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.

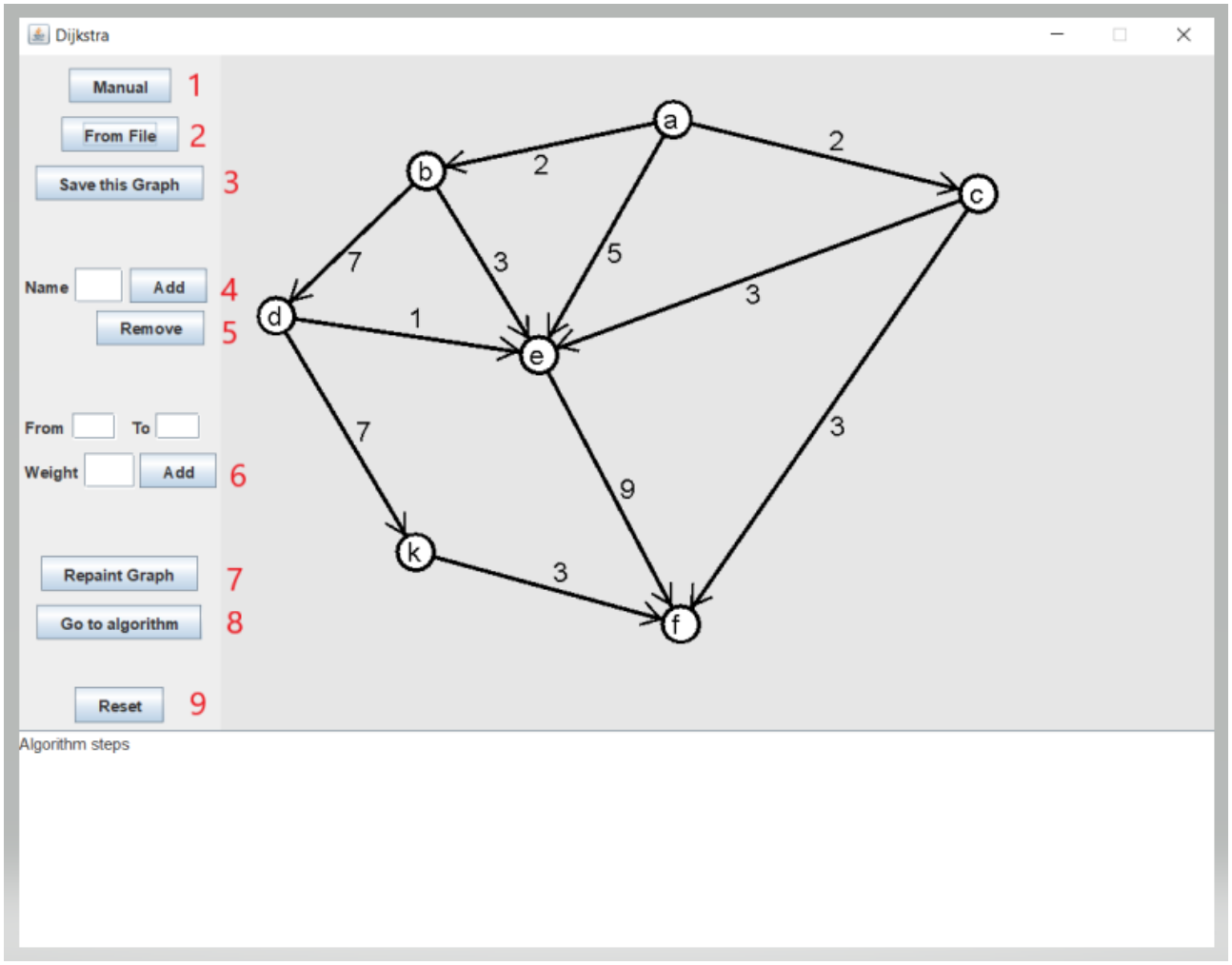

Рисунок 1

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.

