

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 консультации с преподавателем	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. Тестирование на наличие ошибок предыдущей итерации	15
4.3. Тестирование добавленного функционала	18
Заключение	23
Список использованных источников	24
Приложение А. Исходный код – только в электронном виде	24

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

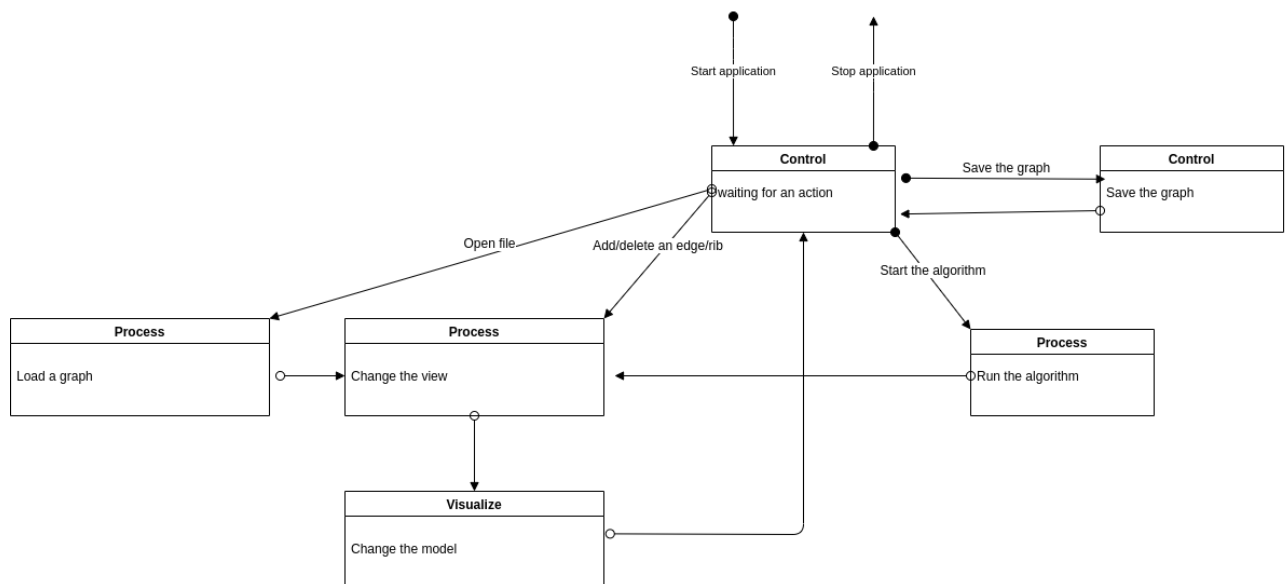
3.4 Особенности проектирования

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;

