
3.6.	Описание работы программы и алгоритма через UML диаграммы состояний	11
3.7.	Описание работы программы и алгоритма через UML диаграммы последовательности	13
4.	Тестирование	14
4.1	Unit тестирование программы	14
4.2	Мануальное тестирование программы	14
	Заключение	0
	Список использованных источников	0
	Приложение А. Исходный код – только в электронном виде	0

ВВЕДЕНИЕ

Программа должна предоставлять возможность визуализировать алгоритм Прима на произвольном графе. Для поиска минимального остовного дерева применяется следующее правило выбора: на каждом шаге из всех подходящих ребер выбирается ребро наименьшего веса. Это ребро вместе с одной новой вершиной добавляется к дереву.

У пользователя есть возможность пошагового выполнения алгоритма с выводом соответствующей информацией. Кроме того, можно очищать сцену для добавления нового графа, удалять вершины, а также менять веса и удалять рёбра.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные требования к программе

1.1.1 Требования к визуализации

Рёбра не должны накладываться на вершины. Вершина и ребро должны менять цвет на единый при добавлении в остовное дерево. У каждой вершины должен быть номер.

1.1.2 Требования к входным данным

Граф должен быть представлен в виде списка рёбер:

1 - 2 3

2 - 3 3

1 - 3 4

Где каждая вершина представлена числом и разделена от предка символом «-». Через пробел указывается вес ребра.

1.1.3 Требования к выходным данным

Аналогично входным.

1.2. Уточнение требований после консультации с преподавателем

В результате консультации с преподавателем были определены слабые места архитектуры, а именно: ошибка при реализации MVC, недостаточная проработанность UML-диаграмм. В результате было принято решение реализовывать MVVM.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

02.07-05.07 – разработка структур данных графа и алгоритма, изучение материалов по графической части программы

06.07 – тестирование и обсуждение в бригаде

07.07-10.07 – начало разработки графического интерфейса

11.07-13.07 – завершение разработки GUI, связывание с алгоритмом

14.07 – тестирование, документирование, сдача проекта

2.2. Распределение ролей в бригаде

Москаленко — бэкенд, алгоритм

Камакин — фронтенд

Аухадиев — тестирование, фронтенд

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1 Описание алгоритма

Алгоритм Прима — алгоритм поиска минимального остовного дерева во взвешенном неориентированном связном графе.

На вход алгоритма подаётся связный неориентированный граф в виде списков смежности. 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.

Программа позволяет пользователю запустить как весь алгоритм сразу, так и пошагово.

Асимптотика алгоритма.

Если искать каждый раз ребро простым просмотром среди всех возможных вариантов, то асимптотически будет требоваться просмотр $O(m)$ рёбер, чтобы найти среди всех допустимых ребро с наименьшим весом. Суммарная асимптотика алгоритма составит в таком случае $O(nm)$.

В данном случае для каждой ещё не выбранной будем хранить минимальное ребро, ведущее в уже выбранную вершину. Тогда, чтобы на текущем шаге произвести выбор минимального ребра, надо просто просмотреть эти минимальные рёбра у каждой не выбранной ещё вершины — асимптотика составит $O(n)$.

Добавление ребра в остов также выполняется за $O(n)$. Таким образом, алгоритм Прима имеет асимптотику $O(n^2)$.

3.2 Структуры данных

Название класса	Описание
Vertex	Класс, описывающий вершину графа; хранит номер вершины и словарь рёбер (Edge) и вершин, к которым можно перейти по ребру
Edge	Класс, описывающий ребро графа; хранит две вершины и вес произвольного типа
Graph	Класс, описывающий граф; хранит массив вершин

3.3 Основные методы

1) Основные методы класса Edge

Название	Описание
getNextVertex()	Возвращает непосещённую вершину, с которой ребро соединено

2) Основные методы класса Vertex

Название	Описание
getMinimum()	Возвращает ребро с минимальным весом из всех, что соединены с данной вершиной
forPreviousStep(Vertex to)	Параметры: <ul style="list-style-type: none"> Vertex to – ключ для удаления ребра из списка рёбер вершины Реализация отката назад на один шаг
deleteEdgeFromDictionary (int indexVertexTo)	Параметры: <ul style="list-style-type: none"> indexVertexTo – номер вершины Удаление ребра по номеру соединённой