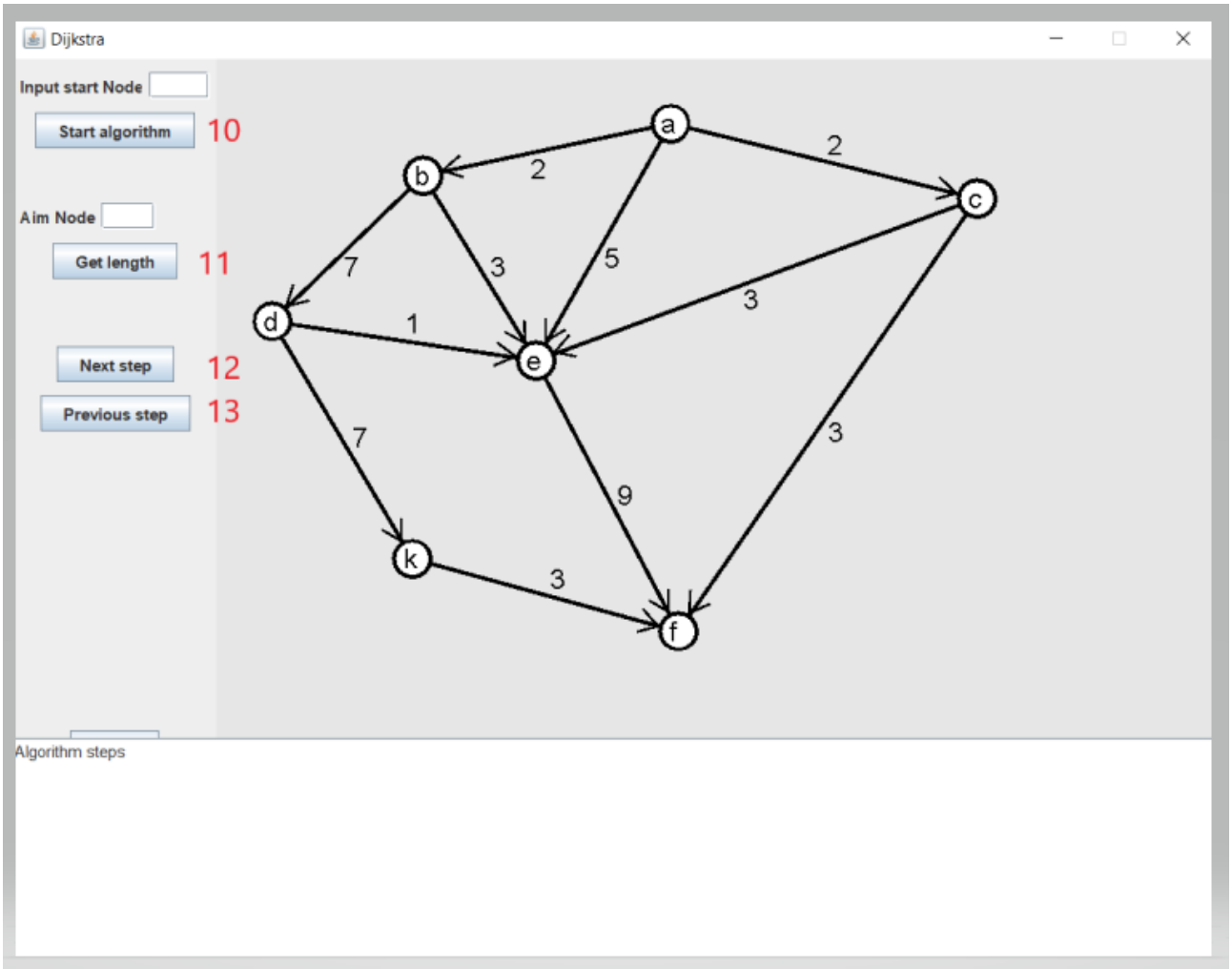

Рисунок 2

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

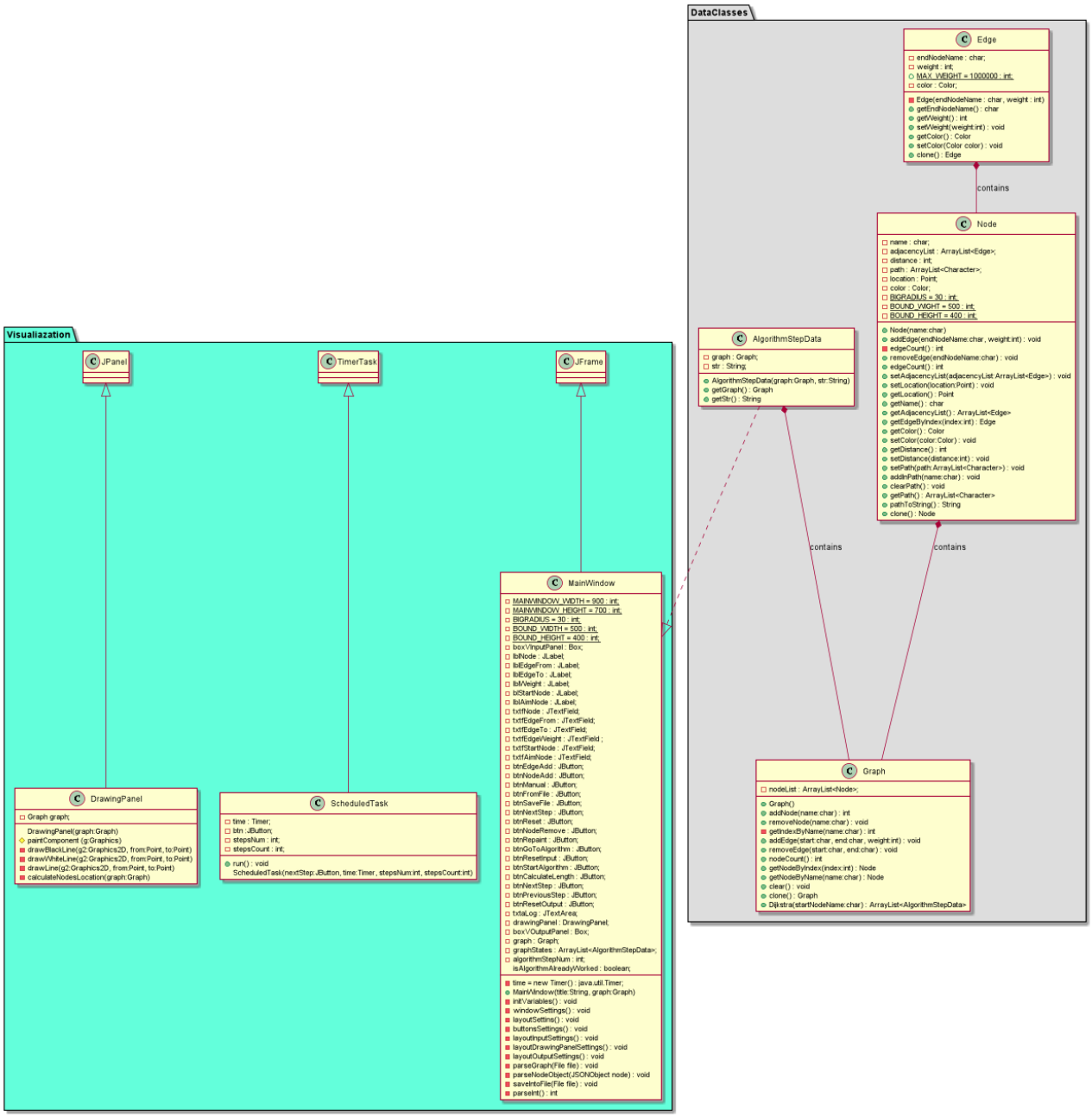

Рисунок 3

2) Диаграмма use-case

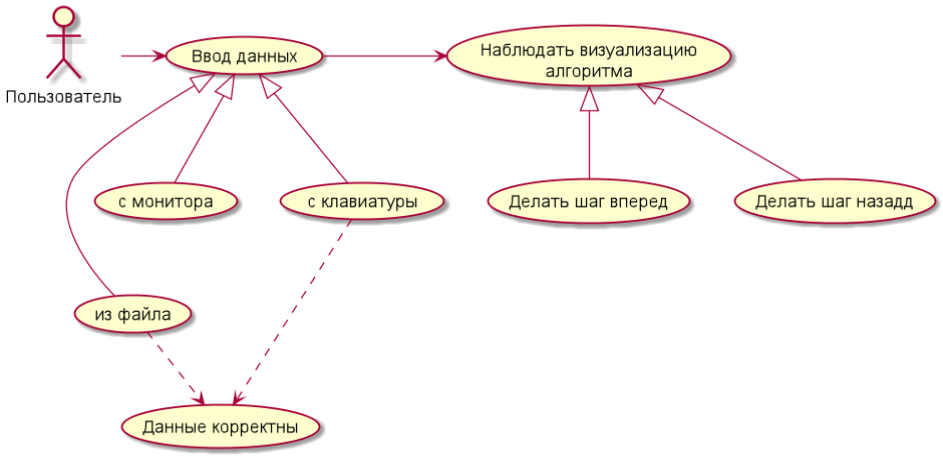

Рисунок 4

3) Диаграмма деятельности

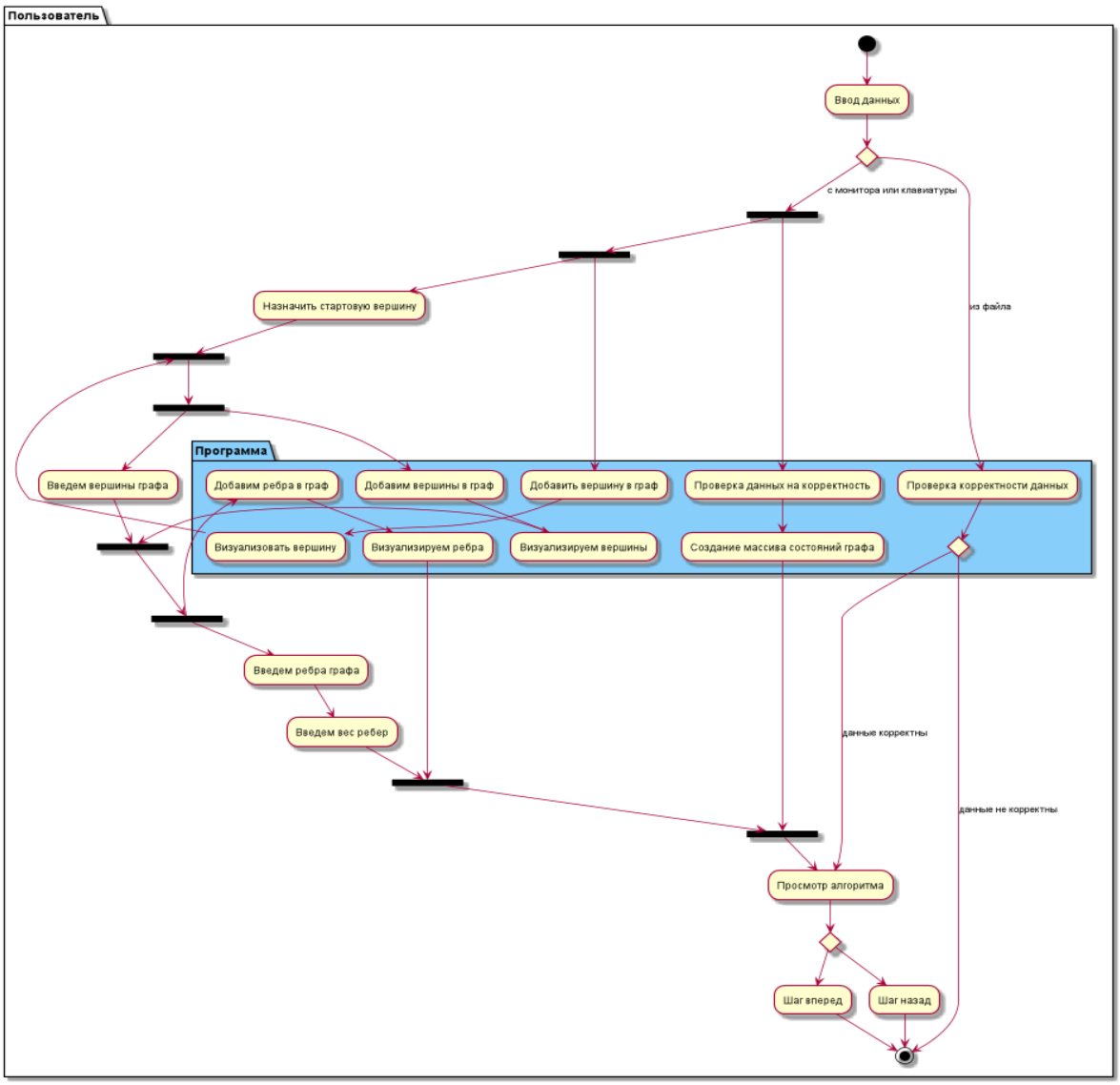

Рисунок 5

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. ТЕТИРОВАНИЕ

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).

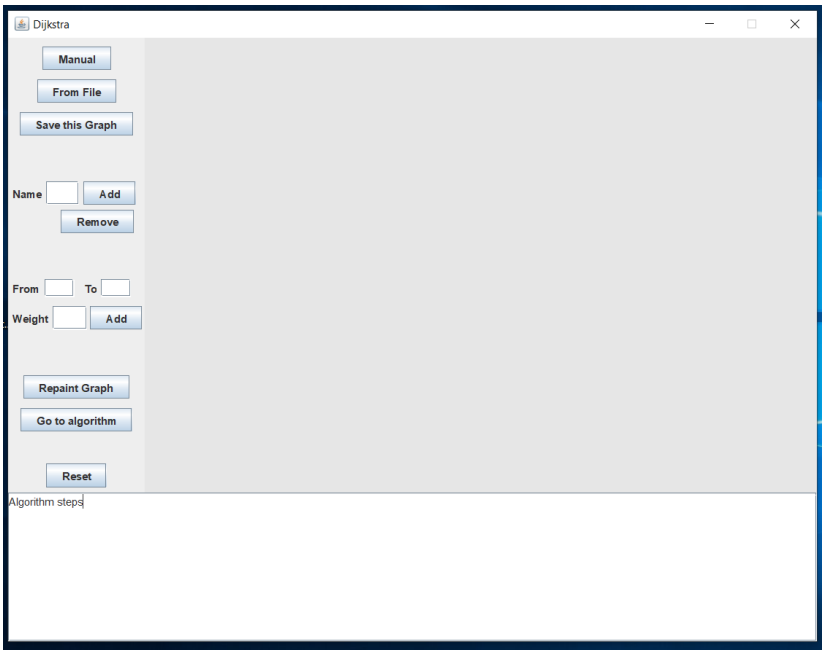

Рисунок 6

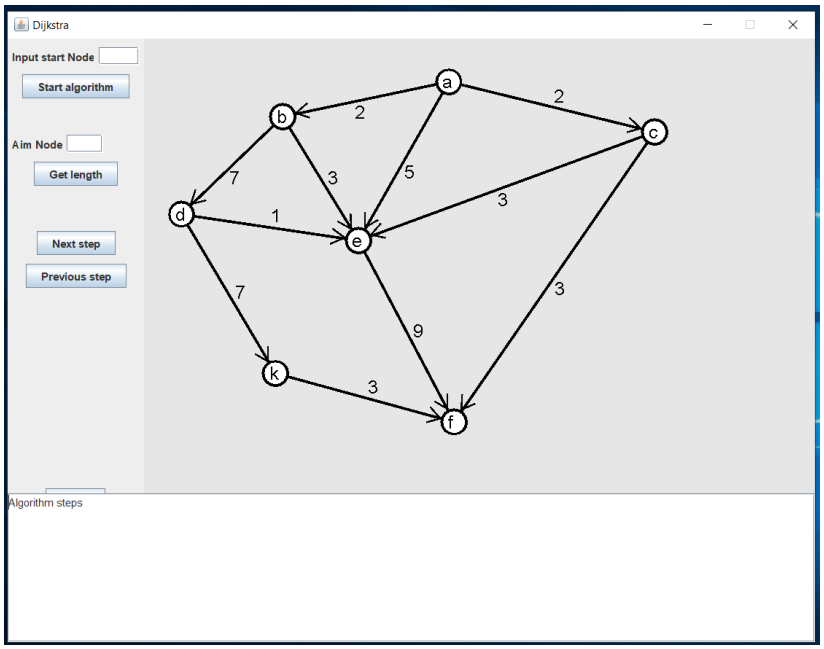

Рисунок 7

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).