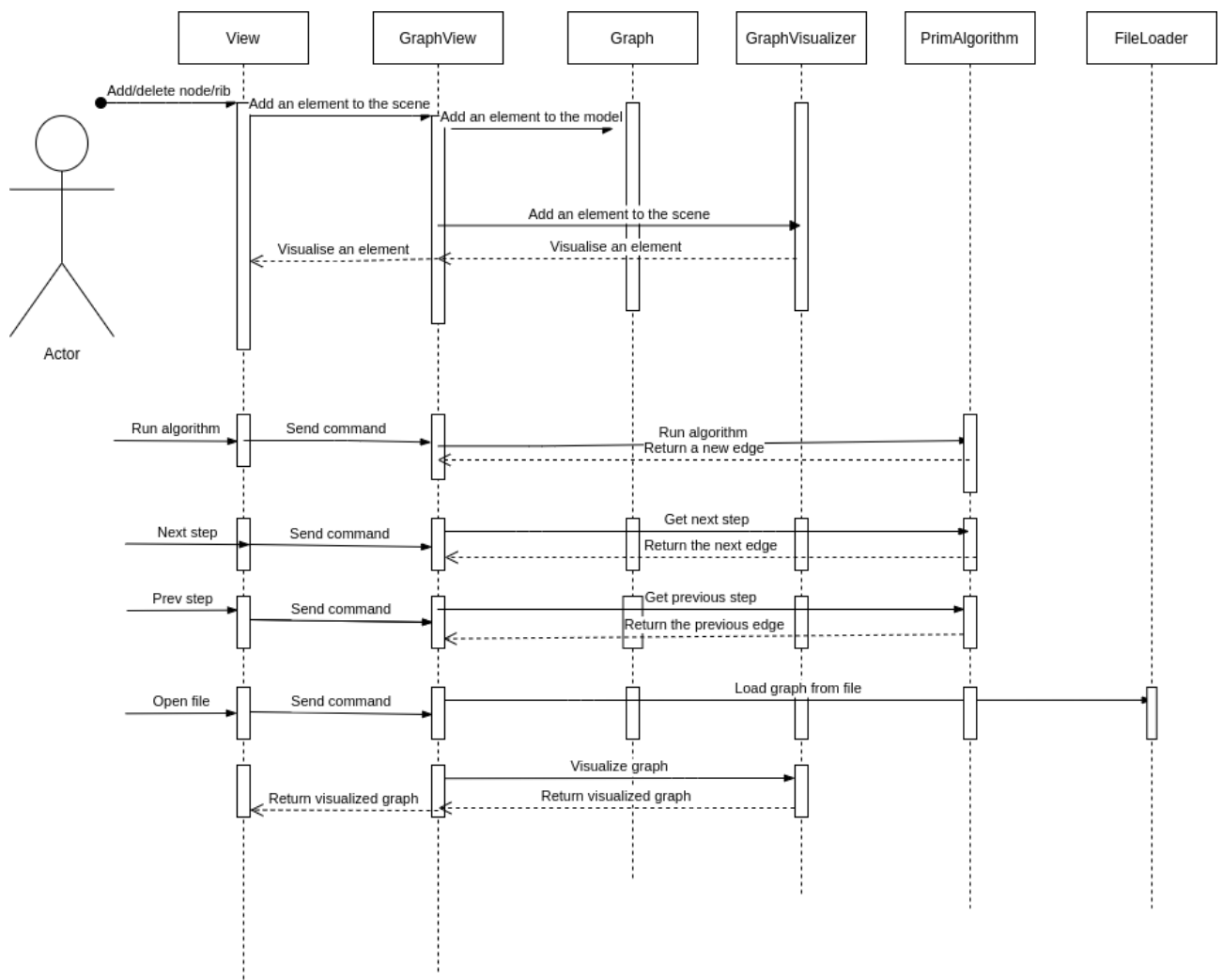
2) Для визуализации используется платформа Java FX, а также встроенные технологии в виде формата разметки fxml. Для проектирования интерфейса также использовалась программа Scene Builder.

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

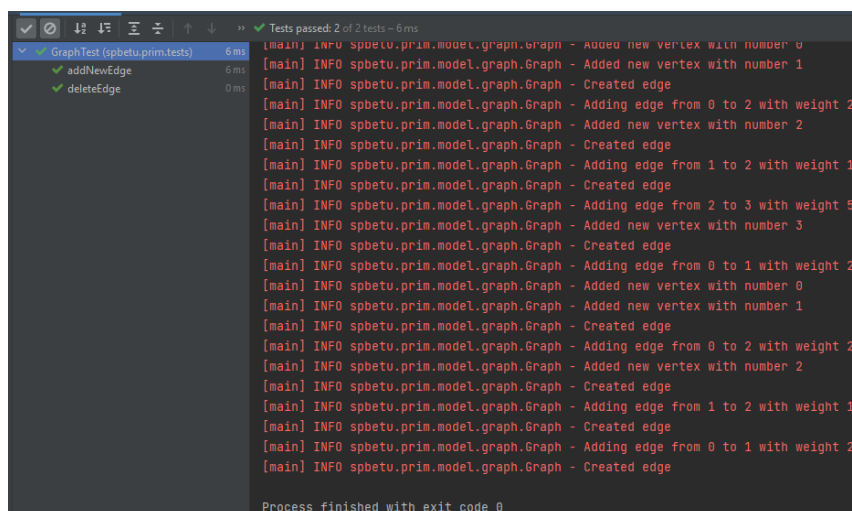
4.1. Unit тестирование программы

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 остовное 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.