	вершины
--	---------

3) Основные методы класса Graph

Название метода	Описание
addNewEdge(int indexVertex1, int indexVertex2, double edge12	Параметры: <ul style="list-style-type: none"> • indexVertex1 – номер первой вершины • indexVertex2 – номер второй вершины • edge12 – вес добавляемого ребра Добавление нового ребра в граф
deleteEdge(int indexVertexFrom, int indexVertexTo)	Параметры: <ul style="list-style-type: none"> • indexVertexFrom – номер первой вершины • indexVertexTo – номер второй вершины Удаление ребра по номерам двух вершин

4) Основные методы класса PrimAlgorithm

Название	Описание
runAlgorithm()	Полное выполнение алгоритма Прима, пока не будет получено минимальное остовное дерево
runAlgorithmByStep()	Реализация пошагового выполнения алгоритма, добавление в остовное дерево следующих ребра и вершины
previousStep()	Откат выполнения алгоритма на шаг назад (удаление из остовного дерева последнего добавленного ребра)
addEdgeToSpanningTree(Vertex from, Vertex to, Double edgeWeight)	Параметры: <ul style="list-style-type: none"> • Vertex from – первая вершина • Vertex to – вторая вершина

	<ul style="list-style-type: none"> • Double edgeWeight – вес ребра Добавление ребра в остовное дерево
restart()	Полное обнуления результатов работы алгоритма на любом шаге выполнения
writeToFile(String fileName)	Параметры: <ul style="list-style-type: none"> • String fileName – имя файла Функция записи графа в файл

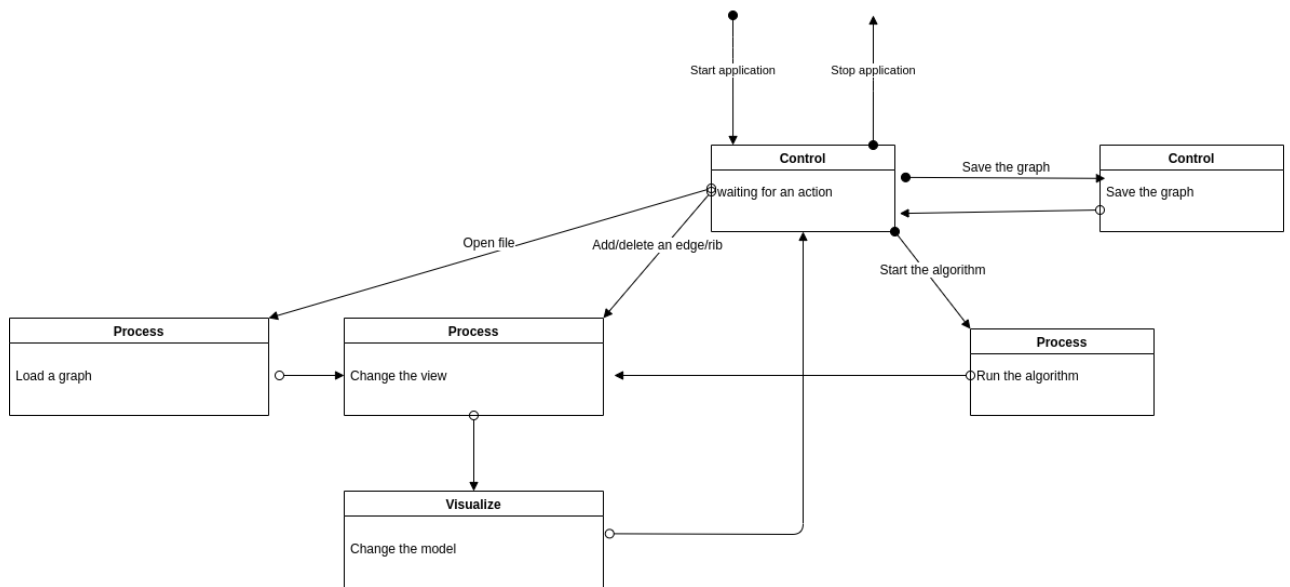
3.4 Особенности проектирования

1) Для связи модели с визуализацией используется шаблон проектирования Model-View-ViewModel (MVVM), реализованный через классы View - <ряд классов модуля ModelView (GraphVisualizer, GraphView, EdgeView и т.д.)> - PrimAlgorithm;

