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

ОТЧЕТ

по учебной практике

Тема: Визуализация алгоритма Дейкстры

Студент гр. 7381 _____ Адамов Я. В.

Студентка гр. 7381 _____ Алясова А. Н.

Студент гр. 7381 _____ Габов Е. С.

Руководитель _____ Жангиров Т. Р.

Санкт-Петербург
2019

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Адамов Я.В. группы 7381

Студентка Алясова А.Н. группы 7381

Студент Габов Е.С. группы 7381

Тема практики: Визуализация алгоритма Дейкстры

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: Дейкстра.

Сроки прохождения практики: 01.07.2019 – 14.07.2019

Дата сдачи отчета: 10.07.2019

Дата защиты отчета: 10.07.2019

Студент	_____	Адамов Я.В.
Студентка	_____	Алясова А.Н.
Студент	_____	Габов Е.С.
Руководитель	_____	Жангиров Т.Р.

АННОТАЦИЯ

В данной работе рассмотрена программа, решающая задачу построения кратчайшего пути в ориентированном графе методом Дейкстры. Программа разработана в среде IntelliJ IDEA на языке Java.

В проекте по входным данным создаётся граф, к нему применяется алгоритм Дейкстры, строится визуальный интерфейс для пошаговой обработки и наглядного показа работы алгоритма.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Формулировка задания.

Требуется разработать программу, которая решает задачу построения кратчайшего пути в ориентированном графе методом Дейкстры. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес.

При этом должен присутствовать графический интерфейс для наглядной демонстрации работы алгоритма и удобства взаимодействия пользователя с программой.

Основные теоретические положения.

Алгоритм Дейкстры – алгоритм, который находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса.

Каждой вершине графа сопоставляется метка — минимальное известное расстояние от этой вершины до ‘a’. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

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.

Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина ‘u’, имеющая минимальную метку. 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. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные требования к программе

Программа должна реализовывать алгоритм Дейкстры и находить с его помощью минимальный путь от стартовой вершины ко всем остальным.
Необходима подробная иллюстрация обхода графа и нахождения пути в нём с пояснениями на каждом шаге работы алгоритма.

1.1.1. Требования к вводу исходных данных.

Входные данные задаются пользователем из файла, вводятся с консоли либо задаются в поле для отображения графа.

Графический ввод.

По клику левой кнопки мыши по рабочему полю – поле, где должен находится граф - создаётся вершина. У вершины есть три состояния: черное, синие и жёлтое. Ребро строится из синей вершины в желтую вершину при вводе веса ребра в поле “Weight” и нажатию кнопки интерфейса “Add”. Переход из состояния в состояние происходит по нажатию правой кнопки мыши. Вершину можно удалить, нажав на нее левой клавишей мыши.

Файловый ввод.

Данные поступают в Json-массив. В каждом элементе массива есть данные по соответствующему ключу:

1) “name” – имя вершины.

2) “location” – координата этой вершины в виде массива из 2-х элементов: x, y.

3) “edge” - массив ребер из 2-х элементов: имя второй вершины, в которую ребро входит, вес ребра.

Консольный ввод.

Для того, чтобы добавить вершину нужно ввести в поле "Name" имя вершины, затем нажать кнопку "Add". Также можно удалить вершину, нажав кнопку “Remove”. Чтобы добавить ребро нужно ввести в поля "From" и "To" названия вершин, затем ввести в поле "Weight" вес ребра, после чего нажать

кнопку "Add" (можно вводить вершины, которые еще не созданы, они появятся автоматически).

1.1.2. Требования к визуализации.

Программа предоставляет интерфейс с пояснениями и изображением графа для его пошаговой обработки алгоритмом Дейкстры. Схематично интерфейс изображён на рис. 1 и рис. 2. На рис. 1 представлен интерфейс при вводе графа и работе с ним, а на рис. 2 – интерфейс, для просмотра алгоритма и работы с ним.

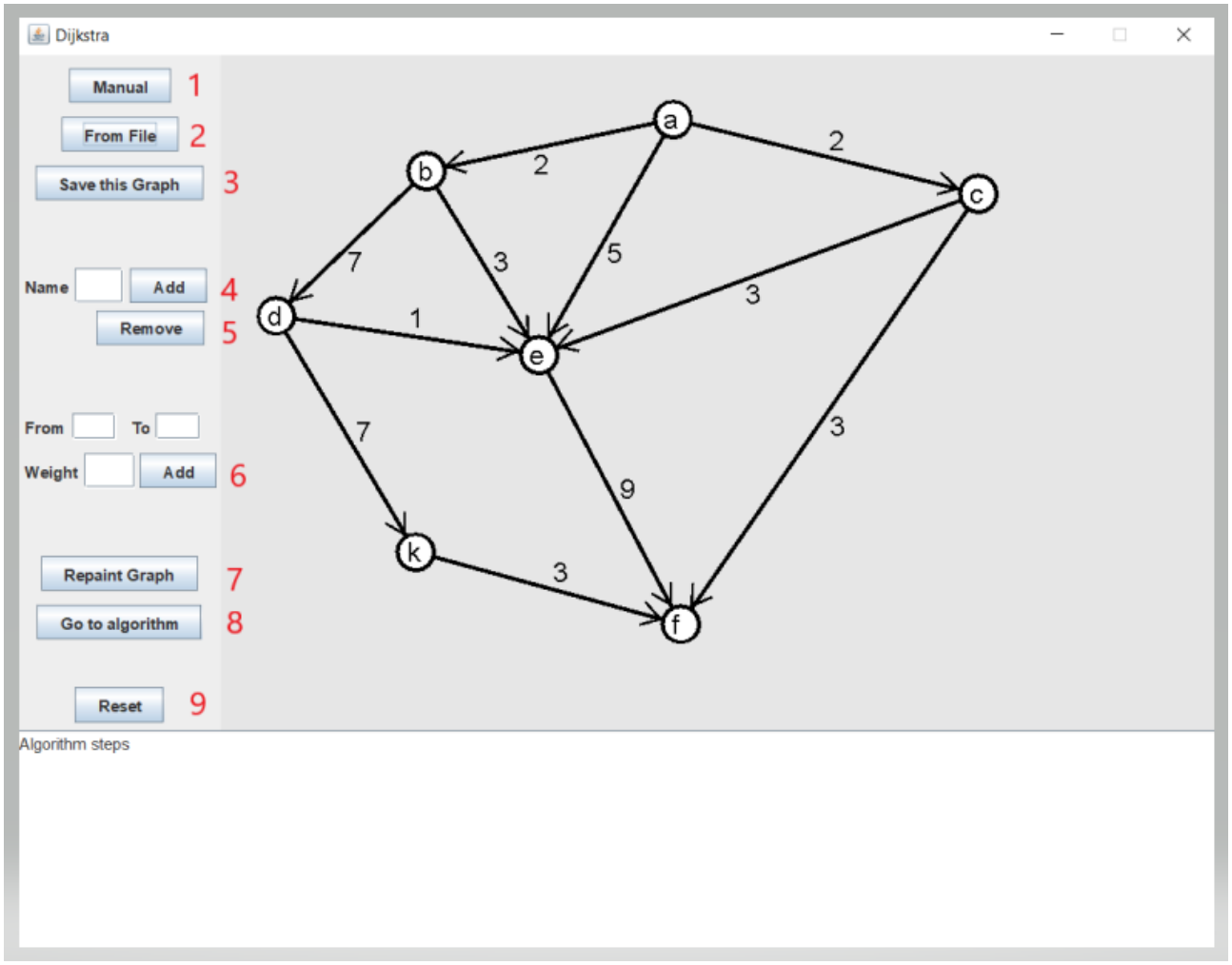

Рисунок 1

1 – Справка.

2 – Считать граф из файла.

3 – Сохранить граф в файл.

4 – Добавить вершину.

5 – Удалить вершину.

6 – Добавить ребро.

7 – Изобразить граф в других координатах.

8 – Приступить к работе алгоритма.

9 – Очистить рабочее поле.

