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

# ОТЧЕТ

**по учебной практике**

**Тема: Кратчайшие пути в графе. Алгоритм Форда-Беллмана**

Студент гр. 8383 _____ Бессуднов Г.И.

Студент гр. 8383 _____ Дейнега В.Е.

Студентка гр. 8383 _____ Кормщикова А.О.

Руководитель _____ Фирсов М.А.

Санкт-Петербург
2020

**ЗАДАНИЕ**
**НА УЧЕБНУЮ ПРАКТИКУ**

Студент Бессуднов Г.И. группы 8383

Студент Дейнега В.Е.  группы 8383

Студентка Кормщикова А.О. группы 8383

Тема практики: Кратчайшие пути в графе. Алгоритм Форда-Беллмана

Задание на практику:

Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.

Алгоритм: Форда-Беллмана.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 08.07.2020

Дата защиты отчета: 10.07.2020

| | | |
|---|---|---|
| Студент | | Бессуднов Г.И. |
| Студент | | Дейнега В.Е. |
| Студентка | | Кормщикова А.О. |
| Руководитель | | Фирсов М.А. |

## АННОТАЦИЯ

В данной работе по учебной практике реализуется и визуализируется пошагово алгоритм Форда-Беллмана с сопровождающими работу текстовыми пояснениями. Разработанное приложение было реализовано на языке программирования Java. Приложение оснащено графическим интерфейсом, который является ясным и удобным для пользователя. Взаимодействие с графическими элементами осуществлено с помощью мыши. Текстовые пояснения для удобства вынесены в область логгера. Также реализован ввод данных с файла. Разработка программы выполнялась итеративно, были сделаны прототип, 1-2 версия программы и финальная версия.



## SUMMARY

Ford-Bellman algorithm with accompanying text explanations. The developed application was implemented in the Java programming language. The application is equipped with a graphical interface. Interaction with graphic elements is carried out using the mouse. Textual explanations for ease of removal to the logger area. Also implemented data input from a file. Program development was carried out iteratively, prototypes, 1-2 versions of the program and the final version were made.

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Целью данной практической работы является ознакомление с принципами работы в команде над приложением, ознакомление с языком программирования Java и его инструментами для работы над графическими приложениями.
Одной из задач выступает организация совместной работы с помощью системы контроля версий и разделения обязанностей разработки, а также поэтапная разработка приложения с четко выделенными версиями программы.
Другой задачей является изучение и организация алгоритма Форда-Беллмана. Данный алгоритм позволяет найти в графе кратчайшие расстояния от стартовой вершины ко всем другим, допуская наличие ребер с отрицательным весом. Алгоритм применяется в протоколе маршрутизации RIP.

# 1. ТРЕБОВАНИЯ К ПРОГРАММЕ

## 1.1. Исходные Требования к программе

### 1.1.1. Требования к вводу исходных данных

Входные данные будут вводится либо из файла, либо пользователем посредством щелчка мыши на холсте. Пользователь сможет создавать вершины графа, а также ребра. Вершины создаются при клике ЛКМ в области холста и удаляются при клике по ним ПКМ. Ребра создаются путем соединения существующих вершин, при добавлении выскакивает диалоговое окно, в которое пользователь вводит вес ребра, удаление ребер происходит при нажатии на них ПКМ.

Формат данных в файле: Данные в файле подаются в следующем виде: сначала записывается количество ребер n и стартовая вершина, затем последовательно идут n строк имеющих формат:

"Начало(число)" "Конец(число)" "Вес(число)"

Пример:

5

1

1 1 2

1 2 3

3 3 -1

1 2 -4

5 3 4

### 1.1.2. Требования к визуализации

Прототип интерфейса программы представлен в приложении А. Будут реализованы два Layout’а, которые визуализируют два режима работы: режим редактирования (Edit mode), режим работы алгоритма (Watch mode);. Пользователь видит граф на холсте, на ребрах будут отмечены веса, у вершин будет метка о дистанции до них. В Watch mode - пользователь увидит активное ребро, конечную вершину этого ребра, новую метку, которая, возможно, будет записана. Все активные элементы будут выделены иным цветом

1.1.3. Требования к работе режимов.

Переключение между режимами происходит посредством нажатия соответствующей кнопки. Edit mode - предоставляет возможность редактировать граф. Watch mode - позволяет сделать шаг алгоритма, либо же просмотреть полностью работу алгоритма.

1.1.4. Требования к логированию

Логи будут зависит от режима работы. В Edit Mode выводится информация о добавленных или удаленных ребрах или вершинах. В Watch Mode выводится информация о шаге алгоритма: выводится просматриваемое ребро, для него вычисляется новое предполагаемое расстояние - выводится старое значение и новое, пишется сообщение о том, произошла ли замена. При завершении одного этапа релаксации будет выводится информация о завершении этого этапа. Также будет выводится сообщение о начале работы алгоритма, инициализации алгоритма (выводится стартовая вершина(номер) ее метка - "0" и сообщение, в котором сказано, что все остальные вершины имеют метку "бесконечность"), о завершении работы алгоритма (выводится список вершин и расстояние до них).