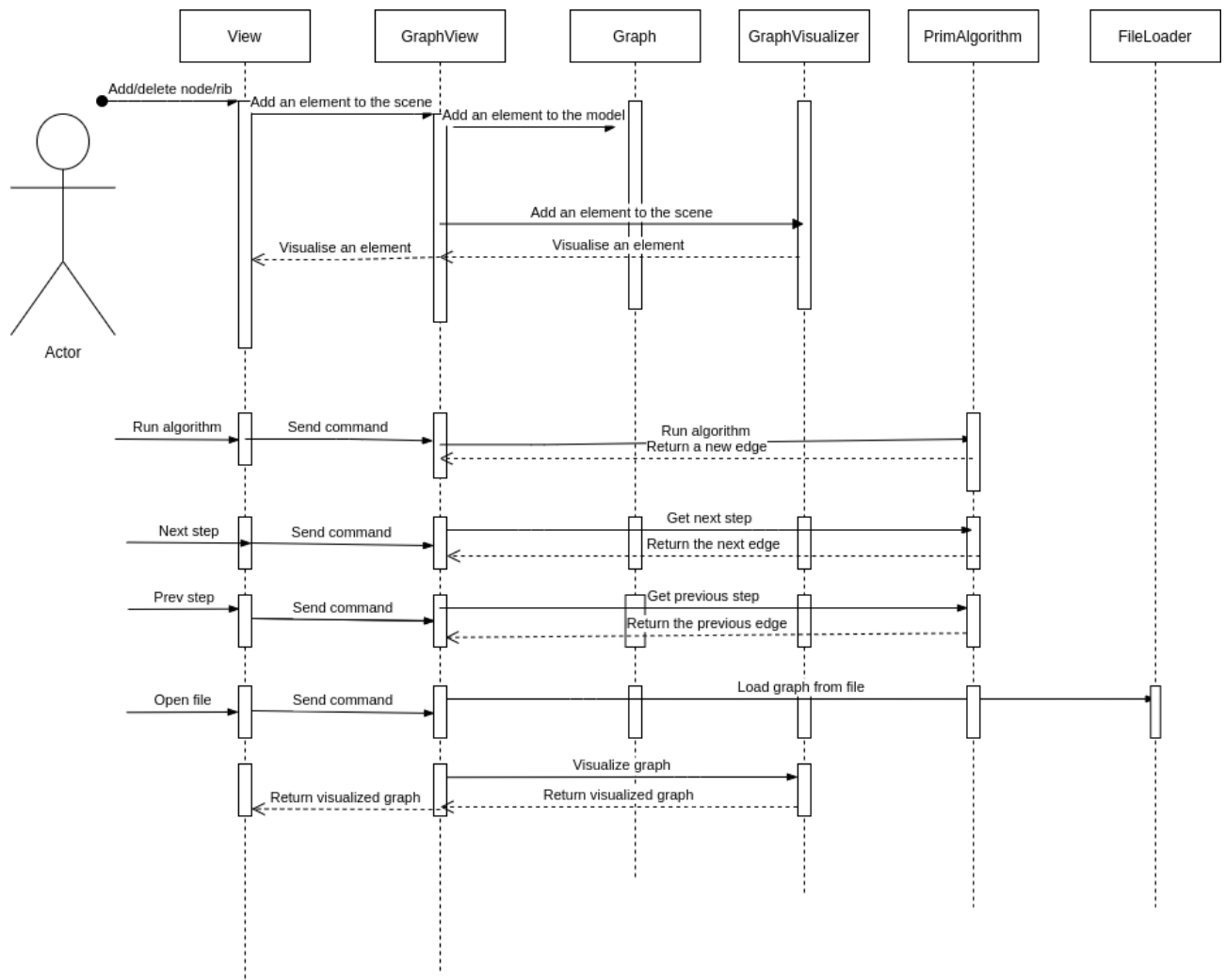
2) Для визуализации используется платформа Java FX, а также встроенные технологии в виде формата разметки fxml. Для проектирования интерфейса также использовалась программа Scene Builder.

3) Основным объектом сцены является AnchorPane, на котором размещается визуализация графа в виде объектов Pane, совмещающих в себе StackPane для отображения вершин, Line для отображения ребра и Text для отображения веса ребра.