На рисунках 1 и 2 продемонстрированы результаты Unit тестирования классов Graph и PrimAlgorithm.



```
Tests passed: 2 of 2 tests - 6 ms
[main] INFO spbetu.prim.model.graph.Graph - Added new vertex with number 0
[main] INFO spbetu.prim.model.graph.Graph - Added new vertex with number 1
[main] INFO spbetu.prim.model.graph.Graph - Created edge
[main] INFO spbetu.prim.model.graph.Graph - Adding edge from 0 to 2 with weight 2
[main] INFO spbetu.prim.model.graph.Graph - Added new vertex with number 2
[main] INFO spbetu.prim.model.graph.Graph - Created edge
[main] INFO spbetu.prim.model.graph.Graph - Adding edge from 1 to 2 with weight 1
[main] INFO spbetu.prim.model.graph.Graph - Created edge
[main] INFO spbetu.prim.model.graph.Graph - Adding edge from 2 to 3 with weight 5
[main] INFO spbetu.prim.model.graph.Graph - Added new vertex with number 3
[main] INFO spbetu.prim.model.graph.Graph - Created edge
[main] INFO spbetu.prim.model.graph.Graph - Adding edge from 0 to 1 with weight 2
[main] INFO spbetu.prim.model.graph.Graph - Added new vertex with number 0
[main] INFO spbetu.prim.model.graph.Graph - Added new vertex with number 1
[main] INFO spbetu.prim.model.graph.Graph - Created edge
[main] INFO spbetu.prim.model.graph.Graph - Adding edge from 0 to 2 with weight 2
[main] INFO spbetu.prim.model.graph.Graph - Added new vertex with number 2
[main] INFO spbetu.prim.model.graph.Graph - Created edge
[main] INFO spbetu.prim.model.graph.Graph - Adding edge from 1 to 2 with weight 1
[main] INFO spbetu.prim.model.graph.Graph - Created edge
[main] INFO spbetu.prim.model.graph.Graph - Adding edge from 0 to 1 with weight 2
[main] INFO spbetu.prim.model.graph.Graph - Created edge
Process finished with exit code 0
```

Рисунок 1.

```

✓ Tests passed: 6 of 6 tests - 1 sec 621 ms
✓ PrimAlgorithmTest (spbetu.f 1 sec 621 ms)
  ✓ runAlgorithmByStep 675 ms
  ✓ previousStep 186 ms
  ✓ addEdgeToSpanningTree 56 ms
  ✓ clearGraph 210 ms
  ✓ runAlgorithm 59 ms
  ✓ graphStartAgain 435 ms
[main] INFO spbetu.prim.model.graphn.Graphn - Created edge
[main] INFO spbetu.prim.model.loader.FileLoader - Done reading
In result added edge 0-4 with weight 1
4 vertex was visited
Looking at edges of 0 vertex
Looking at edges of 4 vertex
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