UML-диаграмма проекта представлена в приложении Б.

**1.2. Уточнение требований после сдачи 1-ой версии**

1.2.1. Обозначение вершин буквами, в т.ч. в формате файлов. Новый формат данных для ввода:

"Начало(буква)" "Конец(буква)" "Вес(число)"

Пример:

5

a

a a 2

a b 3

c d -1

d a -4

b c 4

1.2.2. Рёбра должны быть достаточно толстыми, чтобы щелчок по ним не вызывал затруднений.

1.2.3. Вес ребра должен печататься максимально близко к линии ребра и не должен выводиться ближе к другому ребру.

1.2.4. Сообщения логгера должны быть доступны для копирования.

1.2.5. В логгере должно отмечаться, когда все рёбра перебраны и запускается новый круг.

1.2.6. В логгере должна выводиться причина завершения работы алгоритма.

**1.3. Уточнение требований после сдачи 2-ой версии**

1.3.1. Полупрозрачность весов рёбер - это очень хорошо, но, к сожалению, когда ребро проходит по цифрам, это мешает их видеть. С этим надо что-то сделать (возможное решение: делать число видимым на фоне ребра при наведении курсора).

1.3.2. Возможность остановить процесс после нажатия на "Run full". - Определять, какие вершины входят в отрицательные циклы.

1.3.3. Добавить кнопку "1 cycle", по нажатию на которой будет мгновенно выполняться 1 цикл шагов.

1.3.4. Если метка вершины в результате выполнения шага изменилась, то она должна становиться красной и оставаться красной до начала следующего шага (если метка не изменилась, то оставить прежнее поведение с маленькой задержкой перед почернением).

1.3.4. Оптимизировать алгоритм: если в результате выполнения цикла не было изменений, то алгоритм должен завершать работу.

1.3.6. Для вершин с меткой inf в логгере должен выводиться соответствующий результат, а не 0.

**1.4. Уточнение требований после сдачи 3-ей версии**

- 1.4.1. Кнопку "1 cycle" переименовать в "Run 1 cycle", добавить аналогичную кнопку "1 cycle", выполняющую те же действия, но мгновенно.
- 1.4.2. Определять, какие вершины входят в отрицательные циклы.

# 2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

## 2.1. План разработки

1. Прототип, 3 июля. Включает в себя:
   - Необходимые интерфейсы программы
   - Элементы пользовательского интерфейса. (Визуализация)
   - Возможность переключения между режимами работы
2. Версия 1, 5 июля. Включает в себя:
   - Возможность создать граф
   - Рабочий логгер
   - Рабочий алгоритм, без пошаговой визуализации.
3. Версия 2, 7 июля. Включает в себя:
   - Визуализация алгоритма
   - Считывания из файла
   - Отполированный логгер

## 2.2. Распределение ролей в бригаде

Бессуднов Глеб - реализует редактор графа

Дейнега Виктора - реализует GUI

Кормщикова Арина - реализует алгоритм

# 3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

## 3.1. Структуры данных

### 3.1.1. Структура данных Graph

Класс для хранения информации о графе. Имеет свой собственный интерфейс IGraph, через который с ней взаимодействуют другие части программы и используется в классах GraphEditor и Algorithm (каждый экземпляр класса хранит свою копию).

Класс Graph содержит следующие поля:

public Map<Vertex, ArrayList<Edge> > graph; - контейнер, представляющий из себя список смежности. Поле Key - вершина графа, Value - список ребер, исходящих из этой вершины.

public Vertex startVertex; - поле в котором хранится стартовая вершина для алгоритма

public int countOfVertex - количество вершин в графе

public int countOfEdge - количество ребер в графе

### 3.1.2. Структура данных Vertex

Класс для хранения информации о вершине. Содержит следующие поля:

public int number - номер вершины.

public int distance - дистанция до вершины

public boolean isStart - является ли вершина стартовой

public boolean isCheck - было ли изменении дистанции с метки "бесконечность"

public String name - имя вершины

### 3.1.3. Структура данных Edge

Класс для хранения информации о ребре. Содержит следующие поля:

public Vertex start - начальная вершина

public Vertex end - конечная вершина

public int weight - вес ребра

### 3.1.4. Структура данных AlgorithmMessage

Класс хранящий в себе информацию о шаге алгоритма.Содержит следующие поля:

private String message - сообщение о работе алгоритма.

private Edge viewingEdge - рассматриваемое на шаге алгоритма ребро

private Vertex changeV - конечная вершина ребра (для которой изменяется дистанция)

private Vertex startV - начальная вершина ребра

private boolean isFinish - поле, отображающее закончил ли алгоритм работу

private boolean isEndOfCycle - поле, отображающее закончился ли текущий цикл алгоритма