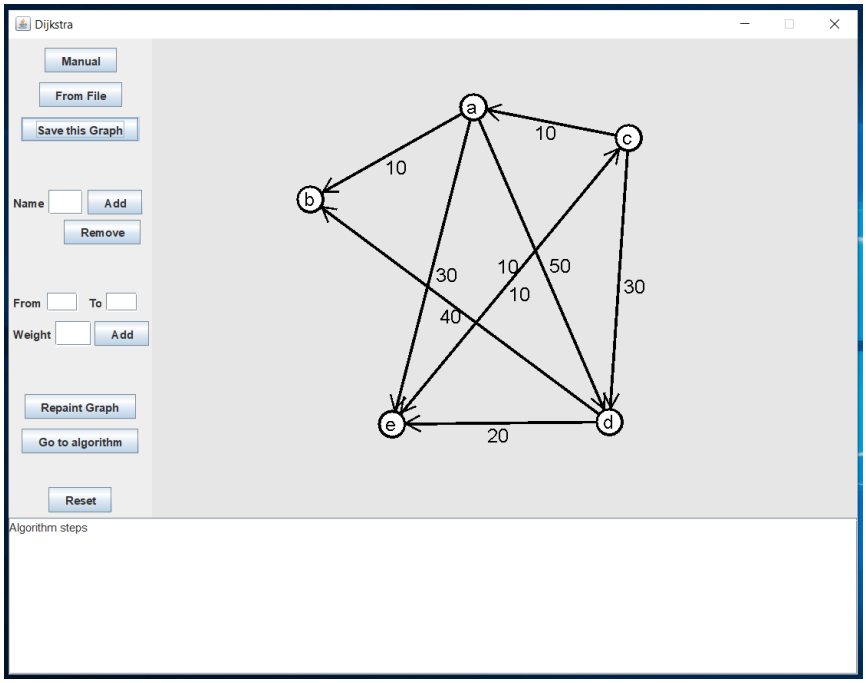

Рисунок 8

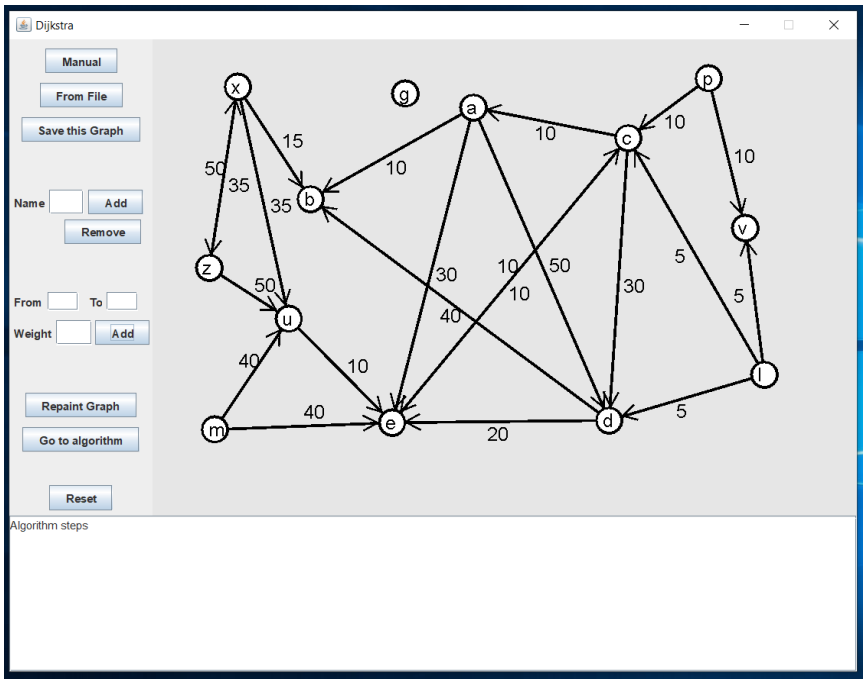

Рисунок 9

3) Возможности при работе с графом

- Построение ребер

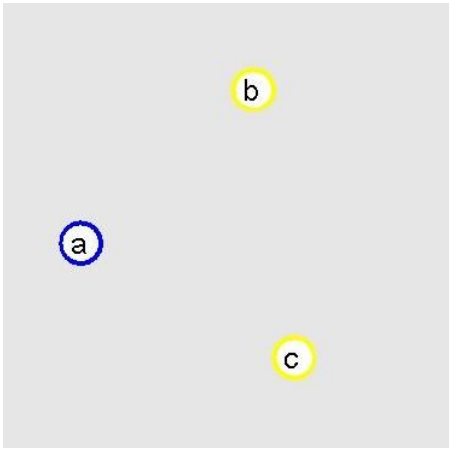

Рисунок 10

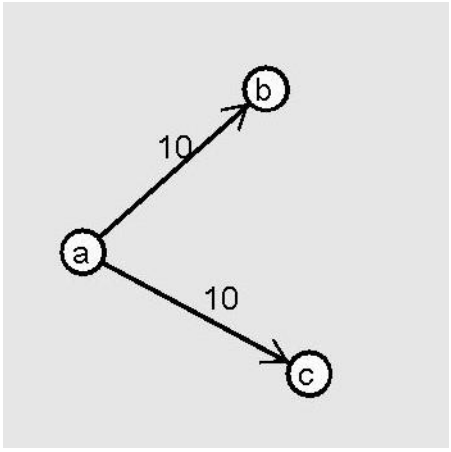

Рисунок 11

- Изменение веса ребра

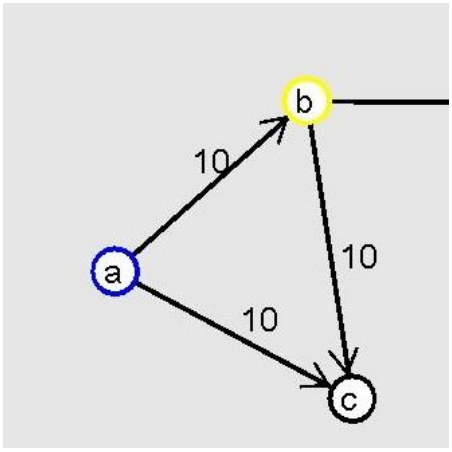

Рисунок 12

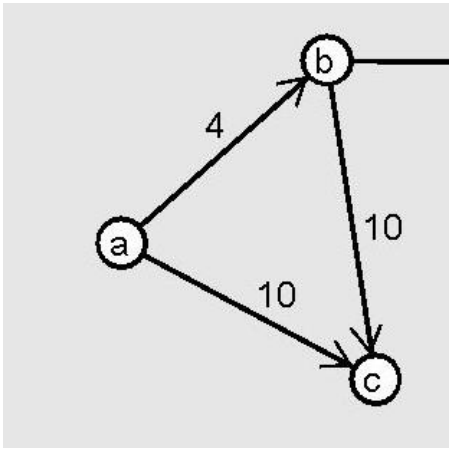

Рисунок 13

- Удаление вершины

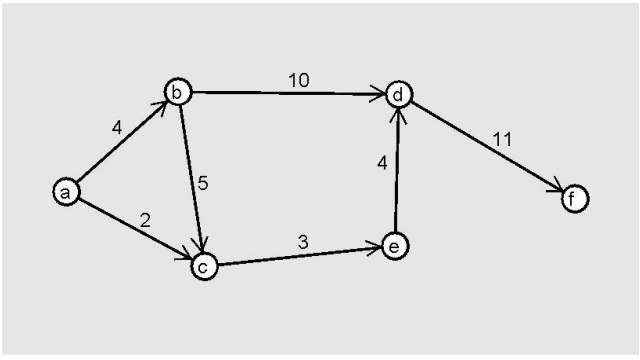

Рисунок 14

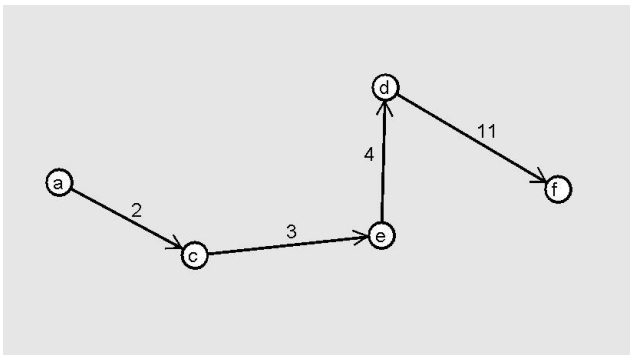

Рисунок 15

- Возможность передвинуть вершину

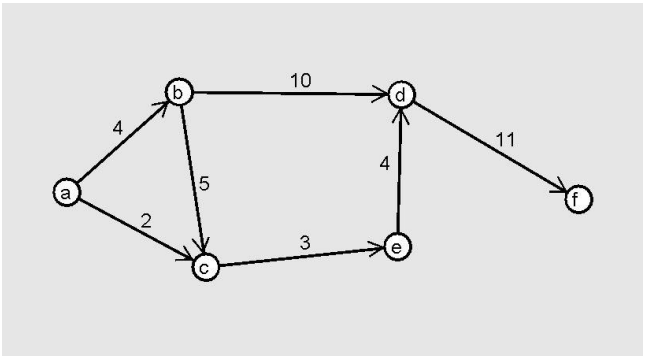

Рисунок 16

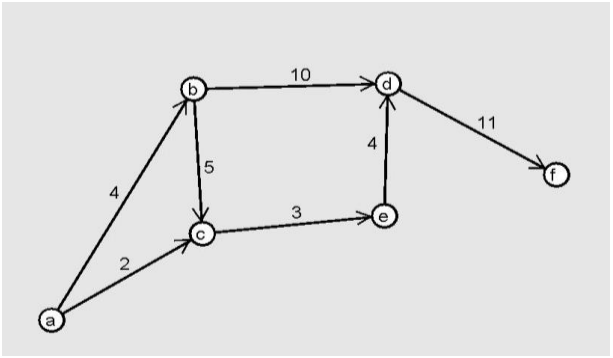

Рисунок 17

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.