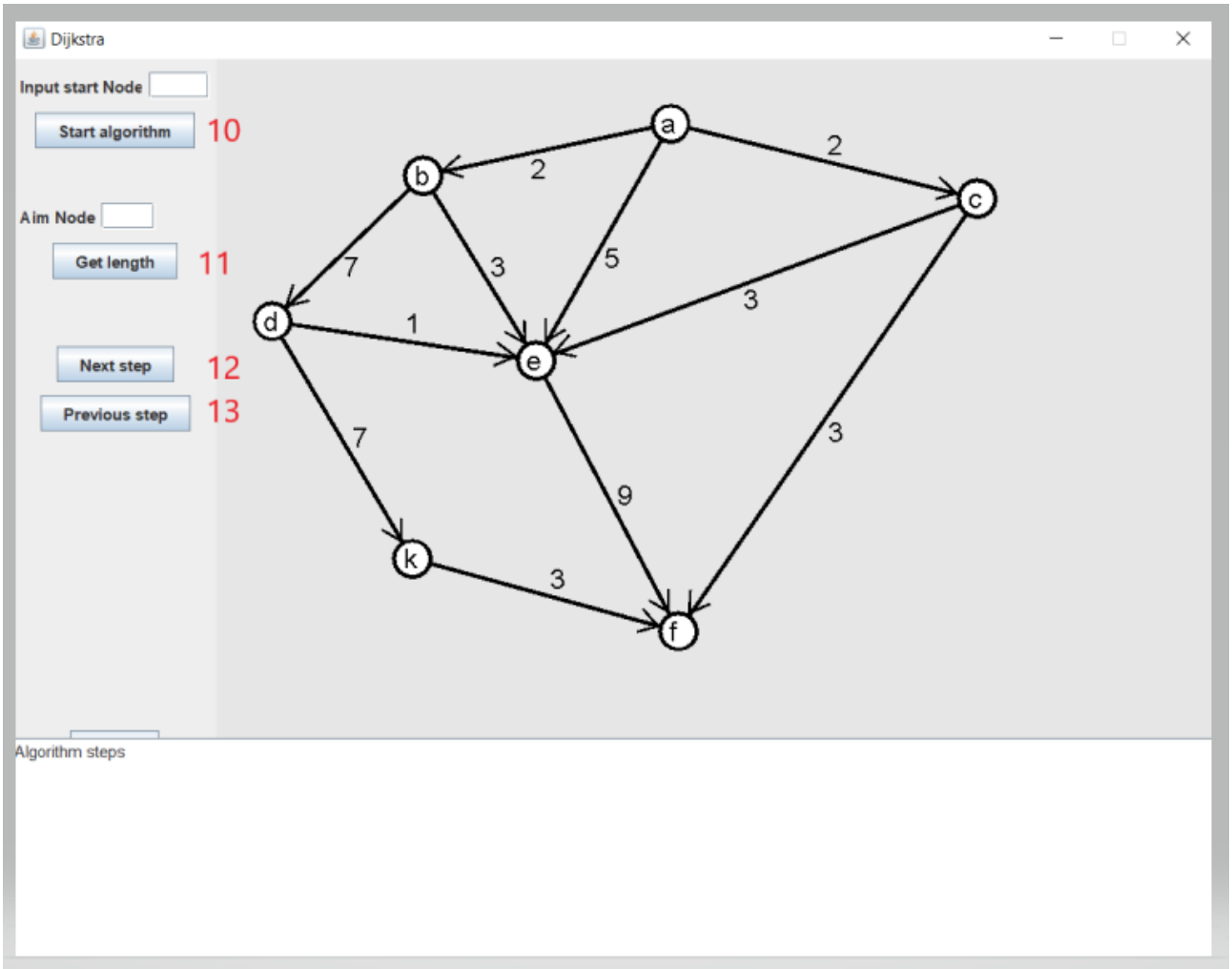

Рисунок 2

10 – Обозначить стартовую вершину и запустить алгоритм.

11 – Узнать расстояние от стартовой вершины до заданной.

12 – Шаг вперед по алгоритму.

13 – Шаг назад по алгоритму.

1.1.3. Требования к выводу результата.

Выходные данные представляются в данном порядке: сначала выводится вершина, до которой нужно было найти кратчайший путь, затем сам путь, далее число, которое является значением расстояния до заданной вершины.