## 3.2. Основные методы

### 3.2.1. Интерфейс IGraphEditor

public abstract void setEditState(boolean isEditState); - функция для установки состояния редактора. Принимает переменную boolean isEditState, если она равна true, то редактор переходит в режим редактирвоания, в ином случае он переходит в режим просмотра.

public abstract IGraph getGraph(); - возвращает объект типа IGraph, который является редактируемым графом.

public abstract void setCurrentEdge(Edge e); - функция, которая устанавливает и корректно отображает текущее просматриваемое ребро при работе алгоритма. Принимает объект e типа Edge - ребро, визуальное представление которого необходимо обновить.

public abstract void setCurrentVertex(Vertex v); - функция, которая устанавливает и корректно отображает текущую просматриваемую вершину при работе алгоритма. Принимает объект v типа Vertex - вершину, визуальное представление которой необходимо обновить.

public abstract void clearEditor(); - функция для очистки редактора от всего содержимого в нем.

public abstract void loadGraph(Graph graph); - функция для загрузки графа в редактор с последующим созданием его визуального пердставления. Принимает объект graph - граф, который необходимо построить.

public abstract void rerunEditor(); - функция для перезапуска редактора. Возвращает редактор в его вид, перед началом алгоритма. Не удаляет его содержимое.

### 3.2.2. Интерфейс IGraph

public abstract void addVertex(Vertex v); - метод, принимающий вершину Vertex v. Добавляет данную вершину в граф.

public abstract void addEdge(Edge e) - метод, принимающий ребро Edge e. Добавляет данное ребро в граф.

public abstract void deleteEdge(Edge e) -метод, принимающий ребро Edge e, удаляет данное ребро из графа.

public abstract void deleteVertex(Vertex v) - метод, принимающий вершину Vertex v, удаляет данную вершину из графа.

public abstract void setStartVertex(Vertex v) - метод, принимающий вершину Vertex v, устанавливает данную вершину как начальную для алгоритма

### 3.2.2. Интерфейс IAlgorithm

public abstract AlgorithmMessage stepForward() - метод, реализующий один шаг алгоритма (просмотр одного ребра). Во время своей работы создает AlgorithmMessage и возвращает его.

public abstract void initAlgorithm(Graph g) - метод инициализации, принимает граф Graph g. Ставит алгоритм в начальное состояние

### 3.2.3. Методы fileReader

public Graph readFromFile() - метод, в котором происходит считывание данных из файла формата .txt. Возвращает null - если файл/данные в файле

некорректны, при корректных данных - возвращает Grahp построенные по этим данным.

### 3.2.4. Интерфейс ILogger

public abstract void logEvent(String message); - метод, принимающий строку и выводящий ее на экран.

public abstract void logEvent(AlgorithmMessage message); - метод, принимающий AlgorithmMessage message и выводящий поле типа String этого класса.

public abstract void clear(); - метод, очищающий контейнер, отображающийся в качестве логгера.

public abstract String prepare(String message); - метод, подготавливающий строку логга, добавляя символ переноса строки так, чтобы лог корректно отображался на экане

# 4. ТЕСТИРОВАНИЕ

## 4.1. План тестирования

1. Что надо тестировать:
    1. Графический ввод.
    2. Ввод с файла.
    3. Работа алгоритма.
    4. Тестирование корректного вывода логгера
    5. Влияние смены режима работы на данные и работу алгоритма.
    6. Проверка корректной визуализации алгоритма.
    7. Проверка корректности работы графического интерфейса.
2. Как будем тестировать:
    1. Тесты вручную: добавление вершины, ребра, удаление вершины, ребра.
    2. Тесты на ввод корректных файлов, тесты на ввод некорректных файлов
    3. Тесты на графе с одной вершины, несвязный граф, пустой граф, правильность решения на корректных графах
    4. Проверка всех возможных комбинаций клавиш на корректность ответа логера на них. Проверка на переполняемость логгера. Проверка корректности переключения логера между двумя режимами работы программы. (У Edit mode и Watch mode логирование не пересекается).
    5. Проверка корректного сброса прогресса алгоритма при переключении Watch=>Edit=>Watch.
    6. Ручное тестирование с пошаговым прогоном алгоритма на различных графах: с одной вершиной, несвязном графе, пустом графе, корректном графе.
    7. Тесты на работу взаимодействия пользователя и интерфейса: каждая кнопка корректно обрабатывает нажатие. Проверка на блокировку недоступных в разных режимах кнопок.
3. Когда будем тестировать:
    1. Между 1 и 2 версиями программы

2. Между 1 и 2 версиями программы
3. Между прототипом и 1 версией программы. После 2й версии программы.
4. Между прототипом и 1 версией программы. После 2й версии программы.
5. Между 1 и 2 версиями программы.
6. Между прототипом и 1 версией программы. После 2й версии программы.
7. До и после 2й версии.