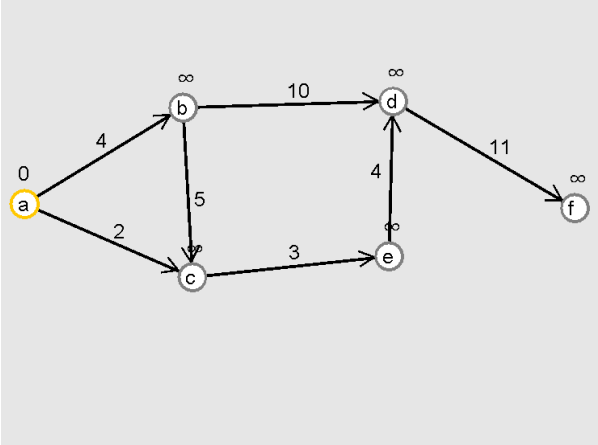

Рисунок 18 - Шаг 1

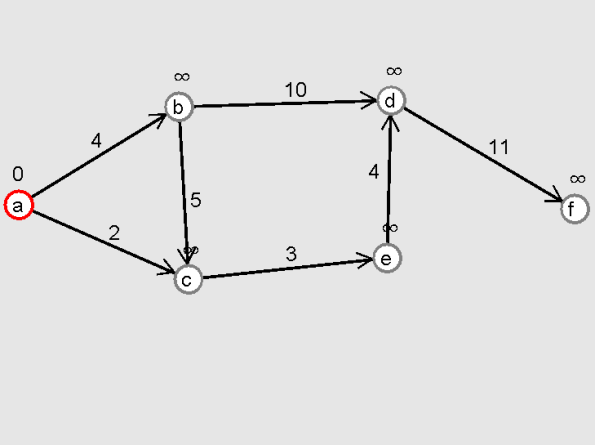

Рисунок 19 - Шаг 2

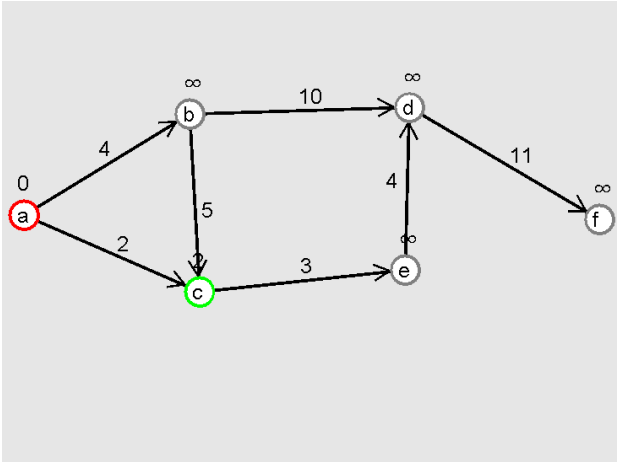

Рисунок 20 - Шаг 3

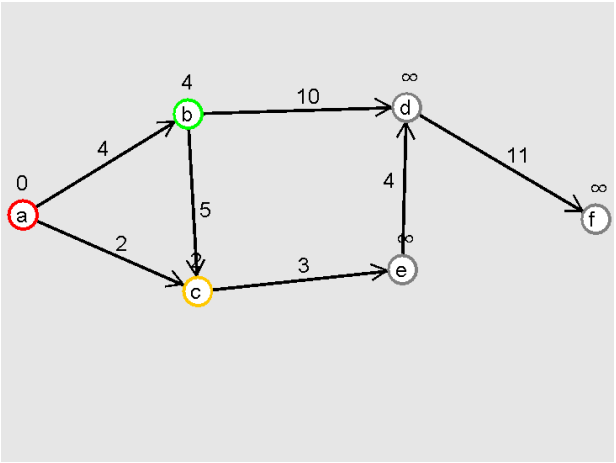

Рисунок 21 - Шаг 4

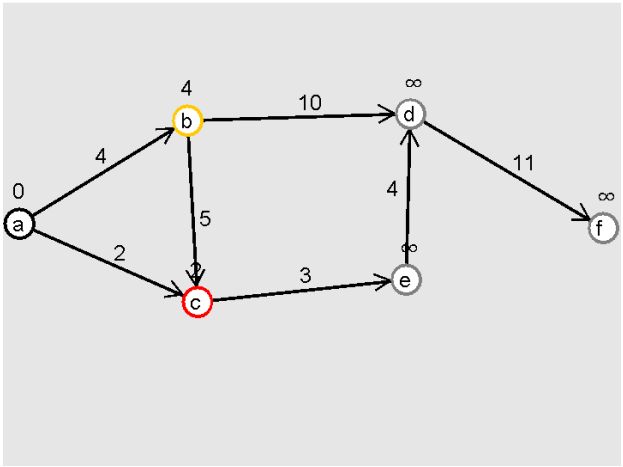

Рисунок 22 - Шаг 5

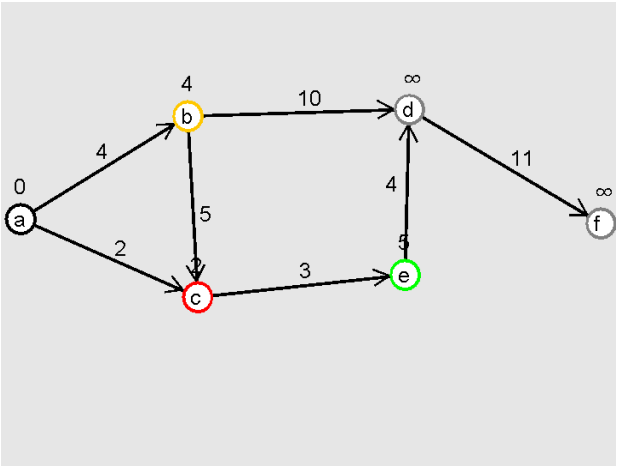

Рисунок 23 - Шаг 6

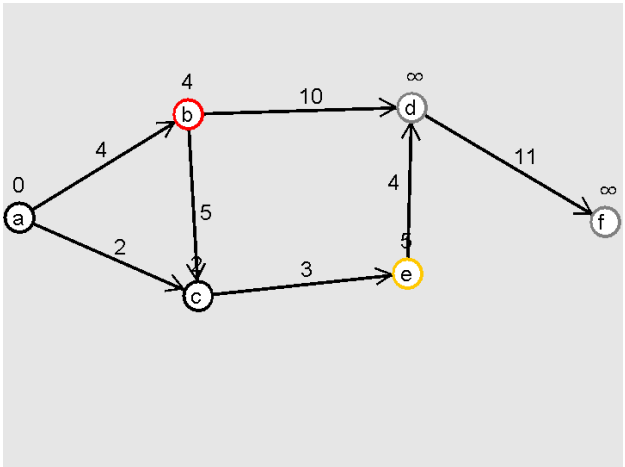

Рисунок 24 - Шаг 7

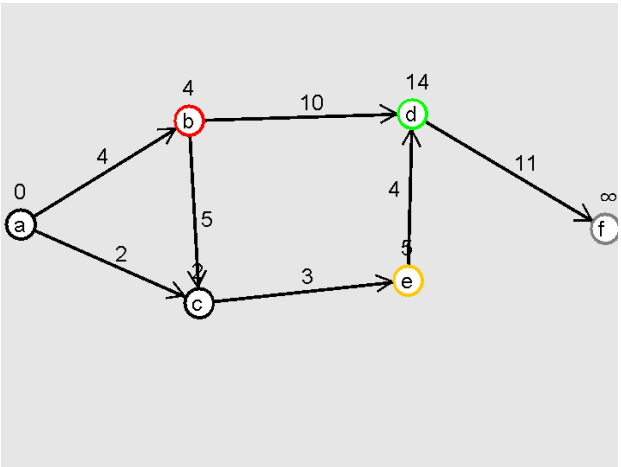

Рисунок 25 - Шаг 8

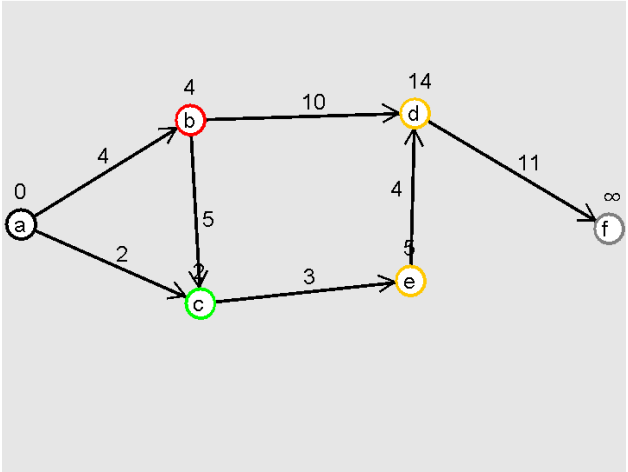

Рисунок 26 - Шаг 9

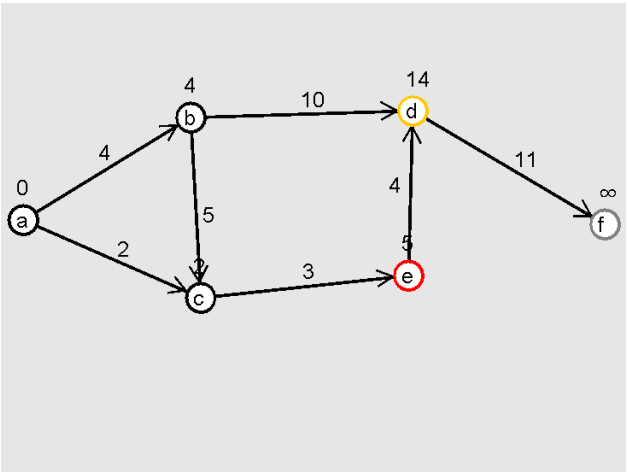

Рисунок 27 - Шаг 10

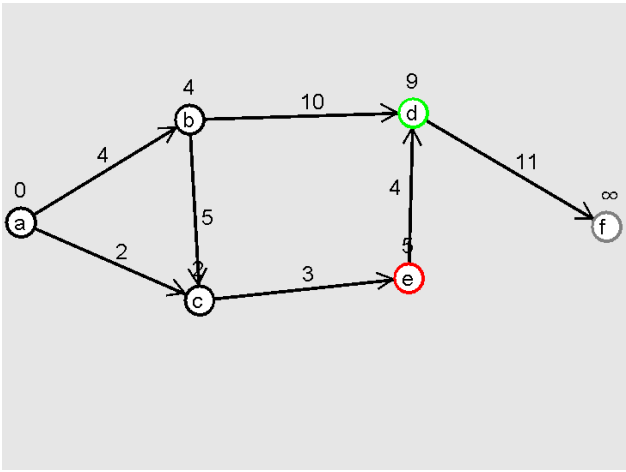

Рисунок 28 - Шаг 11

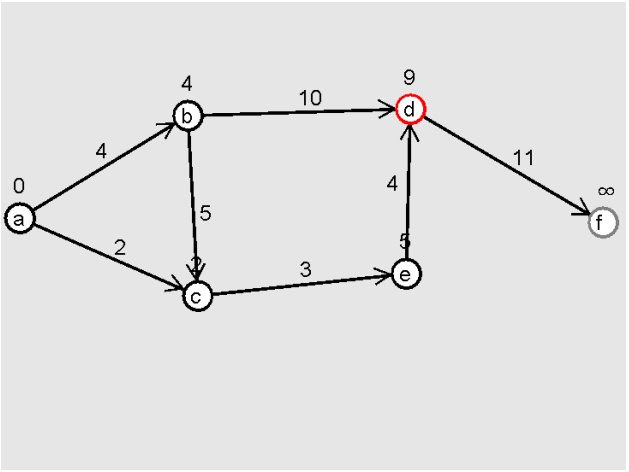

Рисунок 29 - Шаг 12

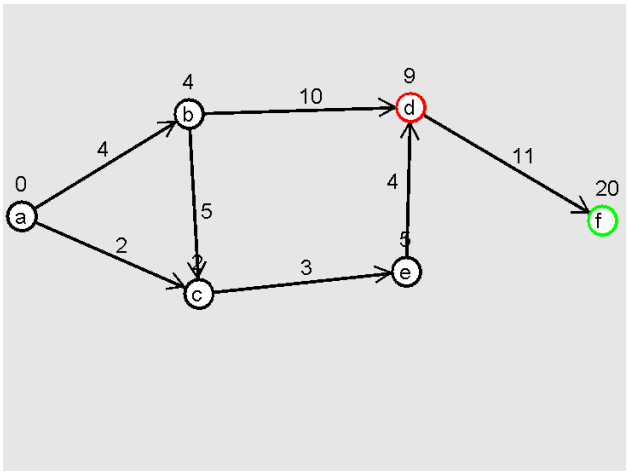

Рисунок 30 - Шаг 13

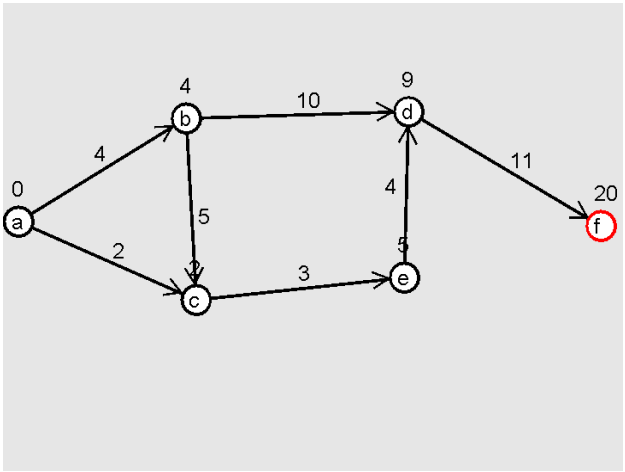

Рисунок 31 - Шаг 14

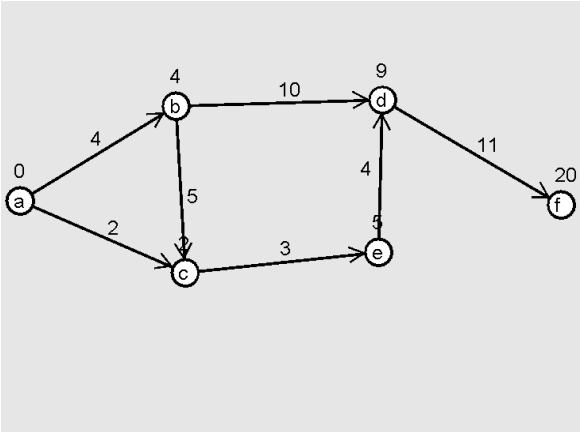

Рисунок 32 - Шаг 15

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 p 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 l 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).

ПРИЛОЖЕНИЕ А
КОД ПРОЕКТА

1) MAIN.JAVA

```
import DataClasses.Graph;
import Visualiazation.*;

public class Main {

    public static void main(String[] args) {
        MainWindow mainWindow = new MainWindow("Dijkstra", new Graph());
        mainWindow.setVisible(true);
    }
}
```