1.1.4 UML-диаграммы

1) Диаграмма классов

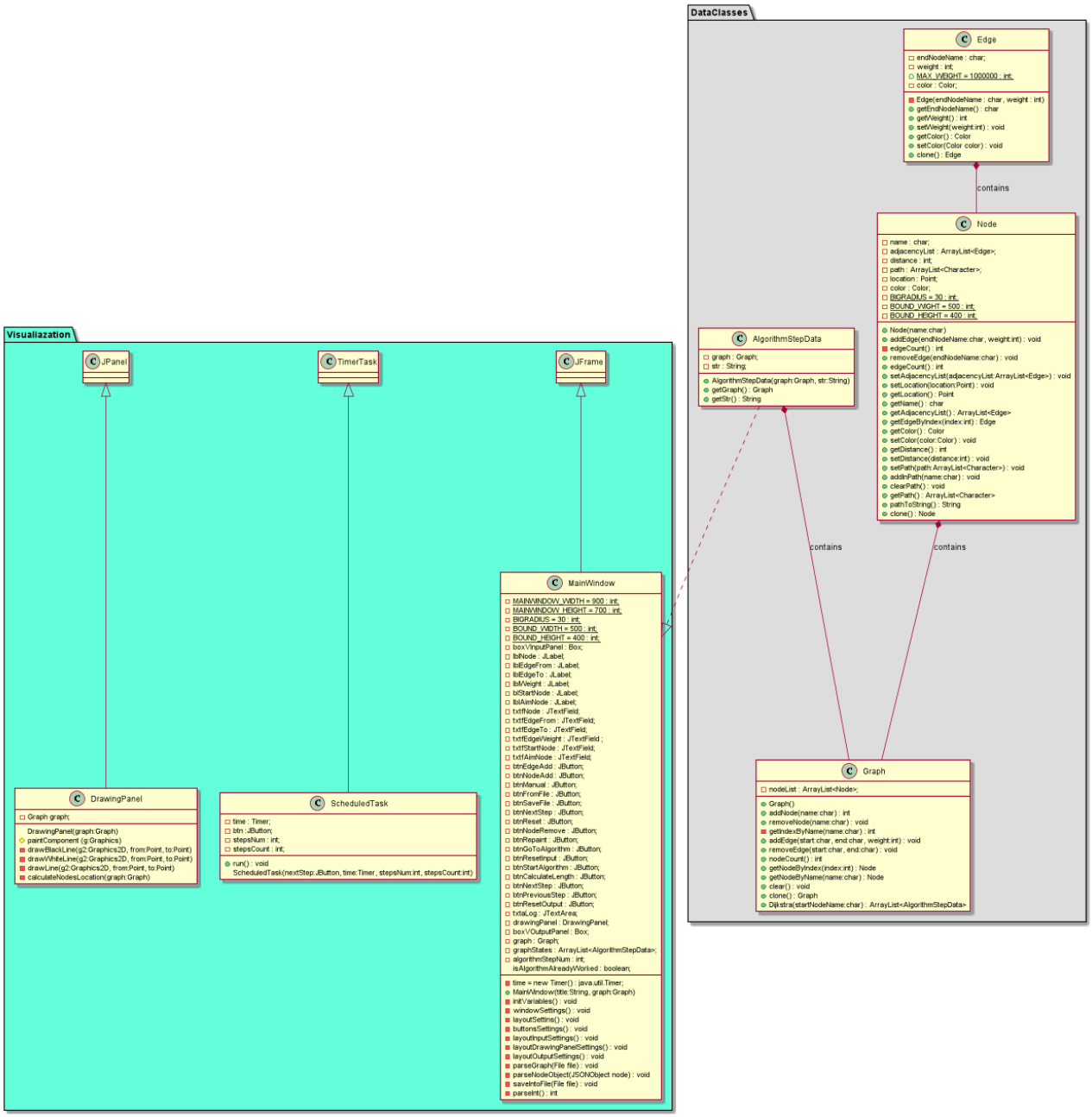

Рисунок 3

2) Диаграмма use-case

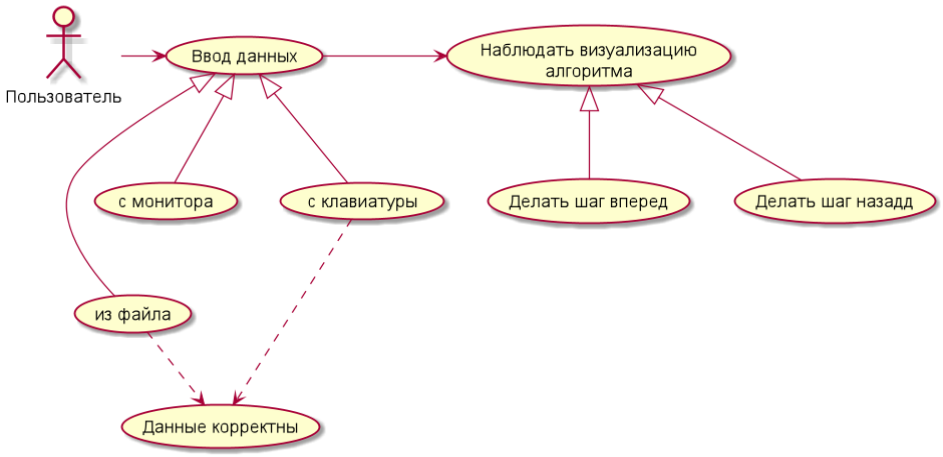

Рисунок 4

3) Диаграмма деятельности

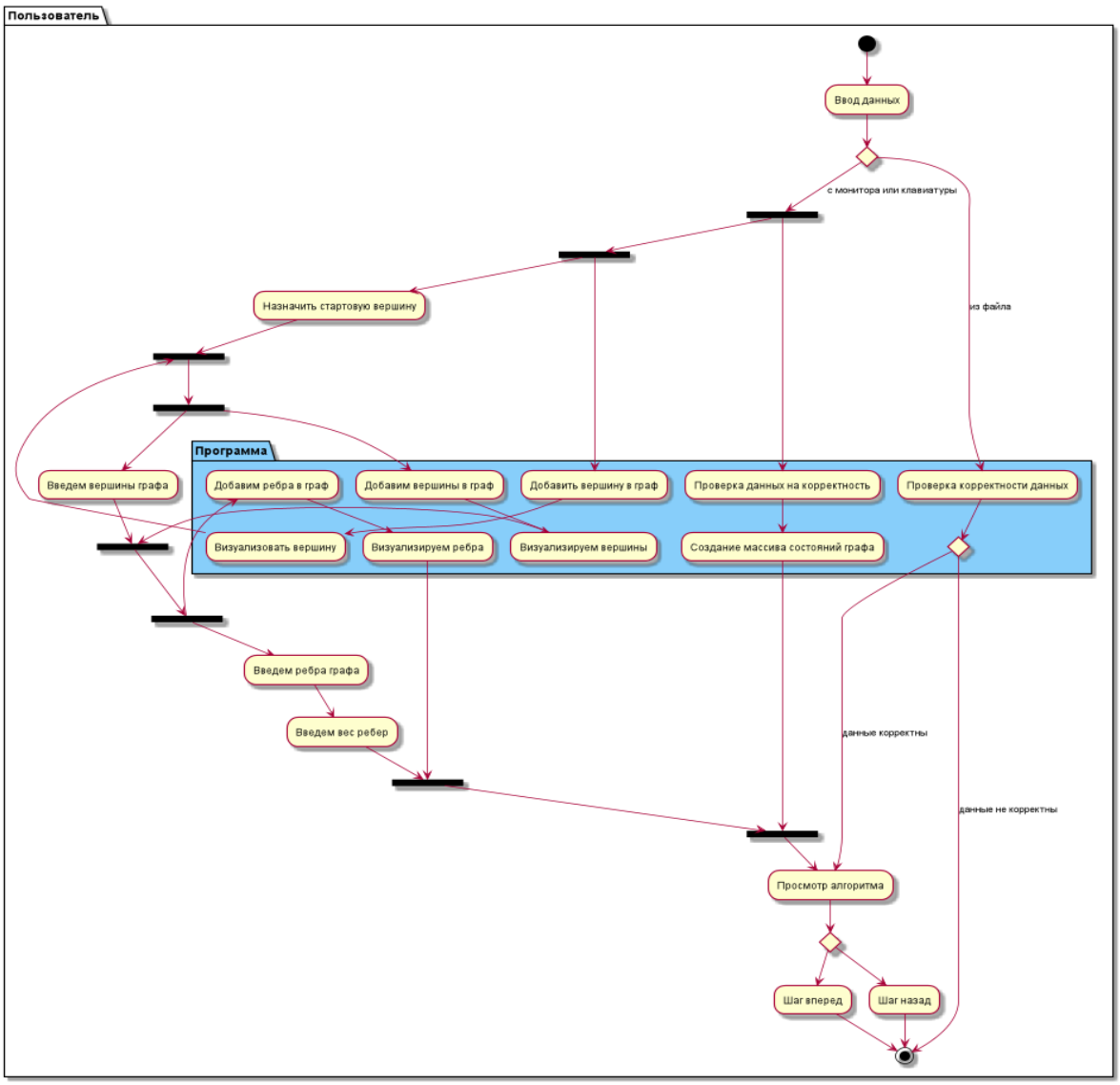

Рисунок 5

1.2. Уточнение требований после сдачи прототипа

1.2.1 Уточнение требований к интерфейсу

В ходе разработки прототипа интерфейса была добавлена возможность сохранения графа в файл, и было улучшено считывание графа с файла с помощью библиотеки Json. Также были убраны полосы прокрутки.

1.2.2. Уточнение требований после сдачи первой версии.

В ходе сдачи первой версии были уточнены следующие требования:

1) был реализован вывод сообщения о том, что вершина уже существует, при попытке создания вершины с одинаковым названием;

2) была создана кнопка, для автоматического показа алгоритма;

3) была добавлена справка, которая описывает основные возможности работы с приложением.

4) была добавлена возможность удаления вершины.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

02.07.2019 — 04.07.2019 — разработка спецификации, согласование спецификации с руководителем, реализация некоторых отдельных частей программы (представление графа).

04.07.2019 — 08.07.2019 — разработка части визуализации, ответственной за представление графа и редактирование графа; разработка структуры проекта и разделение процесса работы по разным классам; разработка графической части визуализации, ответственной за отрисовку шагов алгоритма; подготовка к сдаче 1-ой версии проекта.

08.07.2019 — 10.07.2019 — разработка части реализации, ответственной за считывание графа из файла, сохранение графа в файл; исправление недочётов, выявленных при сдаче 1-ой версии проекта; тестирование проекта; подготовка проекта к финальной сдачи.

2.2. Распределение ролей в бригаде

1. Архитектура – Алясова А.Н.
2. Реализация интерфейса – Адамов Я.В.
3. Визуализация хода работы алгоритма – Габов Е.С.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Описание структур данных и их методов

При реализации алгоритма Дейкстры как алгоритма нахождения кратчайшего пути в ориентированном взвешенном графе использовались следующие структуры данных:

1. Класс `Graph` – это класс, предназначенный для хранения графа. Он содержит следующие переменные и методы.

Переменные:

`private ArrayList<Node> nodeList;` - переменная для хранения списка вершин графа.

Методы:

`1) public Graph()` – конструктор.

Принимаемые аргументы: нет.

Возвращаемое значение: нет.

`2) public int addNode(char name)` – метод, предназначенный для добавления вершины в граф.

Принимаемые аргументы:

`char name;` – имя вершины.

Возвращаемое значение:

`int` - индекс добавленной вершины;

`3) public void removeNode(char name)` – метод для удаления вершины из графа.

Принимаемые аргументы:

`char name;` – имя вершины.

Возвращаемое значение:

Метод ничего не возвращает.

`4) private int getIndexByName(char name)` – метод для того, чтобы узнать индекс заданной вершины.

Принимаемые аргументы:

`char name;` – имя вершины.

Возвращаемое значение:

`int` - индекс вершины с именем name в nodeList, если вершины не существует, тогда -1.

`5) public void addEdge(char start, char end, int weight)` – метод для того, чтобы добавить ребро в граф.

Принимаемые аргументы:

`char start;` – имя вершины из которой ребро выходит.

`char end;` – имя вершины, в которую ребро входит.

`int weight;` – вес ребра.

Возвращаемое значение:

Метод ничего не возвращает.

`6) public void removeEdge(char start, char end)` – метод для удаления ребра из графа.

Принимаемые аргументы:

`char start;` – имя вершины из которой ребро выходит.

`char end;` – имя вершины, в которую ребро входит.

Возвращаемое значение:

Метод ничего не возвращает.

`7) public int nodeCount()` – метод, который вычисляет количество вершин в графе.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`int` – количество вершин в графе.