### 4.2. Тестирование графического ввода

Были проведены теста на добавление вершины, ребра, удаление вершины ребра

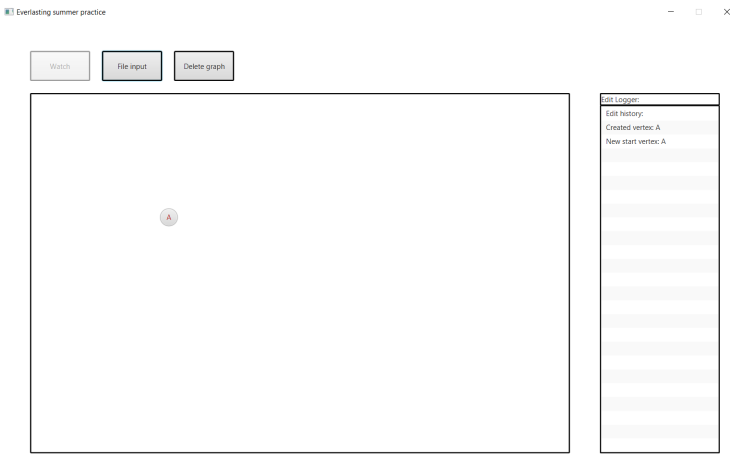


Рисунок 1 - Тест №1. Добавление вершин и ребра.

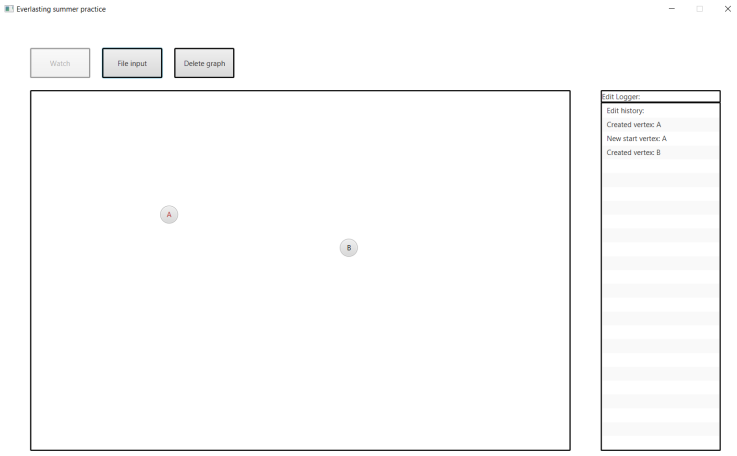


Рисунок 2 - Тест №1. Добавление вершин и ребра.

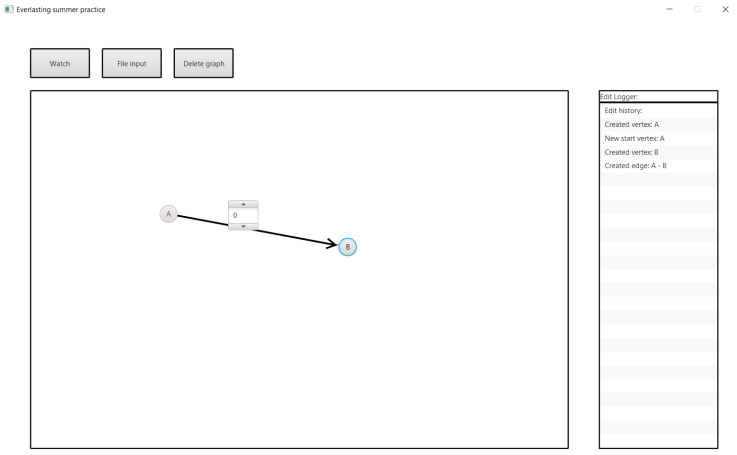


Рисунок 3 - Тест №1. Добавление вершин и ребра.

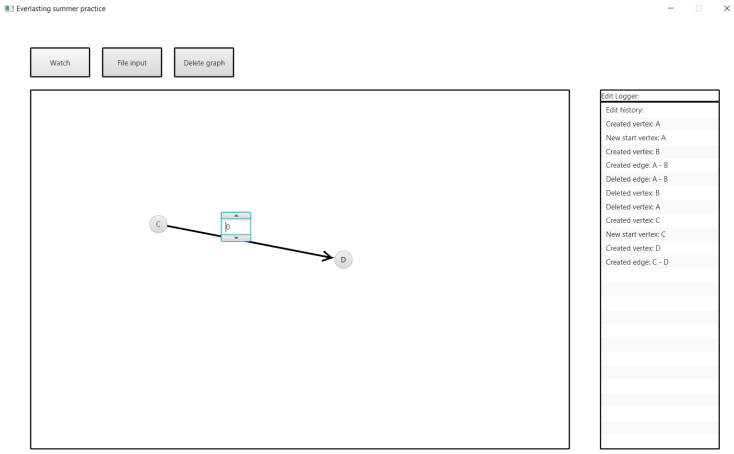


Рисунок 4 - Тест №2. Удаление вершин.