```

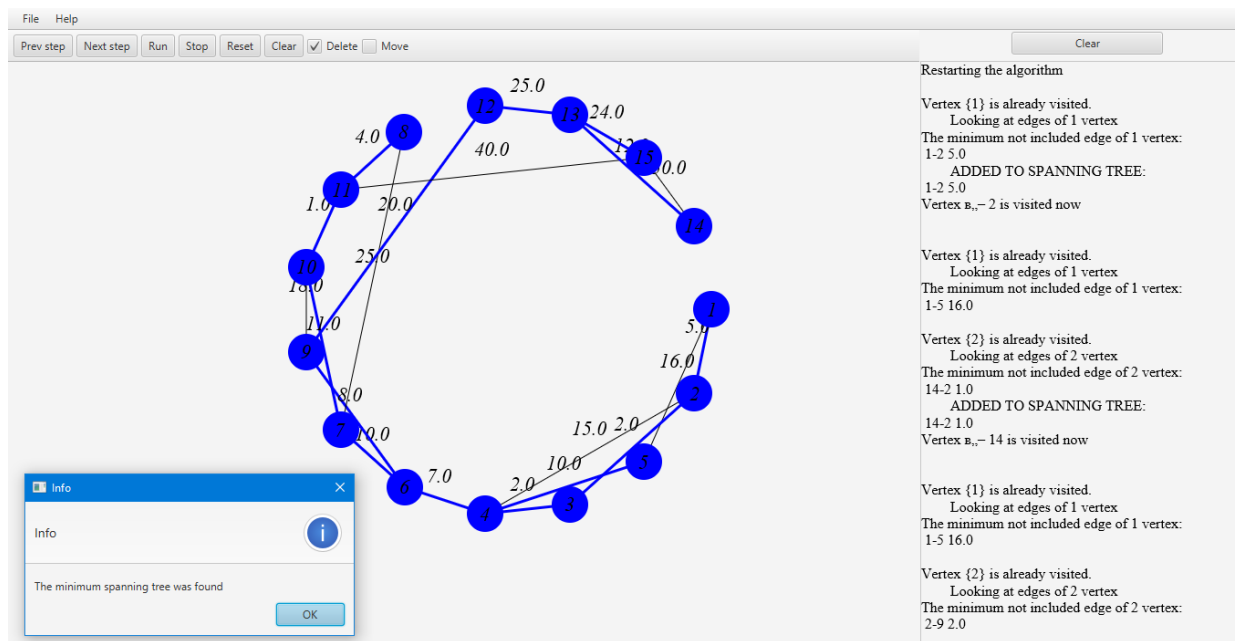
Рисунок 2.

4.2. Тестирование на наличие ошибок предыдущей итерации.

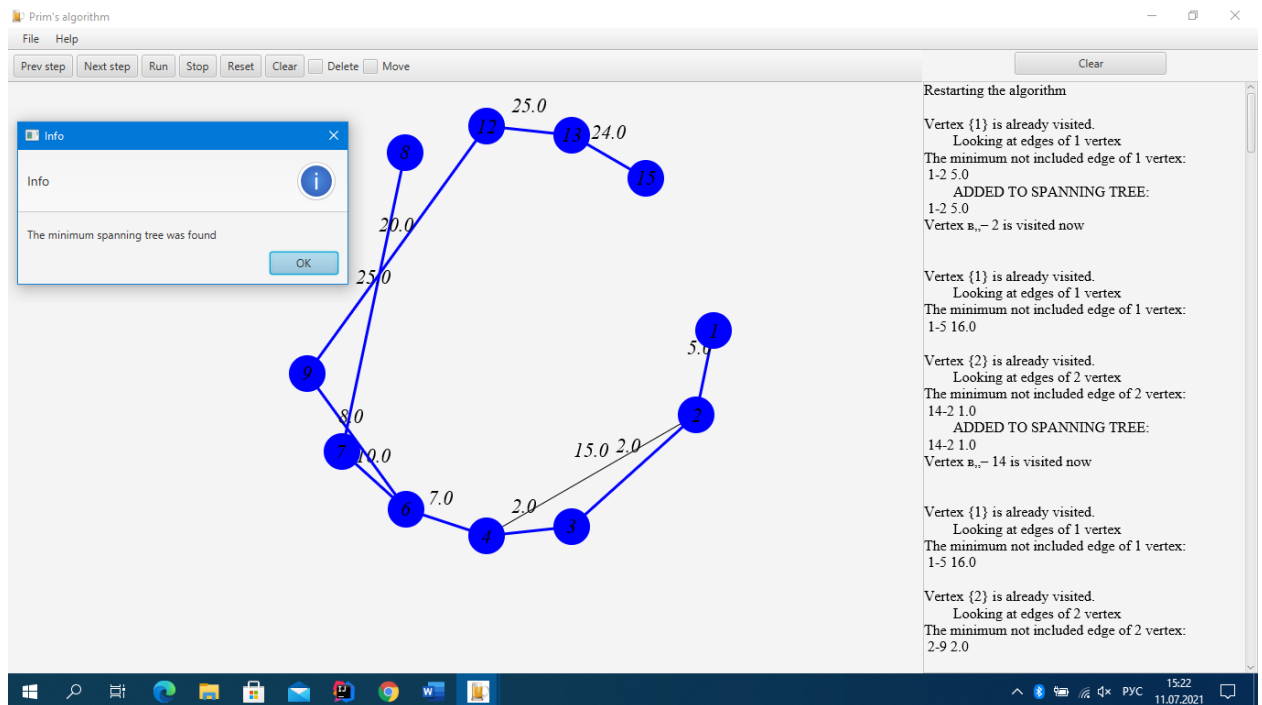
1. Работа с графом после использования Delete и Clear:

The screenshot shows a graph visualization application. The main window displays a graph with 5 vertices (0-4) and a minimum spanning tree. An 'Info' dialog box is open, stating 'The minimum spanning tree was found'. The right panel shows the algorithm's progress, including 'Restarting the algorithm' and 'Clearing the graph'.

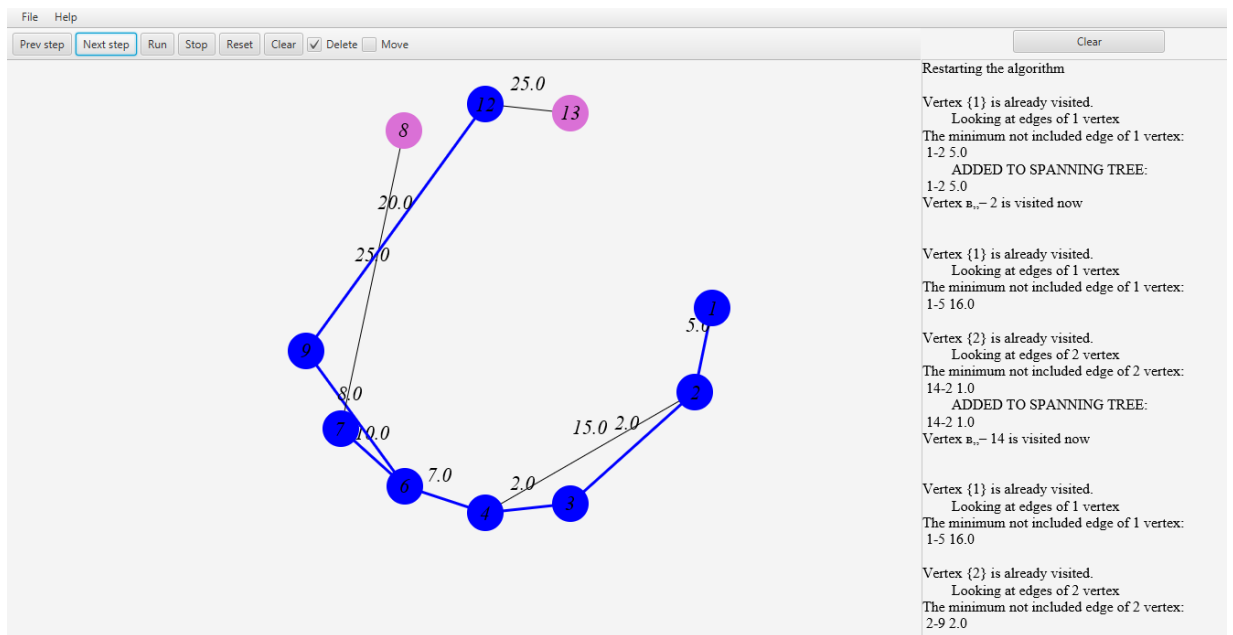
Успешная отработка после удаления через Clear