`8) public Node getNodeByIndex(int index)` – метод, который находит вершину по ее индексу.

Принимаемые аргументы:

`int index;` - индекс вершины.

Возвращаемое значение:

`Node` – геттер для поля `index`.

`9) public Node getNodeByName(char name)` – метод, который возвращает вершину по имени.

Принимаемые аргументы:

`char name;` - имя вершины.

Возвращаемое значение:

`Node` – вершина.

`10) public void clear()` – метод для очистки данных графа.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

Метод ничего не возвращает.

`11) public Graph clone()` – метод для того, чтобы создать копию графа и передавать его не по ссылке, а по значению.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

Graph – копия графа.

`12) public ArrayList<AlgorithmStepData> Dijkstra(char startNodeName)`

Принимаемые аргументы:

`char startNodeName;` – имя стартовой вершины.

Возвращаемое значение:

`ArrayList<AlgorithmStepData>` - массив из состояний графа на каждом шаге алгоритма.

2. Класс `Edge` – класс для хранения ребер. У него есть следующие методы и переменные.

Переменные:

`private char endNodeName;` - имя вершины, в которую ребро входит.

`private int weight;` - вес ребра.

`public static final int MAX_WEIGHT = 1000000;` - максимальный допустимый вес ребра.

`private Color color;` - цвет ребра.

Методы:

`1) public Edge(char endNodeName, int weight)` – конструктор.

Принимаемые аргументы:

`char endNodeName;` - имя вершины, в которую ребро входит.

`int weight;` – вес ребра.

Возвращаемое значение: нет.

`2) public char getEndNodeName()` – метод, который возвращает имя вершины, в которую ребро входит.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`char` - имя вершины.

`3) public int getWeight()` – метод, который возвращает вес ребра.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`int` – вес ребра.

`4) public void setWeight(int weight)` – метод, который устанавливает вес ребра.

Принимаемые аргументы:

`int weight` – вес ребра.

Возвращаемое значение:

Метод ничего не возвращает.

`5) public Color getColor()` – метод, который возвращает цвет ребра.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`Color` – цвет ребра.

6) `public void setColor(Color color)` – метод, который устанавливает цвет ребра.

Принимаемые аргументы:

`Color color;` - цвет ребра.

Возвращаемое значение:

Метод ничего не возвращает.

7) `public Edge clone()` – метод, который создает копию ребра, чтобы передавать ребро не по ссылке, а по значению.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`Edge` – копия ребра.

3. Класс `Node` предназначен для хранения вершин. Его методы и переменные следующие.

Переменные:

`private char name;` - имя вершины.

`private ArrayList<Edge> adjacencyList;` - список смежности.

`private int distance;` - расстояние от стартовой вершины.

`private ArrayList<Character> path;` - путь от стартовой вершины.

`private Point location;` - координаты вершины.

`private Color color;` - цвет вершины.

`public static final int BIGRADIUS = 30;` - максимальный радиус вершины.

`public static final int BOUND = 500;` - граница, после которой нельзя нарисовать вершину.

Методы:

1) `public Node(char name)` – конструктор.

Принимаемые аргументы:

`char name;` – имя вершины.

Возвращаемое значение: нет.

2) `public void addEdge(char endNodeName, int weight)` – метод, который позволяет добавить ребро в список смежности.

Принимаемые аргументы:

`char endNodeName;` - имя вершины, в которую входит ребро.

`int weight;` - вес ребра.

Возвращаемое значение:

Метод ничего не возвращает.

3) `public void removeEdge(char endNodeName)` – метод для удаления ребра в вершине с именем `endNodeName`.

Принимаемые аргументы:

`char endNodeName;` - имя вершины.

Возвращаемое значение:

Метод ничего не возвращает.

4) `public int edgeCount()` – метод, который считает количество рёбер, исходящих из данной вершины.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`int` - количество рёбер.

5) `public void setAdjacencyList(ArrayList<Edge> adjacencyList)` – метод для того, чтобы установить список смежности для вершины.

Принимаемые аргументы:

`ArrayList<Edge> adjacencyList;` - список смежности.

Возвращаемое значение:

Метод ничего не возвращает.

6) `public void setLocation(Point location)` – метод для того, чтобы установить координату для вершины.

Принимаемые аргументы:

`Point location;` - координата.

Возвращаемое значение:

Метод ничего не возвращает.

`7) public Point getLocation()` – метод для того, чтобы получить координату вершины.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`Point` - координата.

`8) public char getName()` – метод для того, чтобы получить имя вершины.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`char` - имя вершины.

`9) public ArrayList<Edge> getAdjacencyList()` – метод для того, чтобы получить список смежности вершины.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`ArrayList<Edge>` - список смежности.

`10) public Edge getEdgeByIndex(int index)` – метод, который возвращает геттер для поля `index`.

Принимаемые аргументы:

`int index;` - индекс ребра.

Возвращаемое значение:

`Edge` – ребро.

`11) public Color getColor()` – метод для того, чтобы получить цвет вершины.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`Color` – цвет вершины;

12) `public void setColor(Color color)` – метод для того, чтобы установить цвет вершины.

Принимаемые аргументы:

`Color color;` - цвет.

Возвращаемое значение:

Метод ничего не возвращает.

13) `public int getDistance()` – метод для того, чтобы получить расстояние.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`int` - расстояние.

14) `public void setDistance(int distance)` – метод для того, чтобы установить расстояние.

Принимаемые аргументы:

`int distance;` - значение расстояния.

Возвращаемое значение:

Метод ничего не возвращает.

15) `public void setPath(ArrayList<Character> path)` – метод, который устанавливает путь.

Принимаемые аргументы:

`ArrayList<Character> path` – путь.

Возвращаемое значение:

Метод ничего не возвращает.

16) `public void addInPath(char name)` – метод, который добавляет вершину при записи пути.

Принимаемые аргументы:

`char name;` – имя вершины, которую нужно добавить в путь.

Возвращаемое значение:

Метод ничего не возвращает.

17) `public void clearPath()` – метод, который очищает данные о пути.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

Метод ничего не возвращает.

`18) public ArrayList<Character> getPath()` – метод для получения пути.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`ArrayList<Character>` - путь.

`19) public String pathToString()` – метод, который переводит путь из массива char в строку.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`String` - строка, в которой содержится путь.

`20) public Node clone()` - метод, который создает копию вершины, чтобы передавать ее не по ссылке, а по значению.

Принимаемые аргументы:

Метод ничего не принимает.

Возвращаемое значение:

`Node` - копия вершины.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

1) Правильность разметки

При подключении изображения графа к интерфейсу не возникает проблем с существующей разметкой окна: поле для графа сохраняет свой размер, не смещаются другие компоненты окна (см. рис. 6,7).