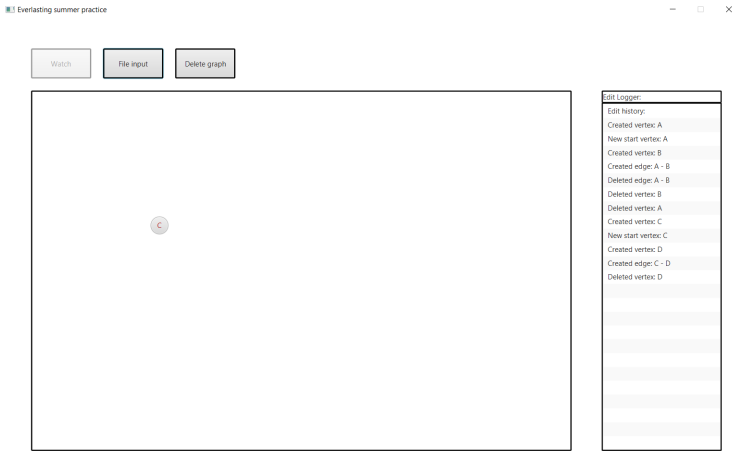


Рисунок 5 - Тест №2. Удаление вершин.

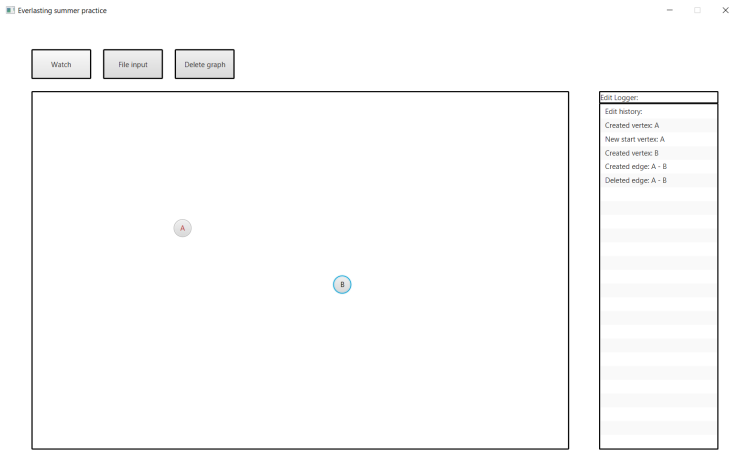


Рисунок 6 - Тест №2. Удаление ребра.

## 4.3. Тестирование ввода с файла

Были проведены тесты на ввод корректных файлов и некорректных.

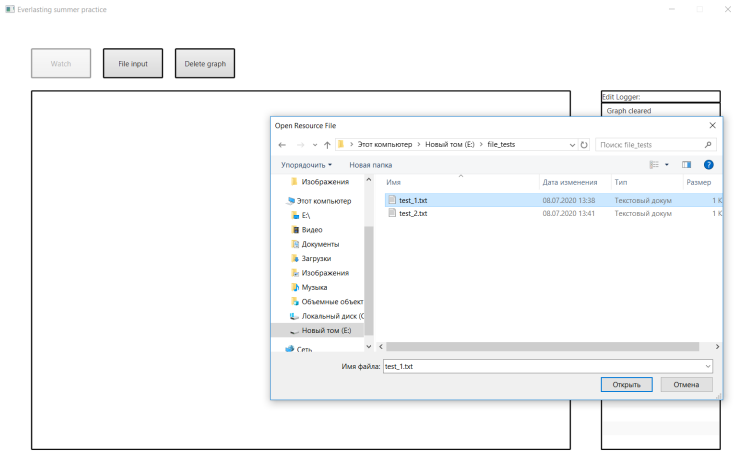


Рисунок 7 - Тест №3. Выбор файла.

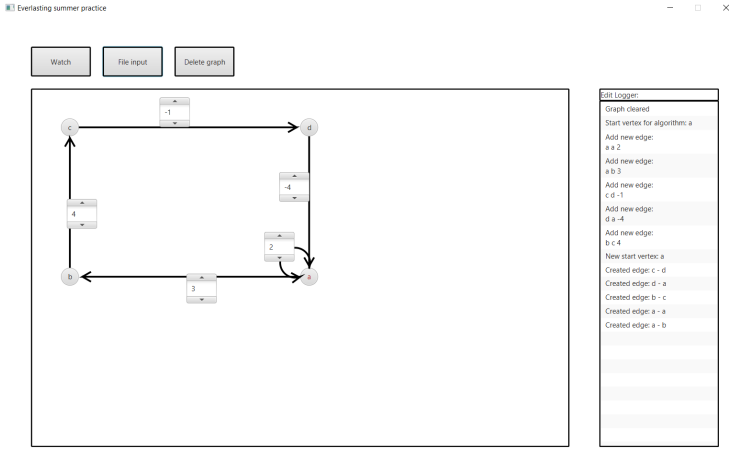


Рисунок 8 - Тест №3. Построение графа по данным из файла.