Успешная работа алгоритма после открытия нового файла

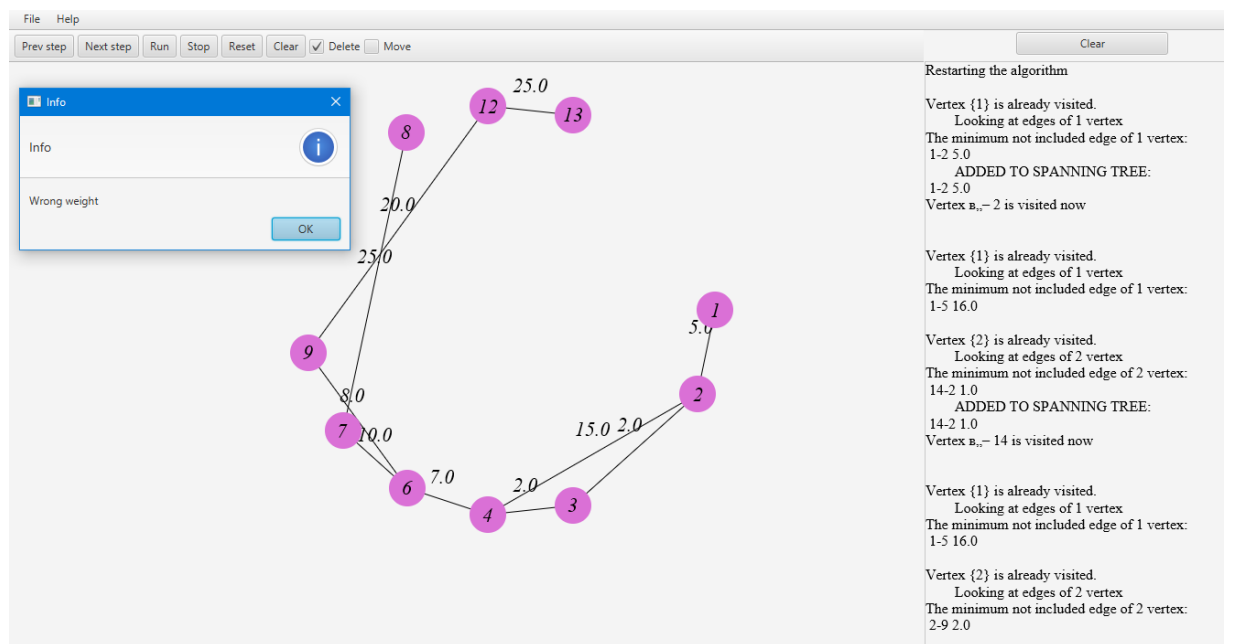


Успешная работа алгоритма после использования Delete



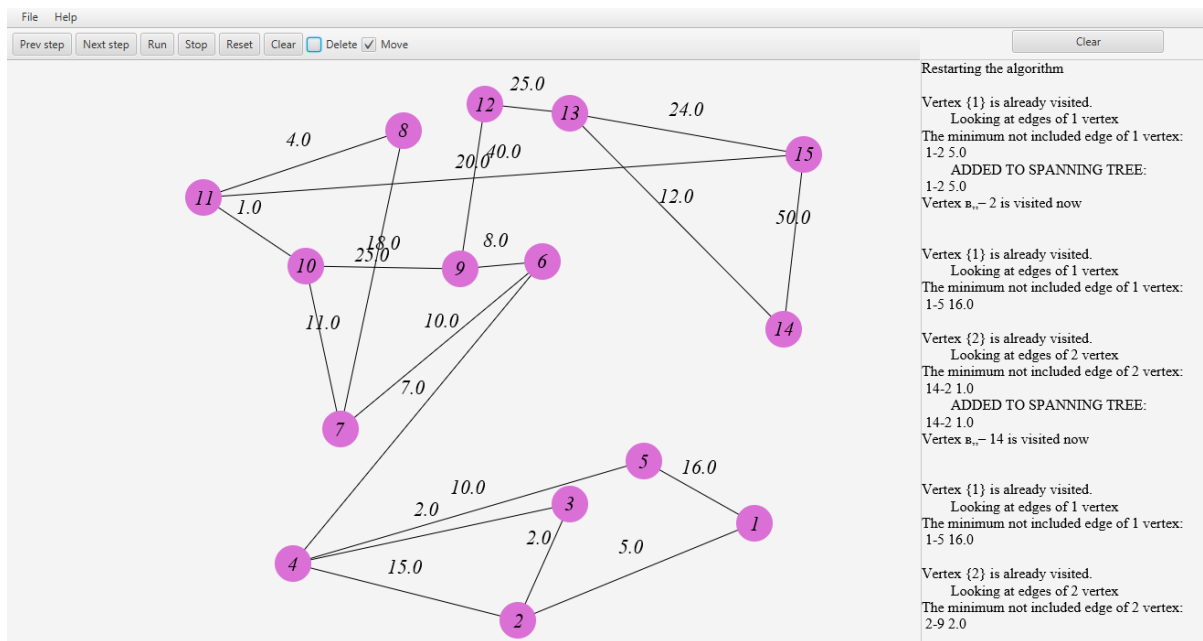
Успешная работа Next Step

2. Ввод нечисловых данных в качестве веса узла.



Успешная обработка ввода символа вместо числа в весе ребра

3. Проверка работы Move после открытия файла.



Успешная работа Move после открытия файла

Вывод: Все ошибки, обнаруженные в ходе тестирования предыдущей итерации, были исправлены.

4.3. Тестирование добавленного функционала (CLI).

1. Считывание графа из файла

```

9. Write to file
0. Exit
Your action: 9
Input path to the file: C:\Users\alexa\Documents\Practice-SPbETU-2021\tests\testGraph.txt

[main] INFO spbetu.prim.cli.viewmodel.ViewModel - Trying to read from C:\Users\alexa\Documents\Practice-SPbETU-2021\tests\testGraph.txt
[main] INFO spbetu.prim.model.loader.FileLoader - Start reading
[main] INFO spbetu.prim.model.graph.Graph - Adding edge from 0 to 1 with weight 10.0
[main] INFO spbetu.prim.model.graph.Graph - Added new vertex with number 0
[main] INFO spbetu.prim.model.graph.Graph - Added new vertex with number 1
[main] INFO spbetu.prim.model.graph.Graph - Created edge
[main] INFO spbetu.prim.model.graph.Graph - Adding edge from 1 to 2 with weight 5.0
[main] INFO spbetu.prim.model.graph.Graph - Added new vertex with number 2
[main] INFO spbetu.prim.model.graph.Graph - Created edge
[main] INFO spbetu.prim.model.graph.Graph - Adding edge from 2 to 3 with weight 4.0
[main] INFO spbetu.prim.model.graph.Graph - Added new vertex with number 3
[main] INFO spbetu.prim.model.graph.Graph - Created edge