2) GRAPH.JAVA

```
package DataClasses;

import java.awt.*;
import java.util.ArrayList;


public class Graph {

    private ArrayList<Node> nodeList;

    public Graph() {
        nodeList = new ArrayList<>();
    }

    public int addNode(char name) {
        int index = getIndexByName(name);
        if (index == -1) {
            nodeList.add(new Node(name));
            return nodeCount() - 1;
        }
        return index;
    }

    public void removeNode(char name) {
        int index = getIndexByName(name);
```

```
        if (index >= 0) {
            for (int i = 0; i < nodeCount(); i++) {
                getNodeByIndex(i).removeEdge(name);
            }
            nodeList.remove(index);
        }
    }

    private int getIndexByName(char name) {
        for (int i = 0; i < nodeCount(); i++)
            if (nodeList.get(i).getName() == name)
                return i;
        return -1;
    }

    public void addEdge(char start, char end, int weight) {
        if (start == end)
            throw new IllegalArgumentException("Граф не может содержать
петель.");
        if (weight <= 0)
            throw new IllegalArgumentException("Граф может содержать
только рёбра с положительным весом.");
        if (weight > Edge.MAX_WEIGHT)
            throw new IllegalArgumentException("Вес ребра не может
превышать " + Edge.MAX_WEIGHT + ".");
        nodeList.get(addNode(start)).addEdge(end, weight);
        addNode(end);
    }

    public int nodeCount() {
        return nodeList.size();
    }

    public Node getNodeByIndex(int index) {
        if (index < 0 || index >= nodeCount())
            throw new IndexOutOfBoundsException("Index is out of
bounds.");
        return nodeList.get(index);
    }

    public Node getNodeByName(char name) {
        if (getIndexByName(name) == -1)
            throw new IllegalArgumentException("В графе нет вершины с
именем " + name + ".");
```

```
        return getNodeByIndex(getIndexByName(name));
    }

    public Graph clone() {
        Graph cloneGraph = new Graph();
        for (int i = 0; i < nodeCount(); i++)
            cloneGraph.nodeList.add(nodeList.get(i).clone());
        return cloneGraph;
    }


    public ArrayList<AlgorithmStepData> Dijkstra(char startNodeName) {
        if (getIndexByName(startNodeName) == -1)
            throw new IllegalArgumentException("Алгоритм не содержит
вершины с именем '" + startNodeName + "'.");

        ArrayList<AlgorithmStepData> result = new ArrayList<>();
        ArrayList<Integer> queue = new ArrayList<>();
        StringBuilder strBuilder;

        for (int i = 0; i < nodeCount(); i++) {
            getNodeByIndex(i).setDistance(Integer.MAX_VALUE);
            getNodeByIndex(i).setColor(Color.gray);
            getNodeByIndex(i).clearPath();
        }
        getNodeByName(startNodeName).setDistance(0);
        getNodeByName(startNodeName).setColor(Color.orange);
        getNodeByName(startNodeName).addInPath(startNodeName);
        queue.add(getIndexByName(startNodeName));
        strBuilder = new StringBuilder("Шаг 0:\nИнициализация.");
        result.add(new AlgorithmStepData(this.clone(),
strBuilder.toString()));

        int m = 1;
        while (queue.size() > 0) {
            int currentNodeIndex = queue.get(0);
            int indexInQueue = 0;
            for (int i = 1; i < queue.size(); i++) {
                if (getNodeByIndex(queue.get(i)).getDistance() <
getNodeByIndex(currentNodeIndex).getDistance()) {
                    currentNodeIndex = queue.get(i);
                    indexInQueue = i;
                }
            }
```

```
            queue.remove(indexInQueue);

            getNodeByIndex(currentNodeIndex).setColor(Color.red);
            strBuilder = new StringBuilder("Шаг " + m + ":\nРаскрытие
вершины '" + getNodeByIndex(currentNodeIndex).getName() + "'.");
            if (getNodeByIndex(currentNodeIndex).edgeCount() == 0) {
                strBuilder.append("\nУ вершины нет исходящих рёбер.");
            }
            result.add(new AlgorithmStepData(this.clone(),
strBuilder.toString()));

            for (int i = 0; i <
getNodeByIndex(currentNodeIndex).edgeCount(); i++) {
                int currentEndOfEdgeIndex =
getIndexByName(getNodeByIndex(currentNodeIndex).getEdgeByIndex(i).getEndNo
deName());

getNodeByIndex(currentEndOfEdgeIndex).setColor(Color.green);
                if ( getNodeByIndex(currentEndOfEdgeIndex).getDistance()
== Integer.MAX_VALUE ) {

getNodeByIndex(currentEndOfEdgeIndex).setDistance(getNodeByIndex(currentNo
deIndex).getDistance() +
getNodeByIndex(currentNodeIndex).getEdgeByIndex(i).getWeight());

getNodeByIndex(currentEndOfEdgeIndex).setPath(getNodeByIndex(currentNodeIn
dex).getPath());

getNodeByIndex(currentEndOfEdgeIndex).addInPath(getNodeByIndex(currentEndO
fEdgeIndex).getName());
                    strBuilder = new StringBuilder("Вершина '" +
getNodeByIndex(currentEndOfEdgeIndex).getName() + "' ещё не
посещалась.\n");
                    strBuilder.append("Назначен новый путь: " +
getNodeByIndex(currentEndOfEdgeIndex).pathToString() + " (расстояние — " +
getNodeByIndex(currentEndOfEdgeIndex).getDistance() + ").");
                    result.add(new AlgorithmStepData(this.clone(),
strBuilder.toString()));
                    queue.add(currentEndOfEdgeIndex);

getNodeByIndex(currentEndOfEdgeIndex).setColor(Color.orange);
                } else {
                    boolean isInQueue = false;
                    for (int j = 0; j < queue.size(); j++) {
```

```
                        if (currentEndOfEdgeIndex == queue.get(j)) {
                            isInQueue = true;
                            break;
                        }
                    }
                    if (isInQueue) {
                        strBuilder = new StringBuilder("Вершина '" +
getNodeByIndex(currentEndOfEdgeIndex).getName() + "' уже посещалась.\n");
                        strBuilder.append("Текущий путь до вершины: " +
getNodeByIndex(currentEndOfEdgeIndex).pathToString() + " (расстояние — " +
getNodeByIndex(currentEndOfEdgeIndex).getDistance() + ").\n");
                        if
(getNodeByIndex(currentEndOfEdgeIndex).getDistance() >=
getNodeByIndex(currentNodeIndex).getDistance() +
getNodeByIndex(currentNodeIndex).getEdgeByIndex(i).getWeight()) {

getNodeByIndex(currentEndOfEdgeIndex).setDistance(getNodeByIndex(currentNo
deIndex).getDistance() +
getNodeByIndex(currentNodeIndex).getEdgeByIndex(i).getWeight());

getNodeByIndex(currentEndOfEdgeIndex).setPath(getNodeByIndex(currentNodeIn
dex).getPath());

getNodeByIndex(currentEndOfEdgeIndex).addInPath(getNodeByIndex(currentEndO
fEdgeIndex).getName());
                            strBuilder.append("Найден более короткий путь:
" + getNodeByIndex(currentEndOfEdgeIndex).pathToString() + " (расстояние —
" + getNodeByIndex(currentEndOfEdgeIndex).getDistance() + ").");
                        } else {
                            strBuilder.append("Путь из вершины '" +
getNodeByIndex(currentNodeIndex).getName() + "' (расстояние — " +
(getNodeByIndex(currentNodeIndex).getDistance() +
getNodeByIndex(currentNodeIndex).getEdgeByIndex(i).getWeight()) + ") не
является более коротким.");
                        }
                        result.add(new AlgorithmStepData(this.clone(),
strBuilder.toString()));

getNodeByIndex(currentEndOfEdgeIndex).setColor(Color.orange);
                    } else {
                        strBuilder = new StringBuilder("До вершины '" +
getNodeByIndex(currentEndOfEdgeIndex).getName() + "' уже был найден
кратчайший путь.");
```

```
                        result.add(new AlgorithmStepData(this.clone(),
strBuilder.toString()));

getNodeByIndex(currentEndOfEdgeIndex).setColor(Color.black);
                    }
                }
            }

            getNodeByIndex(currentNodeIndex).setColor(Color.black);
            m++;
        }

        strBuilder = new StringBuilder("Результат:\n");
        strBuilder.append("Путь от вершины '" + startNodeName + "' до
вершины\n");
        for (int i = 0; i < nodeCount(); i++) {
            strBuilder.append("'" + getNodeByIndex(i).getName() + "': ");
            if (getNodeByIndex(i).getDistance() == Integer.MAX_VALUE) {
                strBuilder.append("вершина недостежима.\n");
            } else {
                strBuilder.append(getNodeByIndex(i).pathToString() + "
(расстояние — " + getNodeByIndex(i).getDistance() + ").\n");
            }
        }
        result.add(new AlgorithmStepData(this.clone(),
strBuilder.toString()));

        return result;
    }
}
```

3) EDGE.JAVA

```
package DataClasses;

import java.awt.*;

public class Edge {
    private char endNodeName;
    private int weight;
    public static final int MAX_WEIGHT = 1000000;
    private Color color;

    public Edge(char endNodeName, int weight) {
```

```
        this.endNodeName = endNodeName;
        this.weight = weight;
        color = Color.black;
    }

    public char getEndNodeName(){
        return endNodeName;
    }

    public int getWeight(){
        return weight;
    }

    public void setWeight(int weight){
        this.weight = weight;
    }

    public Color getColor() {
        return color;
    }

    public Edge clone() {
        Edge cloneEdge = new Edge(endNodeName, weight);
        cloneEdge.color = color;
        return cloneEdge;
    }

}
```

4) NODE.JAVA

```
package DataClasses;
import java.awt.*;
import java.util.ArrayList;
import java.util.Random;

public class Node {

    private char name;
    private ArrayList<Edge> adjacencyList;
    private int distance;
    private ArrayList<Character> path;
    private Point location;
    private Color color;
```

```
    public static final int BIGRADIUS = 30;
    public static final int BOUND_WIGHT = 500;
    public static final int BOUND_HEIGHT = 400;

    public Node(char name) {
        this.name = name;
        adjacencyList = new ArrayList<>();
        distance = Integer.MAX_VALUE;
        path = new ArrayList<>();
        Random random = new Random();
        location = new Point(random.nextInt(BOUND_WIGHT) + BIGRADIUS,
random.nextInt(BOUND_HEIGHT) + BIGRADIUS);
        color = Color.black;
    }

    public void addEdge(char endNodeName, int weight) {
        for (int i = 0; i < edgeCount(); i++) {
            if (adjacencyList.get(i).getEndNodeName() == endNodeName) {
                adjacencyList.get(i).setWeight(weight);
                return;
            }
        }
        adjacencyList.add(new Edge(endNodeName, weight));
    }

     public void removeEdge(char endNodeName) {
        for (int i = 0; i < edgeCount(); i++) {
            if (adjacencyList.get(i).getEndNodeName() == endNodeName) {
                adjacencyList.remove(i);
                break;
            }
        }
    }

    public int edgeCount() {
        return adjacencyList.size();
    }

    public void setAdjacencyList(ArrayList<Edge> adjacencyList){
        this.adjacencyList = adjacencyList;
    }

    public void setLocation(Point location){
        this.location = location;
```

```
    }

    public Point getLocation(){
        return location;
    }

    public char getName() {
        return name;
    }

    public ArrayList<Edge> getAdjacencyList(){
        return adjacencyList;
    }

    public Edge getEdgeByIndex(int index) {
        if (index < 0 || index >= edgeCount())
            throw new IndexOutOfBoundsException("Index is out of
bounds.");
        return adjacencyList.get(index);
    }

    public Color getColor() {
        return color;
    }

    public void setColor(Color color) {
        this.color = color;
    }

    public int getDistance() {
        return distance;
    }

    public void setDistance(int distance) {
        this.distance = distance;
    }

    public void setPath(ArrayList<Character> path) {
        this.path = (ArrayList<Character>) path.clone();
    }

    public void addInPath(char name) {
        path.add(name);
    }
```

```
    public void clearPath() {
        path.clear();
    }

    public ArrayList<Character> getPath() {
        return path;
    }

    public String pathToString() {
        StringBuilder strBuilder = new StringBuilder("");
        if(path.size() == 0)
            return strBuilder.toString();
        for (int i = 0; i < path.size() - 1; i++)
            strBuilder.append(path.get(i) + "->");
        strBuilder.append(path.get(path.size() - 1));
        return strBuilder.toString();
    }

    public Node clone() {
        Node cloneNode = new Node(name);
        for (int i = 0; i < edgeCount(); i++) {
            cloneNode.adjacencyList.add(adjacencyList.get(i).clone());
        }
        cloneNode.distance = distance;
        cloneNode.path = (ArrayList<Character>) path.clone();
        cloneNode.location = location.getLocation();
        cloneNode.color = color;
        return cloneNode;
    }
}
```

5) ALGORITHMSTEPDATA.JAVA

```
package DataClasses;

public class AlgorithmStepData {
    private Graph graph;
    private String str;

    public AlgorithmStepData(Graph graph, String str) {
        this.graph = graph;
        this.str = str;
    }
```

```
    public Graph getGraph() {
        return graph;
    }

    public String getStr() {
        return str;
    }

}
```