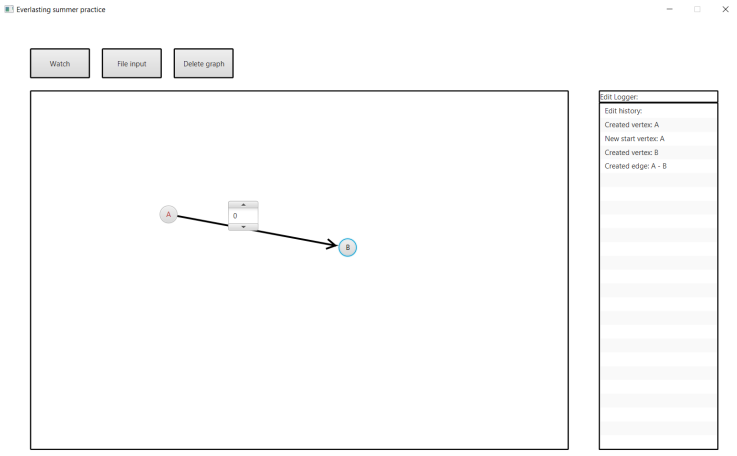


Рисунок 9 - Тест №4. Ввод с файла с некорректными данными.

### 4.4. Тестирование работы алгоритма

Были проведены тесты на различных графах. Для теста 7 лог приведен в приложении Б.


Рисунок 10 - Тест №5. Алгоритм на пустом графе

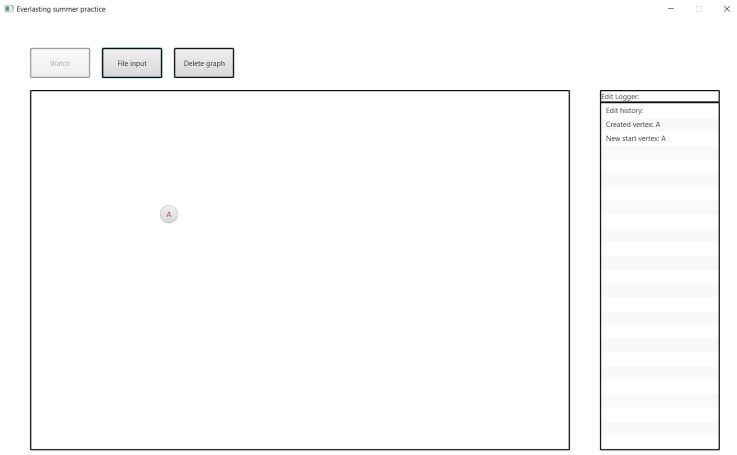


Рисунок 11 - Тест №6. Алгоритм на графе из одной вершины.

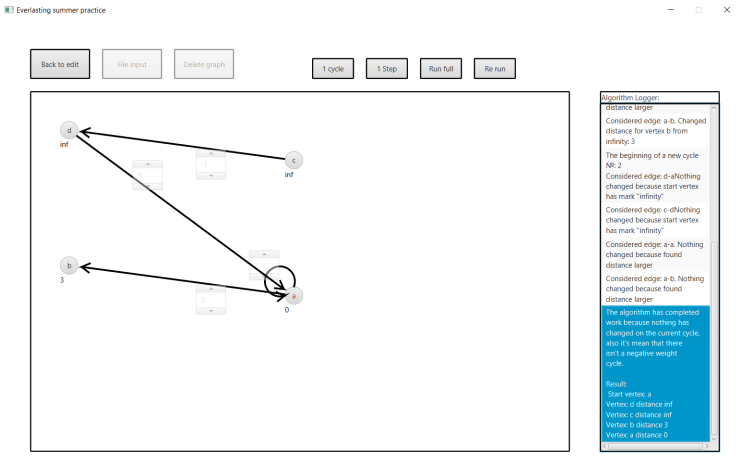


Рисунок 12 - Тест №7. Алгоритм на графе.

### 4.5. Тестирование корректного вывода логгера

Были проведены тесты на перебор всех возможных комбинаций клавиш и корректность ответа логера на них .