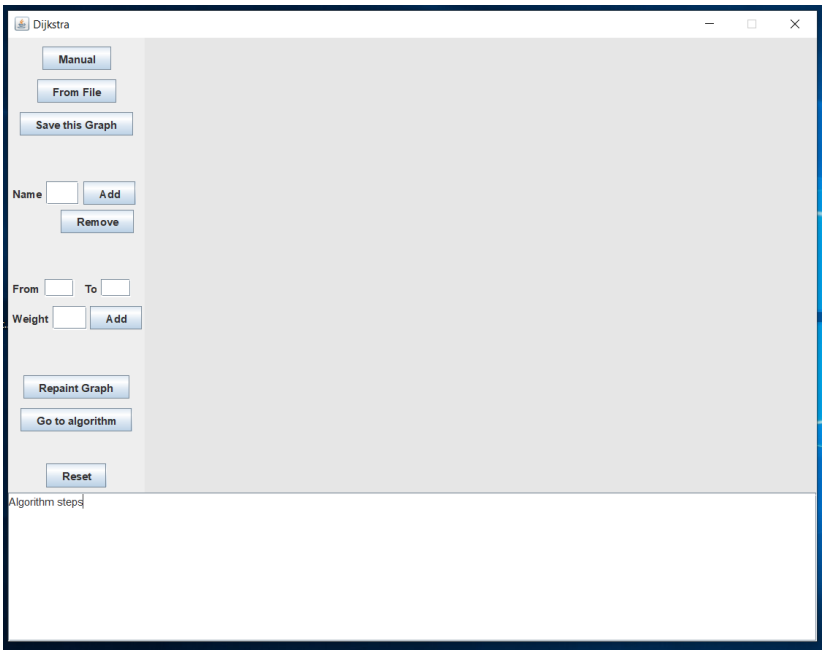

Рисунок 6

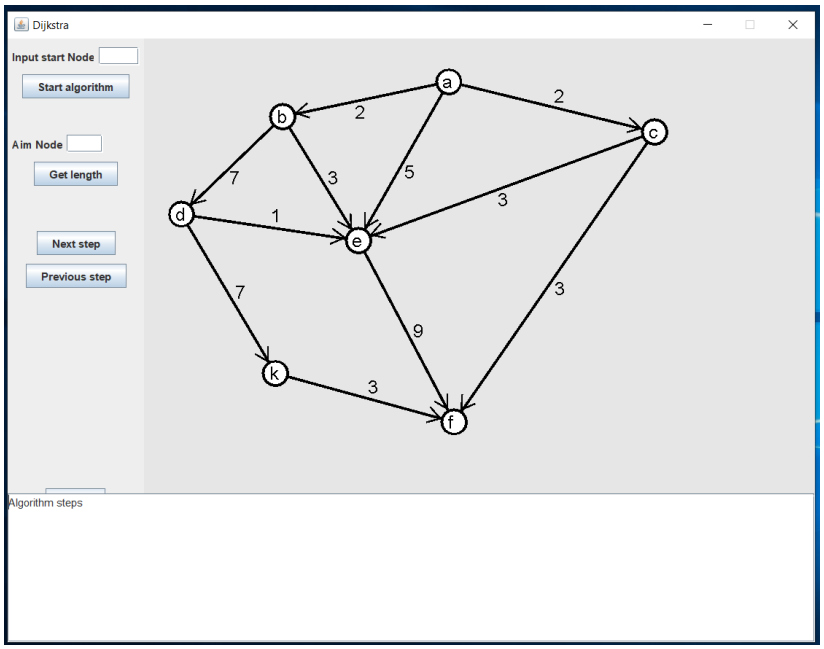

Рисунок 7

2) Правильность отображения графа.

Вне зависимости от размера графа, он отображается корректно, все вершины подписаны, ребра имеют подписи с соответствующими весами (см. рис. 7, 8).