6) DRAWINGPANEL.JAVA

```
package Visualiazation;

import DataClasses.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;

public class DrawingPanel extends JPanel {
    private Graph graph;
    public boolean isAlgorithm = false;
    public static final int BIGRADIUS = 30;
    private static final int LITTLERADIUS = 24;
    private static final int ARROWANGLE = 50;
    private static final int ARROWLENGTH = 30;
    private static final int OFFSETFORNAME = 7;
    private static final int OFFSETFORWEIGHT = 15;


    DrawingPanel(Graph graph, JTextField txtfNode){
        this.graph = graph;
        listenerSettings(txtfNode);
    }

    @Override
    protected void paintComponent ( Graphics g ) {
        super.paintComponent ( g );
        repaint();
        Graphics2D g2 = (Graphics2D) g;
```

```
        drawEdges(g2);
        drawNodes(g2);
    }

    private void drawNodes(Graphics2D g2){
        for(int i = 0; i < graph.nodeCount(); ++i){
            drawOneNode(g2,
String.valueOf(graph.getNodeByIndex(i).getName()),
graph.getNodeByIndex(i).getDistance(),
                    graph.getNodeByIndex(i).getLocation(),
graph.getNodeByIndex(i).getColor());
        }
    }

    private void drawArrows(Graphics2D g2, Point nodeFromLocation, Point
nodeToLocation){
        g2.setColor(Color.black);
        g2.setStroke(new BasicStroke(2.0f));
        double  edgeAngle = Math.atan2(nodeFromLocation.y -
nodeToLocation.y, nodeFromLocation.x - nodeToLocation.x);
        g2.drawLine((int)(nodeToLocation.x + BIGRADIUS / 2 *
Math.cos(edgeAngle)), (int)(nodeToLocation.y + BIGRADIUS / 2 *
Math.sin(edgeAngle)),
                (int)(nodeToLocation.x + ARROWLENGTH * Math.cos(edgeAngle
+ ARROWANGLE)),
                (int)(nodeToLocation.y + ARROWLENGTH * Math.sin(edgeAngle
+ ARROWANGLE)));

        g2.drawLine((int)(nodeToLocation.x + BIGRADIUS / 2 *
Math.cos(edgeAngle)), (int)(nodeToLocation.y + BIGRADIUS / 2 *
Math.sin(edgeAngle)),
                (int)(nodeToLocation.x + ARROWLENGTH * Math.cos(edgeAngle
- ARROWANGLE)),
                (int)(nodeToLocation.y + ARROWLENGTH * Math.sin(edgeAngle
- ARROWANGLE)));
    }

    private void drawEdges(Graphics2D g2){
        for(int i = 0; i < graph.nodeCount(); ++i){
            Point nodeFromLocation = new
Point(graph.getNodeByIndex(i).getLocation());
            ArrayList <Edge> currentAdjacencyList =
graph.getNodeByIndex(i).getAdjacencyList();
            for(int j = 0; j < currentAdjacencyList.size(); ++j){
```

```
                Point nodeToLocation = new
Point(graph.getNodeByName(currentAdjacencyList.get(j).getEndNodeName()).ge
tLocation());
                drawLine(g2, nodeFromLocation, nodeToLocation,
currentAdjacencyList.get(j).getColor());
                drawArrows(g2, nodeFromLocation, nodeToLocation);
                printEdgeWeightInPoint(g2,
String.valueOf(currentAdjacencyList.get(j).getWeight()), nodeFromLocation,
nodeToLocation);
            }
        }
    }

    private void drawMainLine(Graphics2D g2, Point from, Point to, Color
color){
        g2.setColor(color);
        g2.setStroke(new BasicStroke(3.0f));
        g2.drawLine(from.x, from.y, to.x, to.y);
    }

    private void drawLine(Graphics2D g2, Point from, Point to, Color
color){
        drawMainLine(g2, from, to, color);
    }

    private void printStringInPoint(Graphics2D g2, String string, Point
point){
        g2.setColor(Color.black);
        g2.setFont(new Font("TimesRoman", Font.PLAIN, 20));
        g2.drawString(string, point.x - OFFSETFORNAME, point.y +
OFFSETFORNAME);
    }

    private void printEdgeWeightInPoint(Graphics2D g2, String weight,
Point nodeFromLocation, Point nodeToLocation){
        double edgeAngle = Math.atan2(nodeFromLocation.y -
nodeToLocation.y, nodeFromLocation.x - nodeToLocation.x);
        int length  = (int)Math.sqrt(Math.pow(nodeFromLocation.x -
nodeToLocation.x, 2) + Math.pow(nodeFromLocation.y - nodeToLocation.y, 2))
/ 2;
        printStringInPoint(g2, weight, new Point((int)(nodeToLocation.x +
length * Math.cos(edgeAngle) + OFFSETFORWEIGHT * Math.cos(edgeAngle +
90)),
```

```
                                                    (int)(nodeToLocation.y +
length * Math.sin(edgeAngle) + OFFSETFORWEIGHT * Math.sin(edgeAngle +
90))));

    }

    private void drawOneNode(Graphics2D g2, String string, int distance,
Point point, Color color){
        if(isAlgorithm) {
            if(distance == Integer.MAX_VALUE)
                printStringInPoint(g2, String.valueOf('\u221E'), new
Point(point.x, point.y - BIGRADIUS));
            else
                printStringInPoint(g2, String.valueOf(distance), new
Point(point.x, point.y - BIGRADIUS));
        }

        g2.setColor(color);
        g2.fillOval(point.x - BIGRADIUS / 2, point.y - BIGRADIUS /2,
BIGRADIUS, BIGRADIUS);
        g2.setColor(Color.WHITE);
        g2.fillOval(point.x - LITTLERADIUS / 2, point.y - LITTLERADIUS /2,
LITTLERADIUS, LITTLERADIUS);
        printStringInPoint(g2, string, point);
    }

    private void listenerSettings(JTextField txtfNode) {
        addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                super.mouseClicked(e);

                if (e.getButton() == MouseEvent.BUTTON1) {
                    if (!txtfNode.getText().isEmpty()) {

graph.getNodeByIndex(graph.addNode(txtfNode.getText().charAt(0))).setLocat
ion(e.getPoint());
                        txtfNode.setText("");
                        return;
                    }
                    else
                        for (int i = 0; i < graph.nodeCount(); ++i) {
                            if (graph.getNodeByIndex(i).getLocation().x <=
e.getPoint().x + BIGRADIUS / 2 &&
```

```
graph.getNodeByIndex(i).getLocation().x >= e.getPoint().x - BIGRADIUS / 2
&&

graph.getNodeByIndex(i).getLocation().y <= e.getPoint().y + BIGRADIUS / 2
&&

graph.getNodeByIndex(i).getLocation().y >= e.getPoint().y - BIGRADIUS / 2)
{

graph.removeNode(graph.getNodeByIndex(i).getName());
                                return;
                            }
                        }
                    JOptionPane.showMessageDialog(null, "Node`s name
empty");
                }

                if (e.getButton() == MouseEvent.BUTTON3) {
                    for (int i = 0; i < graph.nodeCount(); ++i) {
                        if (graph.getNodeByIndex(i).getLocation().x <=
e.getPoint().x + BIGRADIUS / 2 &&
                                graph.getNodeByIndex(i).getLocation().x >=
e.getPoint().x - BIGRADIUS / 2 &&
                                graph.getNodeByIndex(i).getLocation().y <=
e.getPoint().y + BIGRADIUS / 2 &&
                                graph.getNodeByIndex(i).getLocation().y >=
e.getPoint().y - BIGRADIUS / 2)
                            if (graph.getNodeByIndex(i).getColor() ==
Color.black)

graph.getNodeByIndex(i).setColor(Color.BLUE);
                            else if (graph.getNodeByIndex(i).getColor() ==
Color.BLUE)

graph.getNodeByIndex(i).setColor(Color.YELLOW);
                            else

graph.getNodeByIndex(i).setColor(Color.black);
                        }
                }
            }
        });
    }
```

```
    public void updateGraph(Graph newGraph){
        this.graph = newGraph;
    }

    public void setTrueIsAlgorithmValue(){
        isAlgorithm = true;
    }
    public void setFalseIsAlgorithmValue() { isAlgorithm = false; }
}
```