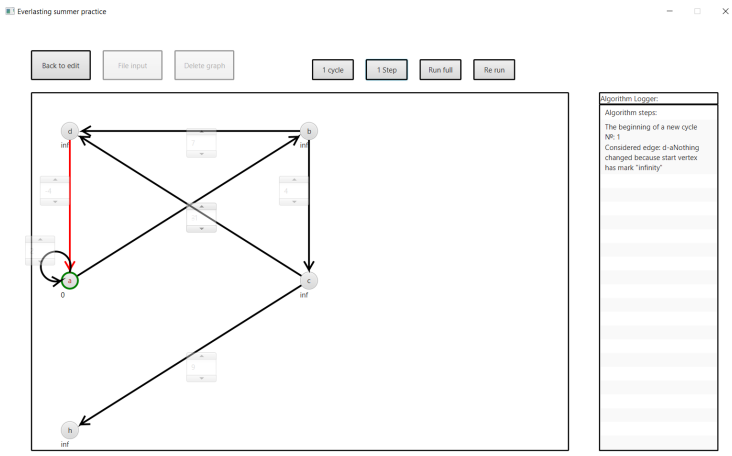


Рисунок 13 - Тест №8. Логгер - 1 шаг алгоритма

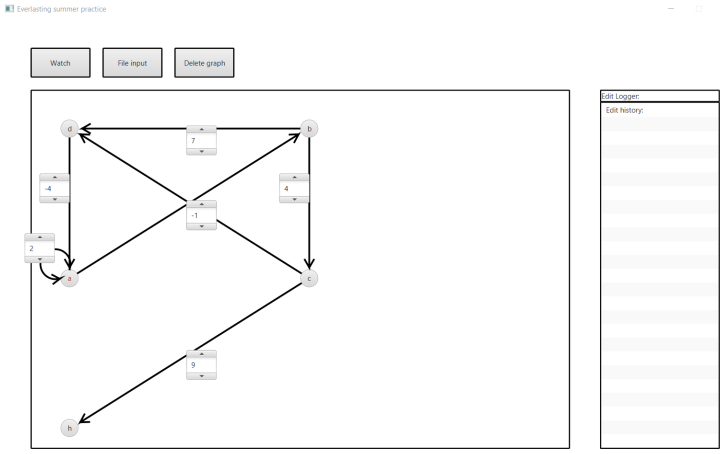


Рисунок 14 - Тест №9. Логгер - Переход в Edit mode.


Рисунок 15 - Тест №10. Логгер - Кнопка Delete graph.

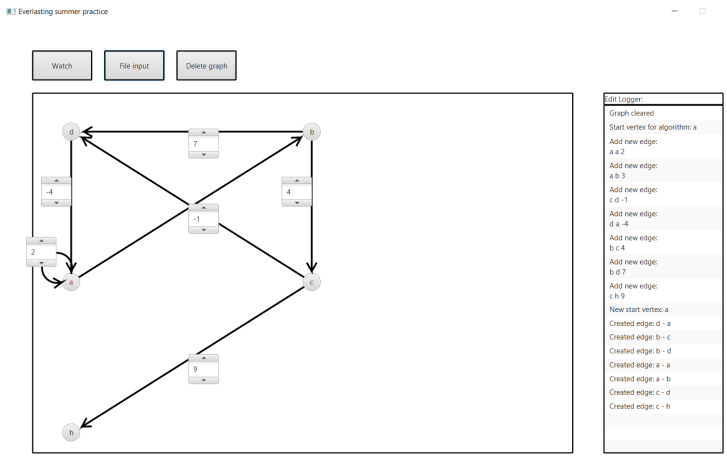


Рисунок 16 - Тест №11. Логгер - Ввод с файла

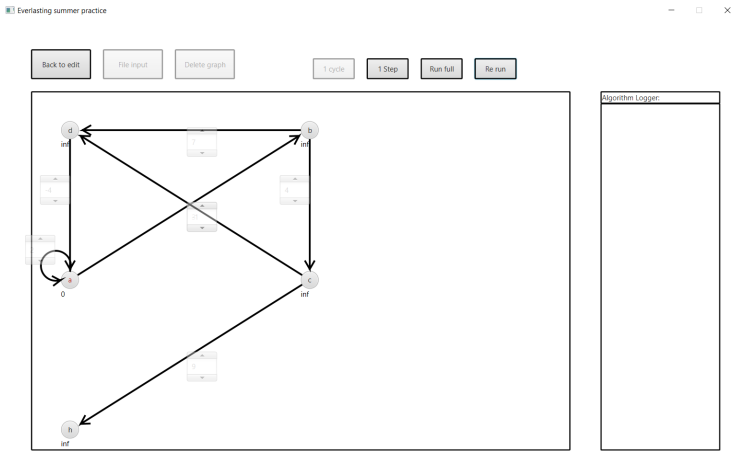


Рисунок 17 - Тест №12. Логгер - Кнопка Re run.