3.5. Описание модулей проекта через UML диаграммы классов



3.7. Описание работы программы и алгоритма через UML диаграммы последовательности



4. ТЕСТИРОВАНИЕ

4.1. Unit тестирование программы

Для автоматической проверки корректности работы алгоритмов была использована JUnit4. Unit тесты покрывают основные методы классов Graph и PrimAlgorithm.

В классе Graph проверяются правильность добавления и удаления вершин и рёбер.

Для класса PrimAlgorithm тестируются методы, отвечающие за полное выполнение алгоритма, выполнение алгоритма по шагам, добавление рёбер в остоное дерево, сброс параметров к началу и откат на один шаг назад. Unit тестирование осуществляет проверку самых необходимых параметров для успешного выполнения алгоритма, используя данные, записанные в файлы для тестирования, которые можно расширять при возможном усложнении алгоритма.

4.2. Мануальное тестирование программы

Ручное тестирование программы выполнялось с использованием графического интерфейса, чтобы была возможность проверить корректность программы при работе с визуализацией. Наибольшее внимание было уделено нетривиальным случаям, чтобы убедиться в надежности алгоритма.

На рисунках 1 и 2 продемонстрированы результаты Unit тестирования классов Graph и PrimAlgorithm. На рисунках 3-8 представлены результаты ручного тестирования частично реализованного функционала.

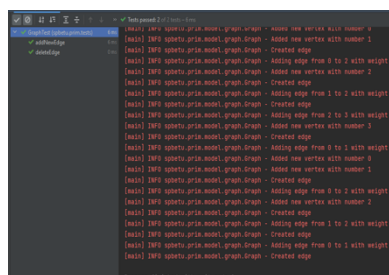
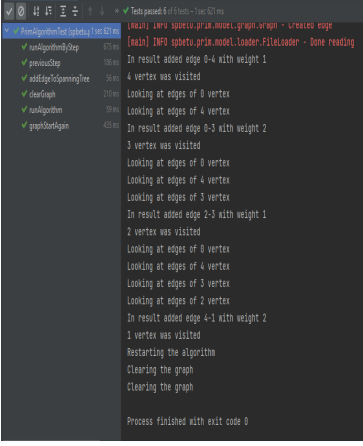


Рисунок 1.



```
Test passed 8 of 10 tests - 1 sec 521 ms
(main) java spbctu.prim.model.graph.Origin - Creates edge
(main) INFO spbctu.prim.model.loader.FileLoader - Done reading
returnStep 473 ms In result added edge 0-4 with weight 1
addStepSpanningTree 100 ms 4 vertex was visited
clearGraph 200 ms Looking at edges of 0 vertex
runAlgorithm 30 ms Looking at edges of 4 vertex
graphShortestPath 450 ms In result added edge 0-3 with weight 2
3 vertex was visited
Looking at edges of 0 vertex
Looking at edges of 4 vertex
Looking at edges of 3 vertex
In result added edge 2-3 with weight 1
2 vertex was visited
Looking at edges of 0 vertex
Looking at edges of 4 vertex
Looking at edges of 3 vertex
Looking at edges of 2 vertex
In result added edge 4-1 with weight 2
1 vertex was visited
Restarting the algorithm
Clearing the graph
Clearing the graph
Process finished with exit code 0
```

Рисунок 2.

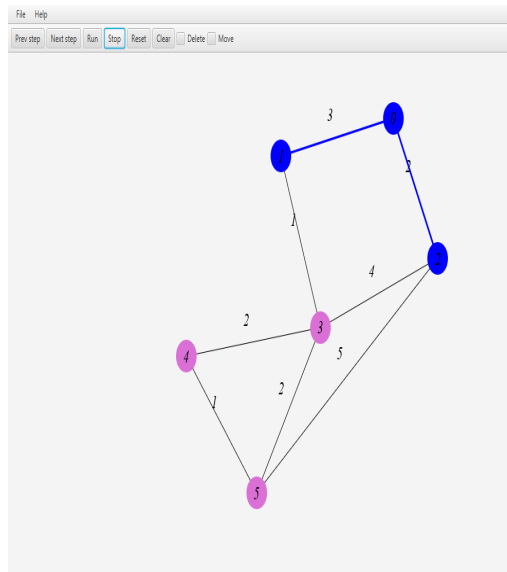


Рисунок 3. Остановка выполнения алгоритма (Stop) в произвольный момент выполнения после нажатия Run.