[main] INFO spbetu.prim.model.graph.Graph - Adding edge from 0 to 3 with weight 2.0
[main] INFO spbetu.prim.model.graph.Graph - Created edge
[main] INFO spbetu.prim.model.graph.Graph - Adding edge from 2 to 0 with weight 1.0
[main] INFO spbetu.prim.model.graph.Graph - Created edge
[main] INFO spbetu.prim.model.loader.FileLoader - Done reading
Choose one of the following actions:

```

2. Вывод графа на экран

```

Choose one of the following actions:
1. Add an edge to the graph
2. Delete an edge from the graph
3. Next step
4. Prev step
5. Run algorithm
6. Output graph
7. Output the tree
8. Read from file
9. Write to file
0. Exit
Your action: 4

[main] INFO spbetu.prim.cli.view.CliApplication - Current graph:
[main] INFO spbetu.prim.cli.view.CliApplication - 2 - 0 1.0
[main] INFO spbetu.prim.cli.view.CliApplication - 0 - 3 2.0
[main] INFO spbetu.prim.cli.view.CliApplication - 0 - 1 10.0
[main] INFO spbetu.prim.cli.view.CliApplication - 1 - 2 5.0
[main] INFO spbetu.prim.cli.view.CliApplication - 2 - 3 4.0
Choose one of the following actions:

```

3. Результат работы алгоритма:

```

Vertex {3} is already visited.
    Looking at edges of 3 vertex
Vertex № 3 doesn't have not included edges.

Choose one of the following actions:
1. Add an edge to the graph
2. Delete an edge from the graph
3. Next step
4. Prev step
5. Run algorithm
6. Output graph
7. Output the tree
8. Read from file
9. Write to file
0. Exit
Your action: 7

[main] INFO spbetu.prim.cli.view.CliApplication - Current tree:
[main] INFO spbetu.prim.cli.view.CliApplication - 0 - 2 1.0
[main] INFO spbetu.prim.cli.view.CliApplication - 0 - 3 2.0
[main] INFO spbetu.prim.cli.view.CliApplication - 2 - 1 5.0