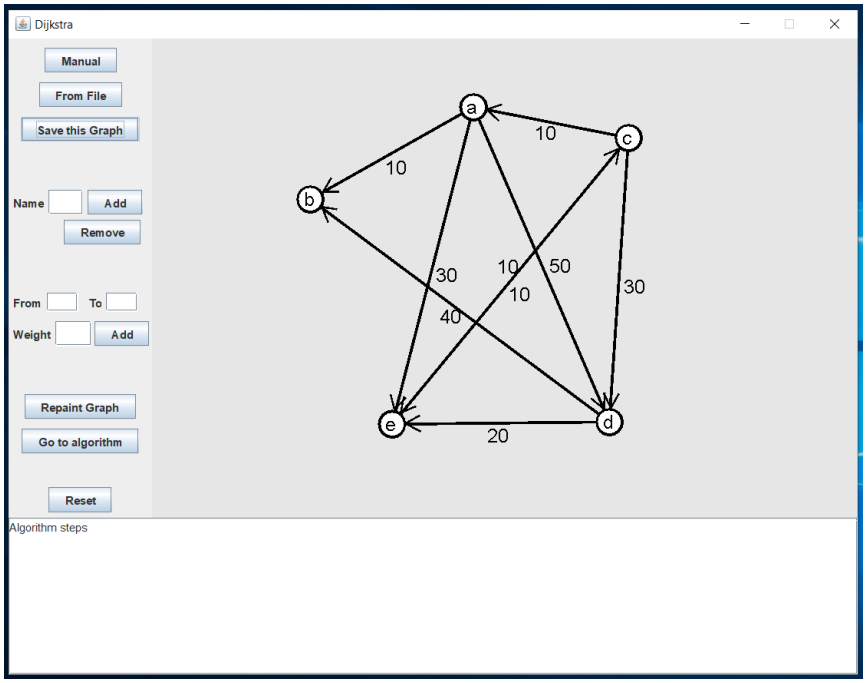

Рисунок 8

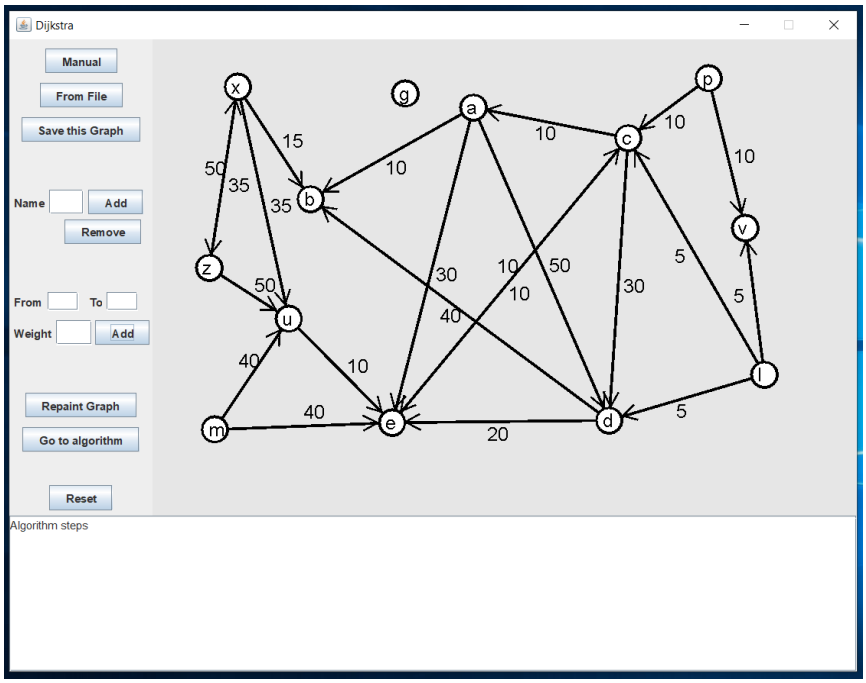

Рисунок 9

3) Возможности при работе с графом

- Построение ребер

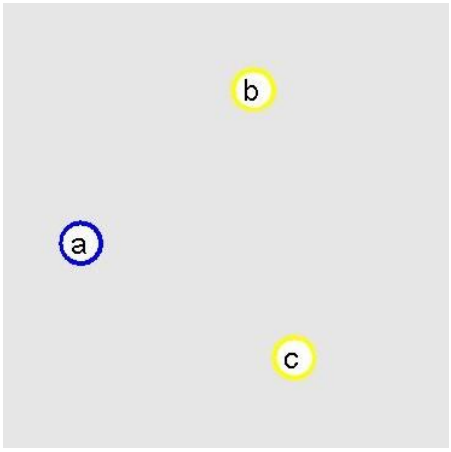

Рисунок 10

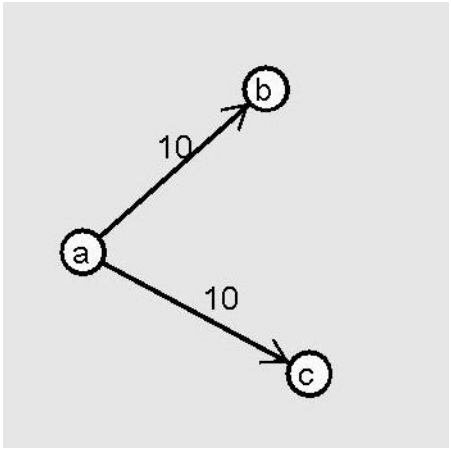

Рисунок 11

- Изменение веса ребра

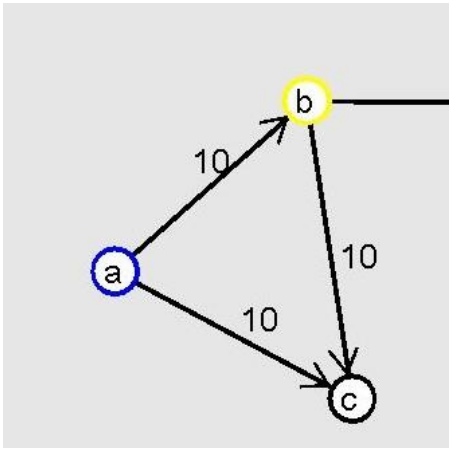

Рисунок 12

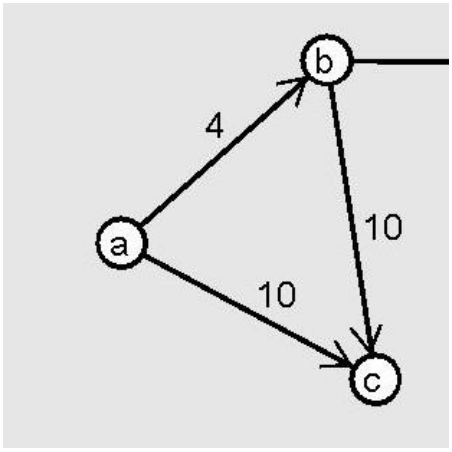

Рисунок 13

- Удаление вершины

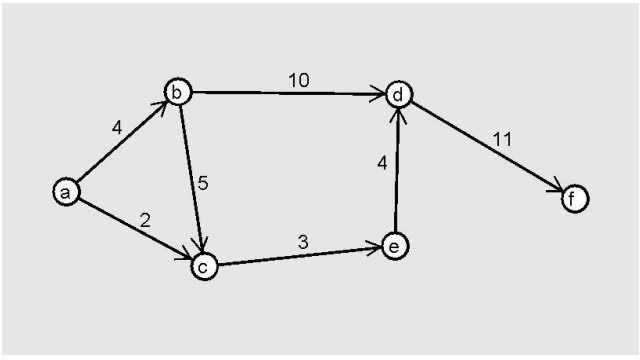

Рисунок 14

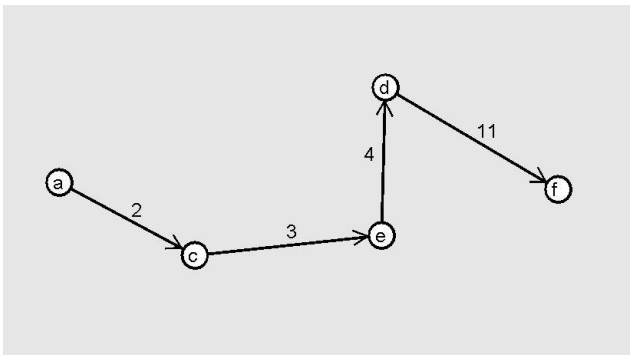

Рисунок 15

- Возможность передвинуть вершину

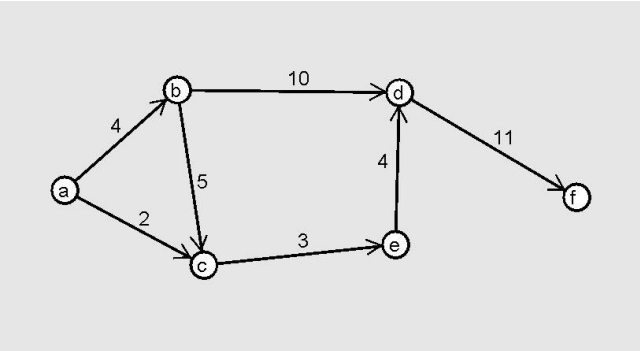

Рисунок 16

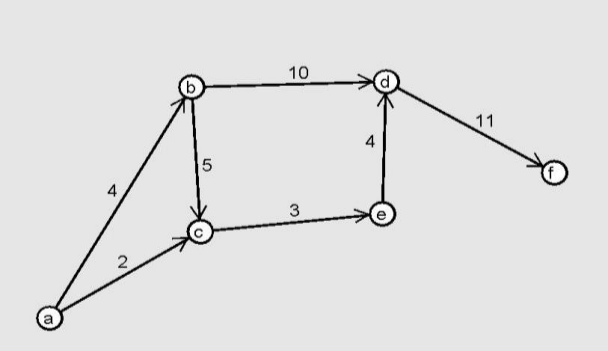

Рисунок 17

4) Демонстрация пошаговой визуализации алгоритма

У вершины есть 5 состояний:

Серый цвет – вершина не просматривалась;

Желтый цвет – вершина в очереди на раскрытие;

Красный цвет – вершина раскрывается на данном шаге;

Зеленый цвет – релаксируемая вершина;

Черный цвет – вершина полностью обработана.

Шаги просматриваются слева на право.