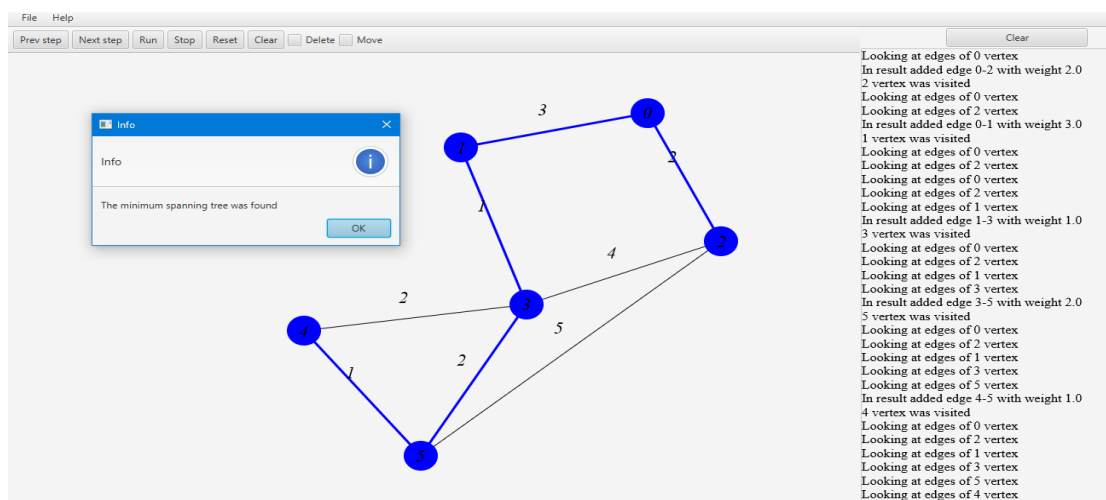


Рисунок 4. Корректное завершение после повторного нажатия Run.