7) MAINWINDOW.JAVA

```
package Visualiazation;

import DataClasses.AlgorithmStepData;
import DataClasses.Edge;
import DataClasses.Graph;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Random;
import java.util.Timer;
import java.util.TimerTask;

import org.json.simple.*;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

class ScheduledTask extends TimerTask {
    private Timer time;
    private int stepsNum;
    private int stepsCount;
    private MainWindow windowEx;
    ScheduledTask(JButton nextStep, Timer time, int stepsNum, int
stepsCount, MainWindow windowEx){
        this.time = time;
        this.stepsNum = stepsNum;
        this.stepsCount = stepsCount;
        this.windowEx = windowEx;
```

```
    }

    @Override
    public void run() {
        if(stepsCount >= stepsNum ) {
            time.cancel();
            time.purge();
            windowEx.setUpCloseMainThreadAlgorithm();
            JOptionPane.showMessageDialog(null, "algorithm end work!");
            return;
        }
        stepsCount++;
        windowEx.nextStep();
    }
}

public class MainWindow extends JFrame {
    private static final int MAINWINDOW_WIDTH = 900;
    private static final int MAINWINDOW_HEIGHT = 700;
    private static final int BIGRADIUS = 30;
    private static final int BOUND_WIDTH = 500;
    private static final int BOUND_HEIGHT = 400;

    private Box boxVInputPanel;
    private JLabel lblNode;
    private JLabel lblEdgeFrom;
    private JLabel lblEdgeTo;
    private JLabel lblWeight;
    private JTextField txtfNode;
    private JTextField txtfEdgeFrom;
    private JTextField txtfEdgeTo;
    private JTextField txtfEdgeWeight;
    private JButton btnManual;
    private JButton btnFromFile;
    private JButton btnSaveFile;
    private JButton btnEdgeAdd;
    private JButton btnNodeAdd;
    private JButton btnNodeRemove;
    private JButton btnRepaint;
    private JButton btnGoToAlgorithm;
    private JButton btnResetInput;

    private Box boxVOutputPanel;
    private JLabel lblStartNode;
```

```
private JLabel lblAimNode;
private JTextField txtfStartNode;
private JTextField txtfAimNode;
private JButton btnStartAlgorithm;
private JButton btnCalculateLength;
private JButton btnNextStep;
private JButton btnPreviousStep;
private JButton btnResetOutput;

private JTextArea txtaLog;

private DrawingPanel drawingPanel;

private Graph graph;
private ArrayList<AlgorithmStepData> graphStates;
private int algorithmStepNum;
private java.util.Timer time = new Timer();
private boolean isAlgorithmAlreadyWorked;
private boolean closeMainThreadAlgorithm;

private MainWindow thisWindow = this;

public MainWindow(String title, Graph graph){
    super(title);
    this.graph = graph;
    initVariables();
    windowSettings();
    layoutSettins();
    buttonsSettings();
}

private void initVariables(){
    boxVInputPanel = Box.createVerticalBox();
    lblNode      = new JLabel("Name");
    lblEdgeFrom  = new JLabel("From");
    lblEdgeTo    = new JLabel("To");
    lblWeight    = new JLabel("Weight");

    txtfNode       = new JTextField(2);
    txtfNode.addKeyListener(new KeyAdapter() {
        public void keyTyped(KeyEvent e) {
            if(txtfNode.getText().length() >= 1)
                e.consume();  // ignore event
            }
```

```
        });

        txtfEdgeFrom   = new JTextField(2);
        txtfEdgeFrom.addKeyListener(new KeyAdapter() {
            public void keyTyped(KeyEvent e) {
                if(txtfEdgeFrom.getText().length() >= 1)
                    e.consume();  // ignore event
            }
        });

        txtfEdgeTo     = new JTextField(2);
        txtfEdgeTo.addKeyListener(new KeyAdapter() {
            public void keyTyped(KeyEvent e) {
                if(txtfEdgeTo.getText().length() >= 1)
                    e.consume();  // ignore event
            }
        });

        txtfEdgeWeight = new JTextField(5);
        txtfEdgeWeight.addKeyListener(new KeyAdapter() {
            public void keyTyped(KeyEvent e) {
                if(txtfEdgeWeight.getText().length() >= 5) {
                    e.consume();  // ignore event
                    JOptionPane.showMessageDialog(null, "very big
weight");
                }
            }
        });

        btnManual = new JButton("Manual");
        btnFromFile  = new JButton("From File");
        btnSaveFile = new JButton("Save this Graph");
        btnNodeAdd   = new JButton("Add");
        btnNodeRemove = new JButton("Remove");
        btnEdgeAdd   = new JButton("Add");
        btnRepaint   = new JButton("Repaint Graph");
        btnGoToAlgorithm = new JButton("Go to algorithm");
        btnResetInput = new JButton("Reset");

        txtaLog = new JTextArea(10, 0);
        txtaLog.setText("Algorithm steps\n");
        txtaLog.setLineWrap(true);
        txtaLog.setWrapStyleWord(true);
```

```
        lblStartNode = new JLabel("Input start Node");
        lblAimNode = new JLabel("Aim Node");

        txtfStartNode  = new JTextField(2);
        txtfStartNode.addKeyListener(new KeyAdapter() {
            public void keyTyped(KeyEvent e) {
                if(txtfStartNode.getText().length() >= 1)
                    e.consume();  // ignore event
            }
        });

        txtfAimNode = new JTextField(2);
        txtfAimNode.addKeyListener(new KeyAdapter() {
            public void keyTyped(KeyEvent e) {
                if(txtfAimNode.getText().length() >= 1)
                    e.consume();  // ignore event
            }
        });

        boxVOutputPanel = Box.createVerticalBox();
        btnStartAlgorithm = new JButton("Start algorithm");
        btnCalculateLength = new JButton("Get length");
        btnNextStep = new JButton("Next step");
        btnPreviousStep = new JButton("Previous step");
        btnResetOutput = new JButton("Reset");

        algorithmStepNum = 0;
        time = new Timer();
        isAlgorithmAlreadyWorked = false;
        closeMainThreadAlgorithm = false;
    }

    private void windowSettings(){
        setSize(MAINWINDOW_WIDTH, MAINWINDOW_HEIGHT);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLocation(new Point(450, 200));
        setResizable(false);
    }

    private void layoutSettins(){
        layoutInputSettings();
        layoutOutputSettings();
        layoutDrawingPanelSettings();
        getContentPane().add(boxVInputPanel, BorderLayout.WEST);
```

```
        getContentPane().add(drawingPanel);
        getContentPane().add(new JScrollPane(txtaLog),
BorderLayout.SOUTH);
    }

    private void buttonsSettings(){

        btnManual.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                File file = new File("Docs/manual.txt");
                StringBuilder allStrings = new StringBuilder();
                try (BufferedReader br = new BufferedReader(new
FileReader(file))) {
                    String line;
                    while ((line = br.readLine()) != null) {
                        allStrings.append(line + '\n');
                    }
                } catch (FileNotFoundException ex) {
                    ex.printStackTrace();
                } catch (IOException ex) {
                    ex.printStackTrace();
                }
                JOptionPane.showMessageDialog(null,
allStrings.toString());
            }
        });

        btnFromFile.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JFileChooser fileopen = new JFileChooser();
                File file = null;
                if (fileopen.showDialog(null, "Открыть файл") ==
JFileChooser.APPROVE_OPTION) {
                    graph = new Graph();
                    drawingPanel.updateGraph(graph);
                    file = fileopen.getSelectedFile();
                    parseGraph(file);
                }
            }
        });

        btnSaveFile.addActionListener(new ActionListener() {
            @Override
```

```
            public void actionPerformed(ActionEvent e) {
                JFileChooser fileopen = new JFileChooser();
                File file = null;
                if (fileopen.showDialog(null, "Открыть файл") ==
JFileChooser.APPROVE_OPTION) {
                    file = fileopen.getSelectedFile();
                    saveIntoFile(file);
                }
            }
        });

        btnNodeAdd.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if(!txtfNode.getText().isEmpty()) {
                    for(int i = 0; i < graph.nodeCount(); ++i){
                        if(graph.getNodeByIndex(i).getName() ==
txtfNode.getText().charAt(0)) {
                            JOptionPane.showMessageDialog(null, "This node
is already in graph");
                        }
                    }
                    graph.addNode(txtfNode.getText().charAt(0));
                    txtfNode.setText("");
                }
                else
                    JOptionPane.showMessageDialog(null, "Node`s name
empty");
            }
        });

        btnNodeRemove.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if(!txtfNode.getText().isEmpty()) {
                    boolean isDelelete = false;
                    for (int i = 0; i < graph.nodeCount(); ++i) {
                        if (graph.getNodeByIndex(i).getName() ==
txtfNode.getText().charAt(0)) {

graph.removeNode(txtfNode.getText().charAt(0));
                            isDelelete = true;
                        }
                    }
```

```
                    if (!isDelelete)
                        JOptionPane.showMessageDialog(null, "Graph don`t
contain this node");
                }
                else
                    JOptionPane.showMessageDialog(null, "Node`s name
empty");
                txtfNode.setText("");
            }
        });

        btnEdgeAdd.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if(txtfEdgeTo.getText().charAt(0) ==
txtfEdgeFrom.getText().charAt(0)) {
                    JOptionPane.showMessageDialog(null, "Graph don`t have
loop!");
                    txtfEdgeFrom.setText("");
                    txtfEdgeTo.setText("");
                    txtfEdgeWeight.setText("");
                    return;
                }

                boolean flag = false;
                if(!txtfEdgeTo.getText().isEmpty() &&
!txtfEdgeFrom.getText().isEmpty() && !txtfEdgeWeight.getText().isEmpty())
{
                    int currentWeight = parseInt();
                    if(currentWeight == 0) {
                        txtfEdgeFrom.setText("");
                        txtfEdgeTo.setText("");
                        txtfEdgeWeight.setText("");
                        return;
                    }

                    boolean isContain = false;
                    for(int i = 0; i < graph.nodeCount(); ++i)
                        if(graph.getNodeByIndex(i).getName() ==
txtfEdgeFrom.getText().charAt(0))
                            isContain = true;

                    if(isContain) {
```

```
                        for (int i = 0; i <
graph.getNodeByName(txtfEdgeFrom.getText().charAt(0)).edgeCount(); ++i) {
                            if
(graph.getNodeByName(txtfEdgeFrom.getText().charAt(0)).getEdgeByIndex(i).g
etEndNodeName() == txtfEdgeTo.getText().charAt(0)) {
                                JOptionPane.showMessageDialog(null, "Graph
is already contain edge \n" +

"weight will be update");
                            }
                        }
                    }

                    graph.addEdge(txtfEdgeFrom.getText().charAt(0),
txtfEdgeTo.getText().charAt(0), currentWeight);
                    repaint();
                    flag = true;
                }

                boolean isSetBlack = false;
                for(int i = 0; i < graph.nodeCount(); ++i){
                    if(graph.getNodeByIndex(i).getColor() == Color.BLUE) {
                        for (int j = 0; j < graph.nodeCount(); ++j) {
                            if (graph.getNodeByIndex(j).getColor() ==
Color.YELLOW) {
                                isSetBlack = true;
                                if(!txtfEdgeWeight.getText().isEmpty()) {
                                    int currentWeight = parseInt();
                                    if(currentWeight == 0)
                                        return;

graph.addEdge(graph.getNodeByIndex(i).getName(),
graph.getNodeByIndex(j).getName(), currentWeight);
                                    flag = true;
                                }
                            }
                        }
                    }
                }

                if(isSetBlack)
                    for(int i = 0; i < graph.nodeCount(); ++i)
                        graph.getNodeByIndex(i).setColor(Color.BLACK);
```

```
                txtfEdgeFrom.setText("");
                txtfEdgeTo.setText("");
                txtfEdgeWeight.setText("");

                if(!flag)
                    JOptionPane.showMessageDialog(null, "Edge fields
Empty");
            }
        });

        btnRepaint.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Random random = new Random();
                for(int i = 0; i < graph.nodeCount(); ++i){
                    graph.getNodeByIndex(i).setLocation(new
Point(random.nextInt(BOUND_WIDTH) + BIGRADIUS,
random.nextInt(BOUND_HEIGHT) + BIGRADIUS));
                }
            }
        });

        btnGoToAlgorithm.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                getContentPane().remove(boxVInputPanel);
                getContentPane().add(boxVOutputPanel, BorderLayout.WEST);
                validate();
                repaint();
            }
        });

        btnResetInput.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                graph = new Graph();
                drawingPanel.updateGraph(graph);
            }
        });

        btnStartAlgorithm.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if(txtfStartNode.getText().isEmpty()) {
```

```
                    JOptionPane.showMessageDialog(null, "Input start node
name");
                    return;
                }
                boolean isRightNode = false;
                for(int i = 0; i < graph.nodeCount(); ++i){
                    if(graph.getNodeByIndex(i).getName() ==
txtfStartNode.getText().charAt(0))
                        isRightNode = true;
                }

                if(!isRightNode){
                    JOptionPane.showMessageDialog(null, "There are no node