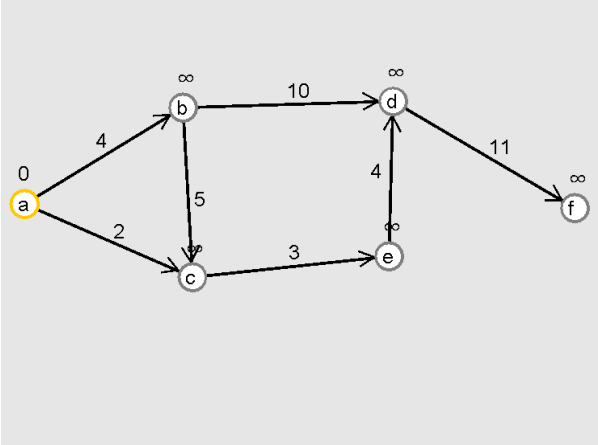

Рисунок 18 - Шаг 1

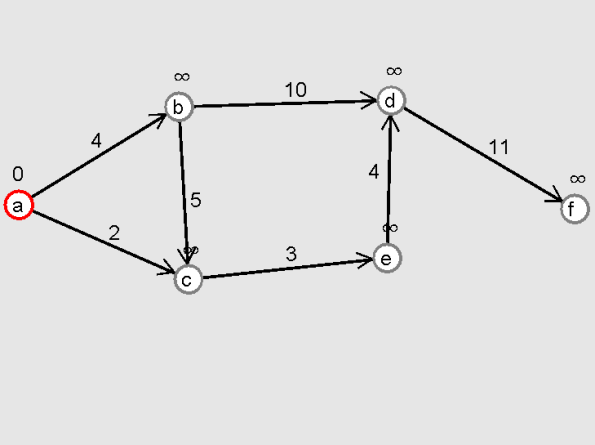

Рисунок 19 - Шаг 2

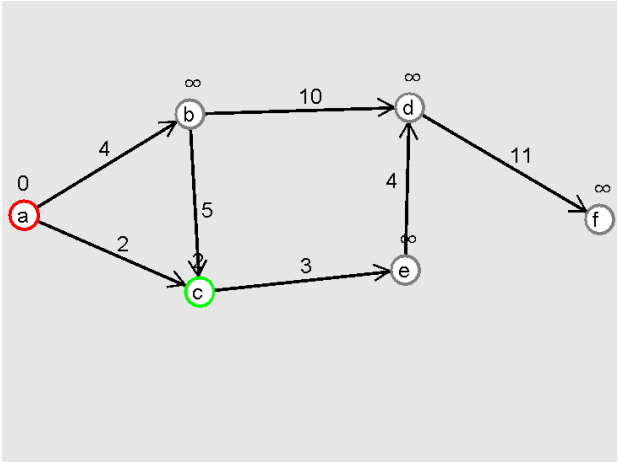

Рисунок 20 - Шаг 3

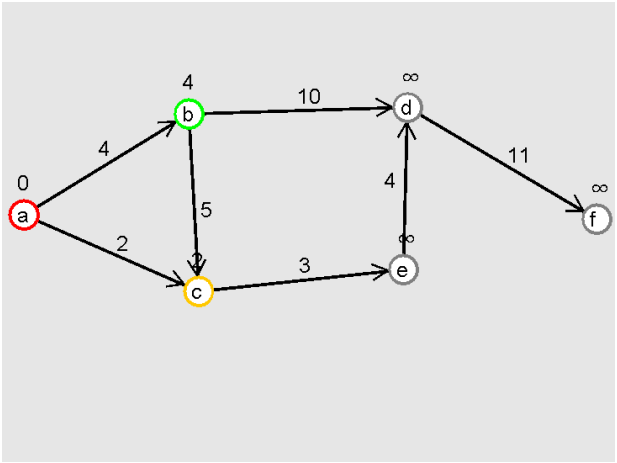

Рисунок 21 - Шаг 4

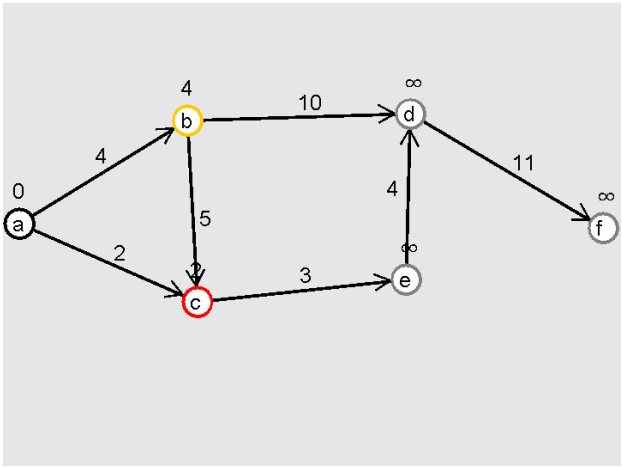

Рисунок 22 - Шаг 5

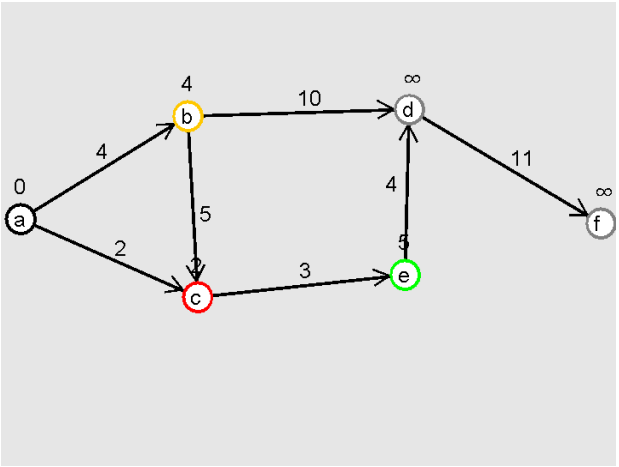

Рисунок 23 - Шаг 6

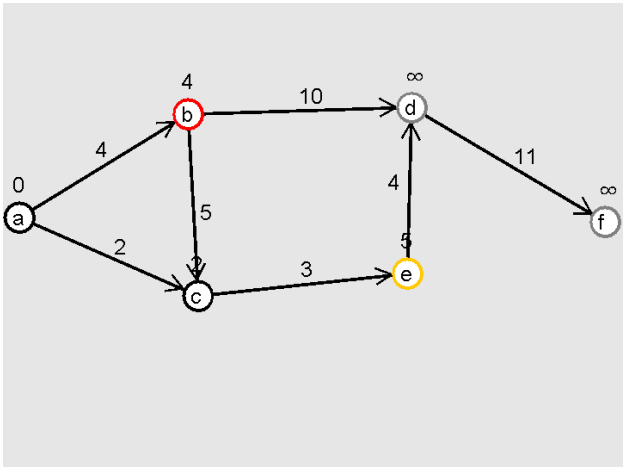

Рисунок 24 - Шаг 7

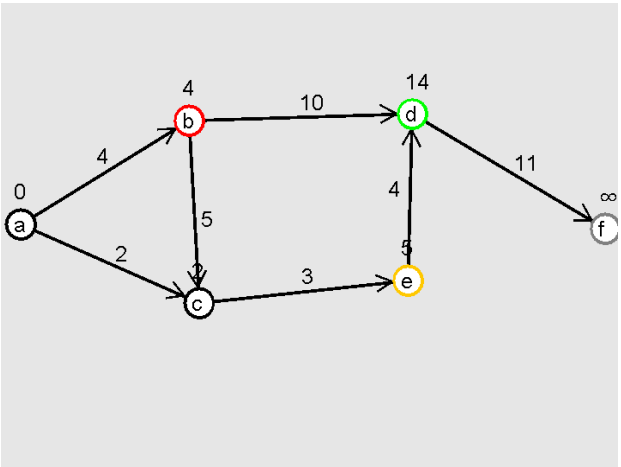

Рисунок 25 - Шаг 8

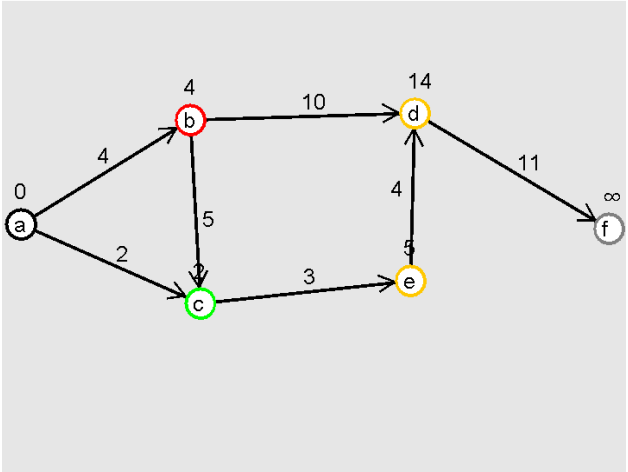

Рисунок 26 - Шаг 9

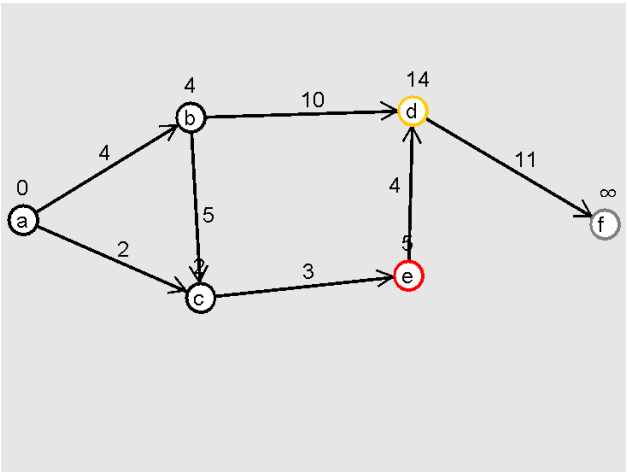

Рисунок 27 - Шаг 10

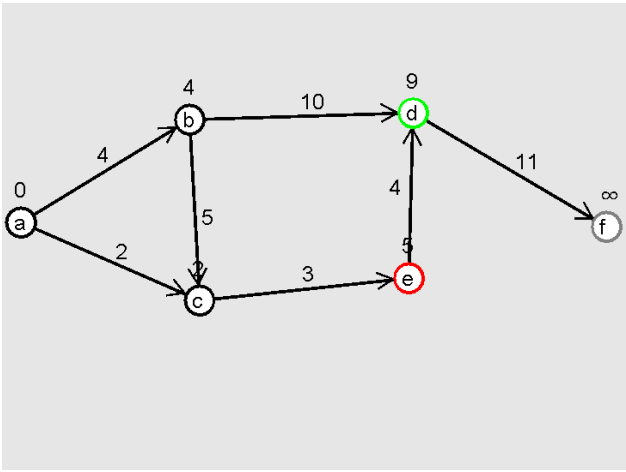

Рисунок 28 - Шаг 11

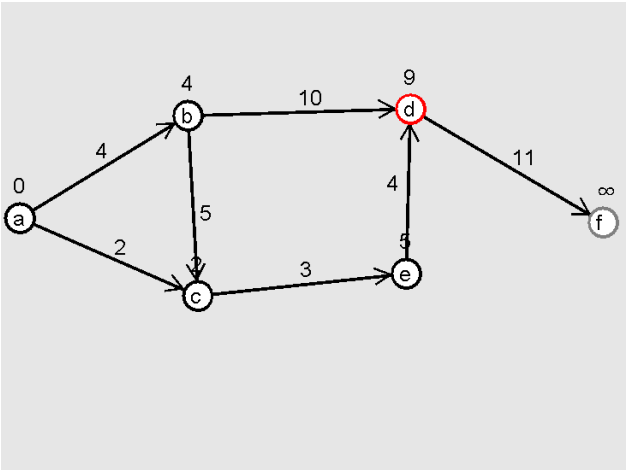

Рисунок 29 - Шаг 12

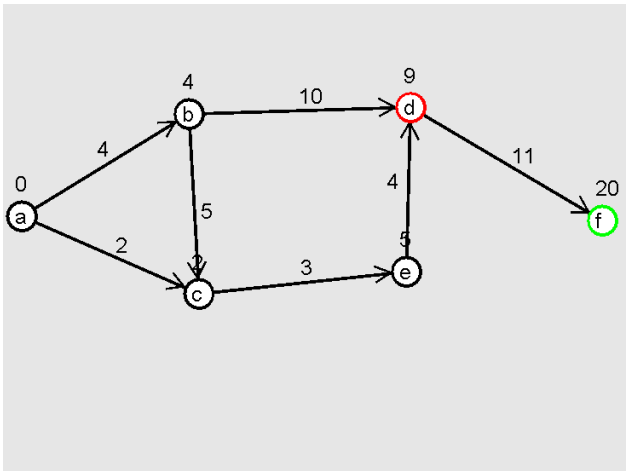

Рисунок 30 - Шаг 13

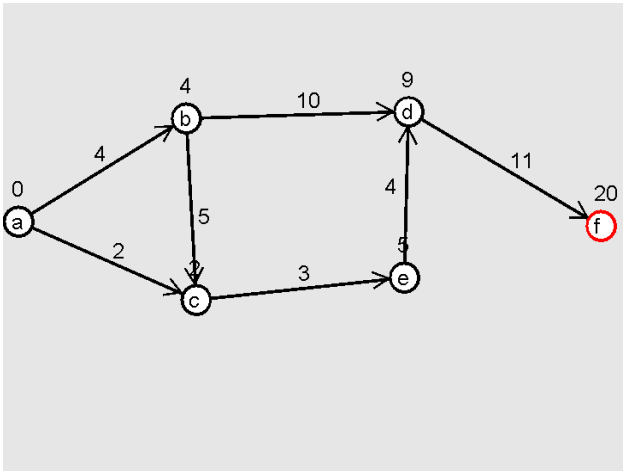

Рисунок 31 - Шаг 14

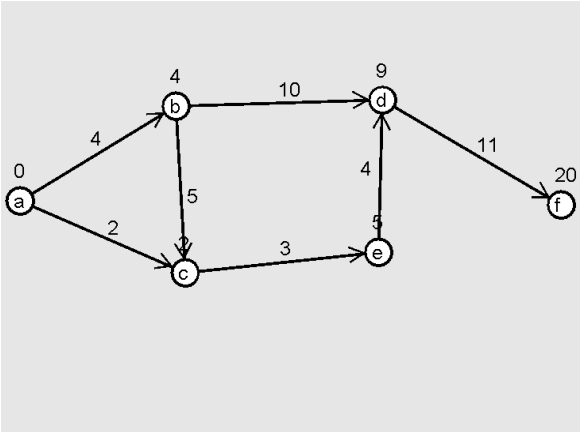

Рисунок 32 - Шаг 15

Результат:

Путь от вершины 'a' до вершины

'a': a (расстояние — 0).

'd': a->c->e->d (расстояние — 9).

'f': a->c->e->d->f (расстояние — 20).

'e': a->c->e (расстояние — 5).

'c': a->c (расстояние — 2).

'b': a->b (расстояние — 4).

4.2. Тестирование кода алгоритма

1) Тестирование на корректных входных данных

Таблица 1

Входные данные	Выходные данные
a b 10 a e 30 a d 50 c a 10 c d 30 d b 40 d e 20 c e 10 e c 10 a	Путь от вершины 'a' до вершины 'b': a->b (расстояние — 10). 'd': a->d (расстояние — 50). 'e': a->e (расстояние — 30). 'a': a (расстояние — 0). 'c': a->e->c (расстояние — 40).

Продолжение таблицы 1

a b 21 a f 3 b c 6 b e 10 b f 4 c d 12 e d 4 e c 18 f e 2 f b 9 a	Путь от вершины 'a' до вершины 'a': a (расстояние — 0). 'c': a->f->b->c (расстояние — 13). 'd': a->f->e->d (расстояние — 9). 'e': a->f->e (расстояние — 5). 'f': a->f (расстояние — 3). 'b': a->f->b (расстояние — 7).
a b 5 a d 3 a c 6 d c 4 c b 1 a	Путь от вершины 'a' до вершины 'a': a (расстояние — 0). 'b': a->b (расстояние — 5). 'c': a->c (расстояние — 6). 'd': a->d (расстояние — 3).
a b 1 b a 1 b c 1 c b 1 a c 1 c a 1 a	Путь от вершины 'a' до вершины 'a': a (расстояние — 0). 'b': a->b (расстояние — 1). 'c': a->c (расстояние — 1).
a e 5 a b 2 a c 2 b e 3 b d 7 d e 1 d k 7 k f 3 e f 9 c f 3 c e 3 c	Путь от вершины 'c' до вершины 'a': вершина недостежима. 'b': вершина недостежима. 'c': c (расстояние — 0). 'd': вершина недостежима. 'e': c->e (расстояние — 3). 'f': c->f (расстояние — 3). 'k': вершина недостежима.

Окончание таблицы 1

n c a 4 a c 4 c b 2 a b 3 c	Путь от вершины 'c' до вершины 'a': c->a (расстояние — 4). 'b': c->b (расстояние — 2). 'c': c (расстояние — 0). 'n': вершина недостежима.