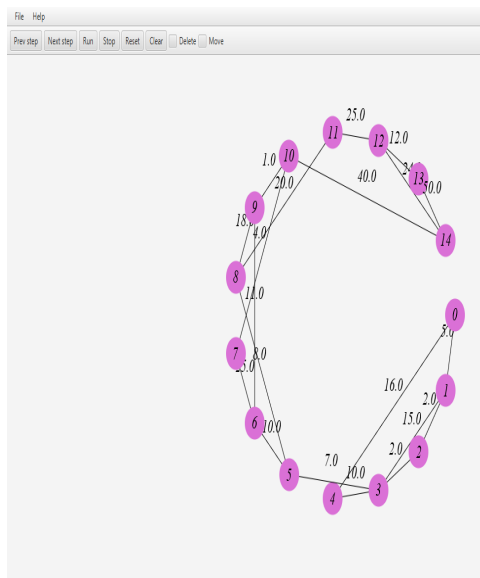


Рисунок 5. Загрузка графа из файла

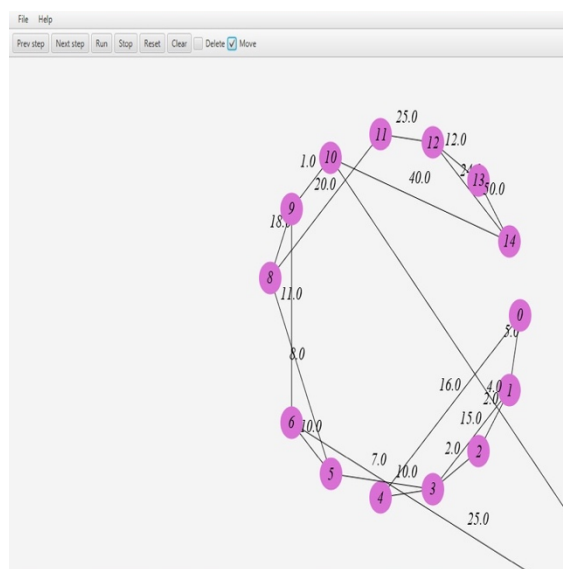


Рисунок 6. Некорректная работа команды Move

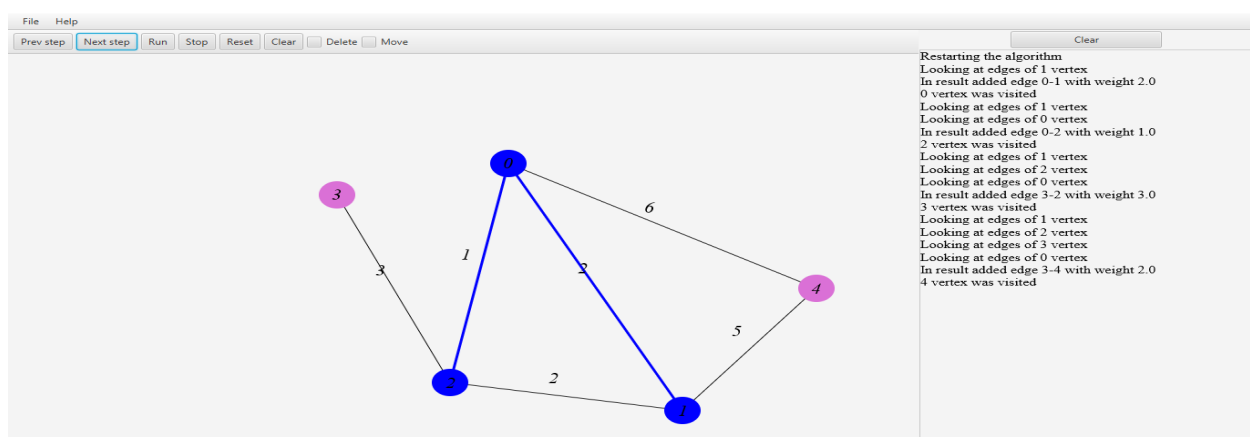


Рисунок 7. Выполнение команды Next step после очистки командой Clear.

Алгоритм корректно выполняет поиск, но некоторые рёбра не отображаются в остове.

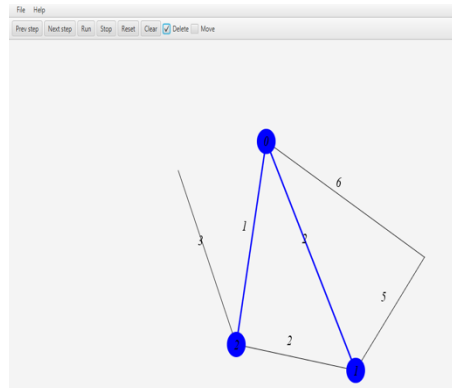


Рисунок 8. Некорректное выполнение команды Delete

Выводы по результатам тестирования:

- 1) Алгоритм корректно работает, в чём можно убедиться путём использования ввода-вывода через файл и консоль с любыми типами данных;
- 2) Визуализация работы алгоритма и выполнение команд, предоставленных в пользовательском интерфейсе, происходит успешно при работе с первым введённым графом, но после использования Clear и Delete связь между графом, изображённым на экране и хранящимся в алгоритме, нарушается;
- 3) При вводе нечисловых данных в качестве веса ребра узлы могут исчезнуть с экрана;
- 4) Имеются некоторые недочёты в реализации графического интерфейса.

Возможные решения:

- 1) Налаживание связи визуальной и алгоритмической составляющей проекта (классы View, GraphVisualizer);
- 2) Исправление ошибок в обработке нажатия кнопок и элементов графа (класс View);

ЗАКЛЮЧЕНИЕ

В результате выполнения данной практической работы была реализована программа на языке Java с использованием фреймворка JavaFX, выполняющая и визуализирующая алгоритм Прима. В процессе была разработана и скорректирована архитектура приложения. Кроме того, было проведено обширное тестирование как графической составляющей, так и самого алгоритма. После этого были устранены все найденные ошибки и проведены соответствующие работы по улучшению пользовательского опыта.

Были выполнены все поставленные задачи, включая реализацию алгоритма, создание графического интерфейса, их связывание и добавление возможности взаимодействия с программой через терминал.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б. И Ритчи Д. Язык программирования Си М.: Вильямс, 1978 288с.
2. Фримен Э. Бейтс Б. Робсон Э. Сьерра К. Head First. Паттерны проектирования, М.:Питер, 2018 657 с.
3. Рефакторинг.гуру <https://refactoring.guru/ru/>