with this name");
                    return;
                }

                if(!isAlgorithmAlreadyWorked) {
                    isAlgorithmAlreadyWorked = true;
                    graphStates =
graph.Dijkstra(txtfStartNode.getText().charAt(0));
                    graph = graphStates.get(algorithmStepNum).getGraph();
                    drawingPanel.updateGraph(graph);
                    drawingPanel.setTrueIsAlgorithmValue();
                    txtaLog.setText("");
                    txtaLog.append(graphStates.get(0).getStr());

                    setUpCloseMainThreadAlgorithm();
                    ScheduledTask task = new ScheduledTask(btnNextStep,
time, graphStates.size() - 1, algorithmStepNum, thisWindow);
                    time.schedule(task, 0, 1000);
                }
                else
                    JOptionPane.showMessageDialog(null, "Algorithm already
worked");
            }
        });

        btnCalculateLength.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if(txtfAimNode.getText().isEmpty()) {
                    JOptionPane.showMessageDialog(null, "Input aim node
name!");
```

```
                    return;
                }

                for(int i = 0; i < graph.nodeCount(); ++i){
                    if(graph.getNodeByIndex(i).getName() ==
txtfAimNode.getText().charAt(0)){
                        if(graph.getNodeByIndex(i).pathToString().length()
!= 0)
                            JOptionPane.showMessageDialog(null, "Lenght =
" + String.valueOf(graph.getNodeByIndex(i).getDistance()) +
                                    '\n' + " PATH FROM " +
txtfStartNode.getText().charAt(0) +
                                    " TO " +
txtfAimNode.getText().charAt(0) + " : " +
graph.getNodeByIndex(i).pathToString());
                        else
                            JOptionPane.showMessageDialog(null, "there is
no way!");
                        return;
                    }
                }

                JOptionPane.showMessageDialog(null, "There are no node
with this name!");
            }
        });

        btnNextStep.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if(closeMainThreadAlgorithm){
                    JOptionPane.showMessageDialog(null, "Main thread
closed");
                    return;
                }
                if (drawingPanel.isAlgorithm){
                    if(algorithmStepNum != graphStates.size() - 1) {
                        graph =
graphStates.get(++algorithmStepNum).getGraph();
                        drawingPanel.updateGraph(graph);
                        txtaLog.setText("");
                        for(int i = 0; i <= algorithmStepNum; ++i){
                            txtaLog.append(graphStates.get(i).getStr() +
'\n' + '\n');
```

```
                    }
                }
                else
                    JOptionPane.showMessageDialog(null, "Algorithm`s
work end!");
            }
            else
                JOptionPane.showMessageDialog(null, "Algorithm don`t
start!");
        }
    });

    btnPreviousStep.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e) {
            if(closeMainThreadAlgorithm){
                JOptionPane.showMessageDialog(null, "Main thread
closed");
                return;
            }

            if(algorithmStepNum != 0){
                graph = graphStates.get(--
algorithmStepNum).getGraph();
                drawingPanel.updateGraph(graph);
                txtaLog.setText("");
                for(int i = 0; i <= algorithmStepNum; ++i){
                    txtaLog.append(graphStates.get(i).getStr() + '\n'
+ '\n');
                }
            }
            else
                JOptionPane.showMessageDialog(null, "It`s already
first step!");
        }
    });

    btnResetOutput.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            getContentPane().remove(boxVOutputPanel);
            JPanel panel = new JPanel();
            panel.setBackground(Color.BLUE);
```

```
            panel.setPreferredSize(new Dimension(130, 0));
            getContentPane().add(boxVInputPanel, BorderLayout.WEST);

            graph = new Graph();
            drawingPanel.setFalseIsAlgorithmValue();
            drawingPanel.updateGraph(graph);
            txtaLog.setText("Algorithm steps");

            time = new Timer();
            isAlgorithmAlreadyWorked = false;
            algorithmStepNum = 0;

            validate();
            repaint();
        }
    });
}

private void layoutInputSettings(){
    boxVInputPanel.setPreferredSize(new Dimension(150, 0));

    boxVInputPanel.add(Box.createVerticalStrut(10));
    Box boxHSetUpManual = Box.createHorizontalBox();
    boxHSetUpManual.add(btnManual);
    boxVInputPanel.add(boxHSetUpManual);

    boxVInputPanel.add(Box.createVerticalStrut(10));
    Box boxHSetUpFromFileBtn = Box.createHorizontalBox();
    boxHSetUpFromFileBtn.add(btnFromFile);
    boxVInputPanel.add(boxHSetUpFromFileBtn);

    boxVInputPanel.add(Box.createVerticalStrut(10));
    Box boxHSetUpSaveFileBtn = Box.createHorizontalBox();
    boxHSetUpSaveFileBtn.add(btnSaveFile);
    boxVInputPanel.add(boxHSetUpSaveFileBtn);

    boxVInputPanel.add(Box.createVerticalStrut(50));
    Box boxHSetUpNodeLbl = Box.createHorizontalBox();
    boxHSetUpNodeLbl.add(Box.createHorizontalStrut(5));
    boxHSetUpNodeLbl.add(lblNode);
    boxHSetUpNodeLbl.add(Box.createHorizontalStrut(5));
    boxHSetUpNodeLbl.add(txtfNode);
    boxHSetUpNodeLbl.add(Box.createHorizontalStrut(5));
    boxHSetUpNodeLbl.add(btnNodeAdd);
```

```
boxHSetUpNodeLbl.add(Box.createHorizontalStrut(10));
boxVInputPanel.add(boxHSetUpNodeLbl);

boxVInputPanel.add(Box.createVerticalStrut(5));
Box boxHSetUpNodeRemove = Box.createHorizontalBox();
boxHSetUpNodeRemove.add(Box.createHorizontalStrut(46));
boxHSetUpNodeRemove.add(btnNodeRemove);
boxVInputPanel.add(boxHSetUpNodeRemove);

boxVInputPanel.add(Box.createVerticalStrut(50));
Box boxHSetUpEdgeLbl = Box.createHorizontalBox();
boxHSetUpEdgeLbl.add(Box.createHorizontalStrut(5));
boxHSetUpEdgeLbl.add(lblEdgeFrom);
boxHSetUpEdgeLbl.add(Box.createHorizontalStrut(5));
boxHSetUpEdgeLbl.add(txtfEdgeFrom);
boxHSetUpEdgeLbl.add(Box.createHorizontalStrut(10));
boxHSetUpEdgeLbl.add(lblEdgeTo);
boxHSetUpEdgeLbl.add(Box.createHorizontalStrut(5));
boxHSetUpEdgeLbl.add(txtfEdgeTo);
boxHSetUpEdgeLbl.add(Box.createHorizontalStrut(15));
boxVInputPanel.add(boxHSetUpEdgeLbl);

boxVInputPanel.add(Box.createVerticalStrut(10));
Box boxHSetUpEdgeWeight = Box.createHorizontalBox();
boxHSetUpEdgeWeight.add(Box.createHorizontalStrut(5));
boxHSetUpEdgeWeight.add(lblWeight);
boxHSetUpEdgeWeight.add(Box.createHorizontalStrut(5));
boxHSetUpEdgeWeight.add(txtfEdgeWeight);
boxHSetUpEdgeWeight.add(Box.createHorizontalStrut(3));
boxHSetUpEdgeWeight.add(btnEdgeAdd);
boxHSetUpEdgeWeight.add(Box.createHorizontalStrut(3));
boxVInputPanel.add(boxHSetUpEdgeWeight);

boxVInputPanel.add(Box.createVerticalStrut(50));
Box boxHSetUpRepaintBtn = Box.createHorizontalBox();
boxHSetUpRepaintBtn.add(btnRepaint);
boxVInputPanel.add(boxHSetUpRepaintBtn);

boxVInputPanel.add(Box.createVerticalStrut(10));
Box boxHSetUpFinishBtn = Box.createHorizontalBox();
boxHSetUpFinishBtn.add(btnGoToAlgorithm);
boxVInputPanel.add(boxHSetUpFinishBtn);

boxVInputPanel.add(Box.createVerticalStrut(64));
```

```
        Box boxHSetUpResetBtn = Box.createHorizontalBox();
        boxHSetUpResetBtn.add(btnResetInput);
        boxVInputPanel.add(boxHSetUpResetBtn);


boxVInputPanel.add(Box.createVerticalStrut((int)Double.POSITIVE_INFINITY))
;
    }

    private void layoutDrawingPanelSettings(){

        drawingPanel = new DrawingPanel(graph, txtfNode);
        drawingPanel.setPreferredSize(new Dimension(1000, 1000));
        drawingPanel.setBackground(new Color(230, 230, 230));
    }

    private void layoutOutputSettings(){
        boxVOutputPanel.setPreferredSize(new Dimension(150, 0));

        boxVOutputPanel.add(Box.createVerticalStrut(10));
        Box boxHSetUpStartNodeLbl = Box.createHorizontalBox();
        boxHSetUpStartNodeLbl.add(Box.createHorizontalStrut(5));
        boxHSetUpStartNodeLbl.add(lblStartNode);
        boxHSetUpStartNodeLbl.add(Box.createHorizontalStrut(5));
        boxHSetUpStartNodeLbl.add(txtfStartNode);
        boxHSetUpStartNodeLbl.add(Box.createHorizontalStrut(5));
        boxVOutputPanel.add(boxHSetUpStartNodeLbl);

        boxVOutputPanel.add(Box.createVerticalStrut(10));
        Box boxHSetStartAlgorithmBtn = Box.createHorizontalBox();
        boxHSetStartAlgorithmBtn.add(btnStartAlgorithm);
        boxVOutputPanel.add(boxHSetStartAlgorithmBtn);

        boxVOutputPanel.add(Box.createVerticalStrut(40));
        Box boxHSetUpAimNodeLbl = Box.createHorizontalBox();
        boxHSetUpAimNodeLbl.add(Box.createHorizontalStrut(5));
        boxHSetUpAimNodeLbl.add(lblAimNode);
        boxHSetUpAimNodeLbl.add(Box.createHorizontalStrut(5));
        boxHSetUpAimNodeLbl.add(txtfAimNode);
        boxHSetUpAimNodeLbl.add(Box.createHorizontalStrut(45));
        boxVOutputPanel.add(boxHSetUpAimNodeLbl);

        boxVOutputPanel.add(Box.createVerticalStrut(10));
        Box boxHSetUpLengthBtn = Box.createHorizontalBox();
```

```
        boxHSetUpLengthBtn.add(btnCalculateLength);
        boxVOutputPanel.add(boxHSetUpLengthBtn);

        boxVOutputPanel.add(Box.createVerticalStrut(50));
        Box boxHSetUpNextBtn = Box.createHorizontalBox();
        boxHSetUpNextBtn.add(btnNextStep);
        boxVOutputPanel.add(boxHSetUpNextBtn);

        boxVOutputPanel.add(Box.createVerticalStrut(10));
        Box boxHSetUpPreviousBtn = Box.createHorizontalBox();
        boxHSetUpPreviousBtn.add(btnPreviousStep);
        boxVOutputPanel.add(boxHSetUpPreviousBtn);

        boxVOutputPanel.add(Box.createVerticalStrut(220));
        Box boxHSetUpResetBtn = Box.createHorizontalBox();
        boxHSetUpResetBtn.add(btnResetOutput);
        boxVOutputPanel.add(boxHSetUpResetBtn);


boxVOutputPanel.add(Box.createVerticalStrut((int)Double.POSITIVE_INFINITY)
);
    }

    private void parseGraph(File file) {
        JSONParser jsonParser = new JSONParser();

        try (FileReader reader = new FileReader(file))
        {
            JSONArray nodesList = (JSONArray) jsonParser.parse(reader);
            nodesList.forEach( emp -> parseNodeObject( (JSONObject) emp )
);

        } catch (FileNotFoundException e) {
            JOptionPane.showMessageDialog(null, "FILE NOT FOUND!");
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "ERROR");
        } catch (ParseException e) {
            JOptionPane.showMessageDialog(null, "PARSE ERROR. MUST BE
.json file format");
        }
    }

    private void parseNodeObject(JSONObject node)
    {
```

```
        String name = (String) node.get("name");
        graph.addNode(name.charAt(0));

        JSONArray location = (JSONArray) node.get("location");
        Long xl = (Long)(location.get(0));
        int x = xl.intValue();
        Long yl = (Long)(location.get(1));
        int y = yl.intValue();
        graph.getNodeByIndex(graph.nodeCount()-1).setLocation(new Point(x,
y));

        JSONArray adjacencyList = (JSONArray) node.get("adjacencyList");
        ArrayList<Edge> adjacencyListForSetUp = new ArrayList<Edge>();
        Iterator adjListItr = adjacencyList.iterator();
        while ((adjListItr.hasNext())){
            JSONArray currentEdge = (JSONArray) adjListItr.next();
            Long  wl = (Long) currentEdge.get(1);
            int weight = wl.intValue();
            adjacencyListForSetUp.add(new
Edge(String.valueOf(currentEdge.get(0)).charAt(0), weight));
        }
        graph.getNodeByIndex(graph.nodeCount()-
1).setAdjacencyList(adjacencyListForSetUp);
    }

    private void saveIntoFile(File file){
        JSONArray nodes = new JSONArray();
        for(int i = 0; i < graph.nodeCount(); ++i) {
            JSONObject oneNode = new JSONObject();
            oneNode.put("name",
String.valueOf(graph.getNodeByIndex(i).getName()));

            JSONArray location = new JSONArray();
            location.add(0, graph.getNodeByIndex(i).getLocation().x);
            location.add(1, graph.getNodeByIndex(i).getLocation().y);
            oneNode.put("location", location);

            JSONArray adjacencyList = new JSONArray();

            for(int j = 0; j < graph.getNodeByIndex(i).edgeCount(); ++j){
                JSONArray edgeInfo = new JSONArray();
                edgeInfo.add(0,
String.valueOf(graph.getNodeByIndex(i).getEdgeByIndex(j).getEndNodeName())
);
```

```
                edgeInfo.add(1,
graph.getNodeByIndex(i).getEdgeByIndex(j).getWeight());
                adjacencyList.add(edgeInfo);
            }
            oneNode.put("adjacencyList", adjacencyList);
            nodes.add(oneNode);
        }

        try (FileWriter writer = new FileWriter(file)) {
            writer.write(nodes.toJSONString());
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "ERROR FILE OPEN");
        }
    }

    private int parseInt(){
        String str = txtfEdgeWeight.getText();
        int totalNum = 0;
        if(str.charAt(0) == '0' || str.charAt(0) == '-'){
            JOptionPane.showMessageDialog(null, "number must be >= 0!");
            return 0;
        }

        for(int i = 0; i < str.length(); ++i){
            if((int)str.charAt(i) < 48 || (int)str.charAt(i) > 57) {
                JOptionPane.showMessageDialog(null, "Edge weight must be
only integer!");
                return 0;
            }
            totalNum = totalNum * 10 + (int)(str.charAt(i)) - 48;
        }
        return totalNum;
    }

    public void nextStep(){
        if (drawingPanel.isAlgorithm){
            if(algorithmStepNum != graphStates.size() - 1) {
                graph = graphStates.get(++algorithmStepNum).getGraph();
                drawingPanel.updateGraph(graph);
                txtaLog.setText("");
                for(int i = 0; i <= algorithmStepNum; ++i){
                    txtaLog.append(graphStates.get(i).getStr() + '\n' +
'\n');
                }
```

```
            }
            else
                JOptionPane.showMessageDialog(null, "Algorithm`s work
end!");
        }
        else
            JOptionPane.showMessageDialog(null, "Algorithm don`t start!");
    }

    public void setUpCloseMainThreadAlgorithm(){
        closeMainThreadAlgorithm = !closeMainThreadAlgorithm;
    }
}
```