q u 3 d z 5 r t 14 d l 98 y q 15 f d 5 y s 15 i d 10 s k 2 I g 10 k j 7 I o 10 j k 7 w d j v 7 v b 2 b n 3 m n 5 m z 5 z m 8 l z 7 l b 8 l f 78 e o 7 e r 7 e a 7 q s 555 r e 7 t c 88 r o 5 r h 14 c x 8 h x 3 h c 3 x g 63 x p 63 o p 11 u w 10 000 q	Путь от вершины 'q' до вершины 'q': q (расстояние — 0). 'w': q->u->w (расстояние — 10003). 'e': вершина недостежима. 'r': вершина недостежима. 't': вершина недостежима. 'y': q->u->y (расстояние — 4). 'u': q->u (расстояние — 3). 'i': q->u->w->d->l->f->i (расстояние — 10186). 'o': q->u->w->d->l->f->i->o (расстояние — 10196). 'p': q->u->w->d->l->f->i->o->p (расстояние — 10207). 'a': вершина недостежима. 's': q->u->y->s (расстояние — 19). 'd': q->u->w->d (расстояние — 10005). 'f': q->u->w->d->l->f (расстояние — 10181). 'g': q->u->w->d->l->f->i->g (расстояние — 10196). 'h': вершина недостежима. 'j': q->u->y->s->k->j (расстояние — 28). 'k': q->u->y->s->k (расстояние — 21). 'l': q->u->w->d->l (расстояние — 10103). 'z': q->u->y->s->k->j->v->b->n->z (расстояние 46). 'x': вершина недостежима. 'c': вершина недостежима. 'v': q->u->y->s->k->j->v (расстояние — 35). 'b': q->u->y->s->k->j->v->b (расстояние — 37). 'n': q->u->y->s->k->j->v->b->n (расстояние —40) 'm': q->u->y->s->k->j->v->b->n->z->m (расстояние — 54).

2) Тестирование при некорректных входных данных

Таблица 2

Входные данные	Выходные данные
a b 21 a f 3 b c 6 b e 10 b f -2 a	number must be >= 0!
c a 4 a c 4 c b 2 a b p c	Edge weight must be only integer!
c a 4 a c 0 c	number must be >= 0!
a a	This node is already in graph
a b 4 a c 5 <>	Input start node name

ЗАКЛЮЧЕНИЕ

В результате выполнения учебной практики была разработана и протестирована программа, реализующая алгоритм Дейкстры и наглядно демонстрирующая принцип его работы.

Во время реализации проекта были проведены некоторые доработки интерфейса программы для улучшения её удобства.

В ходе работы было проведено тестирование с целью выявления возможных ошибок. По результатам было выяснено, что программа работает корректно и успешно справляется со своей задачей.

Выполнение данного проекта позволило приобрести навыки, необходимые для будущей профессии, тесно связанной с IT. Это навыки:

- разработки в среде программирования Java;
- работы в команде;
- использования известной системы контроля версий GitHub;
- использования библиотеки Swing для реализации графического интерфейса.

Таким образом, цели практики успешно достигнуты, и по окончании разработки получен корректно работающий визуализатор алгоритма Дейкстры на языке программирования Java.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кей С. Хорстманн “Java. Библиотека профессионала, том 1. Основы. 10-е издание”. Издательский дом "Вильямс", 2016.

2. https://docs.oracle.com (дата обращения: 06.07.2019).

3. https://refactoring.guru/ru/design-patterns/memento (дата обращения: 05-08.07.2019).

4. http://plantuml.com (дата обращения 05-06.07.2019).

5. https://ru.wikipedia.org/wiki/UML (дата обращения 05-06.07.2019).