```

4. Добавление ребра в граф:

```

Choose one of the following actions:
1. Add an edge to the graph
2. Delete an edge from the graph
3. Next step
4. Prev step
5. Run algorithm
6. Output graph
7. Output the tree
8. Read from file
9. Write to file
0. Exit
Your action: 1
Input an edge with weight (example: 1 - 2 3): 3 - 4 3

[main] INFO spbetu.prim.model.graph.Graph - Adding edge from 3 to 4 with weight 3.0
[main] INFO spbetu.prim.model.graph.Graph - Added new vertex with number 4
[main] INFO spbetu.prim.model.graph.Graph - Created edge

```

```

6
[main] INFO spbetu.prim.cli.view.CliApplication - Current graph:
[main] INFO spbetu.prim.cli.view.CliApplication - 0 - 3 2.0
[main] INFO spbetu.prim.cli.view.CliApplication - 2 - 0 1.0
[main] INFO spbetu.prim.cli.view.CliApplication - 0 - 1 10.0
[main] INFO spbetu.prim.cli.view.CliApplication - 1 - 2 5.0
[main] INFO spbetu.prim.cli.view.CliApplication - 2 - 3 4.0
[main] INFO spbetu.prim.cli.view.CliApplication - 3 - 4 3.0

```

5. Удаление ребра из графа:

```

Your action: 2
Input an edge (example: 1 - 2): 1 - 2

Choose one of the following actions:
1. Add an edge to the graph
2. Delete an edge from the graph
3. Next step
4. Prev step
5. Run algorithm
6. Output graph
7. Output the tree
8. Read from file
9. Write to file
0. Exit
Your action: 6

[main] INFO spbetu.prim.cli.view.CliApplication - Current graph:
[main] INFO spbetu.prim.cli.view.CliApplication - 0 - 3 2.0
[main] INFO spbetu.prim.cli.view.CliApplication - 2 - 0 1.0
[main] INFO spbetu.prim.cli.view.CliApplication - 0 - 1 10.0
[main] INFO spbetu.prim.cli.view.CliApplication - 2 - 3 4.0
[main] INFO spbetu.prim.cli.view.CliApplication - 3 - 4 3.0

```

6. Использование функции Next step:

```

Vertex {0} is already visited.
  Looking at edges of 0 vertex
The minimum not included edge of 0 vertex:
2-0 1.0
  ADDED TO SPANNING TREE:
2-0 1.0
Vertex {2} is visited now

Choose one of the following actions:
1. Add an edge to the graph
2. Delete an edge from the graph
3. Next step
4. Prev step
5. Run algorithm
6. Output graph
7. Output the tree
8. Read from file
9. Write to file
0. Exit
Your action: 7

[main] INFO spbetu.prim.cli.view.CliApplication - Current tree:
[main] INFO spbetu.prim.cli.view.CliApplication - 0 - 2 1.0

```

7. Использование функции Prev step:

```

Your action: 4
Going back

Choose one of the following actions:
1. Add an edge to the graph
2. Delete an edge from the graph
3. Next step
4. Prev step
5. Run algorithm
6. Output graph
7. Output the tree
8. Read from file
9. Write to file
0. Exit
Your action: 7

[main] INFO spbetu.prim.cli.view.CliApplication - Current tree:
Choose one of the following actions:

```

8. Завершение работы:

```

Your action: 0

Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.8/userguide/command\_line\_interface.html#sec:command\_line\_warnings

BUILD SUCCESSFUL in 9m 20s
3 actionable tasks: 3 executed
2:10:12: Task execution finished 'Main.main()'.

```

Выводы по тестированию в рамках третьей итерации:

- 1) Ошибки предыдущей итерации были исправлены;
- 2) Ошибок в добавленном функционале не обнаружено.

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/>