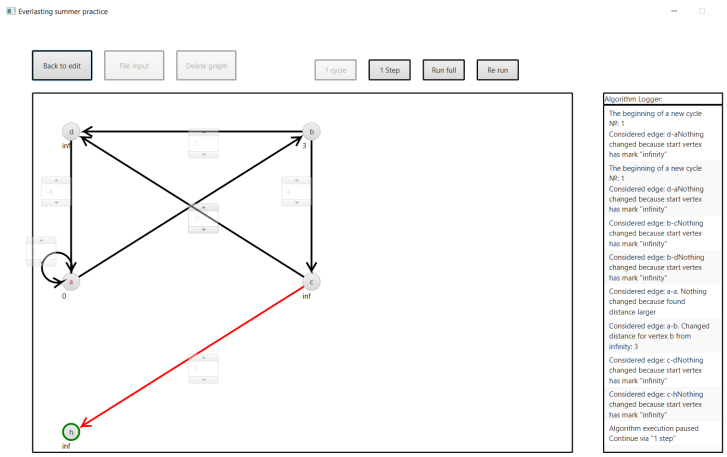


Рисунок 18 - Тест №13. Логгер - Кнопка Stop (при режиме Run full).

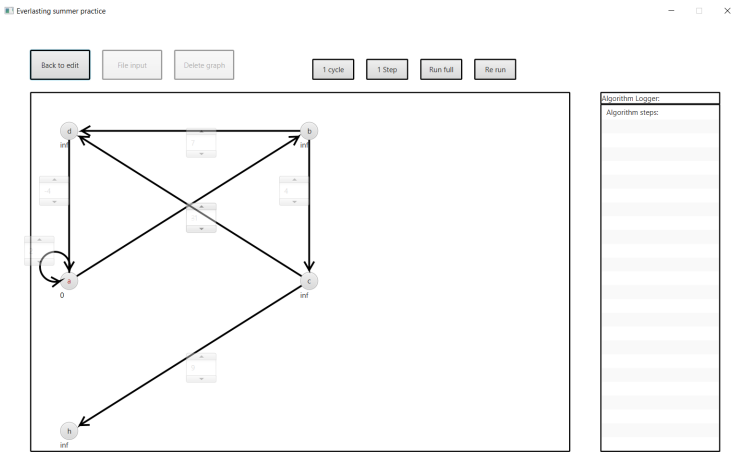


Рисунок 19 - Тест №14. Логгер - переход в Watch mode

## 4.6. Влияние смены режима работы на данные и работу алгоритма.

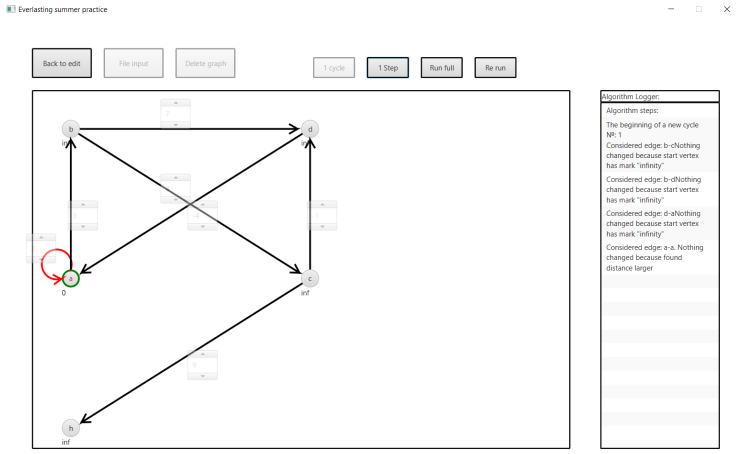


Рисунок 20 - Тест №15. Алгоритм в состоянии Watch mode.

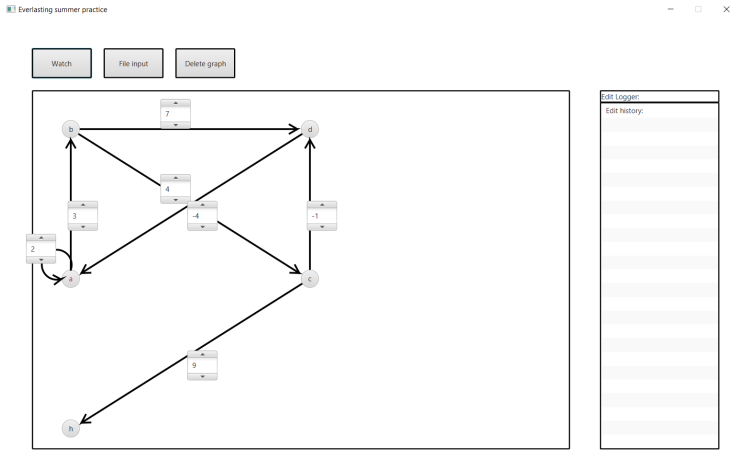


Рисунок 21 - Тест №15. Алгоритм перешел в состояние Edit mode.

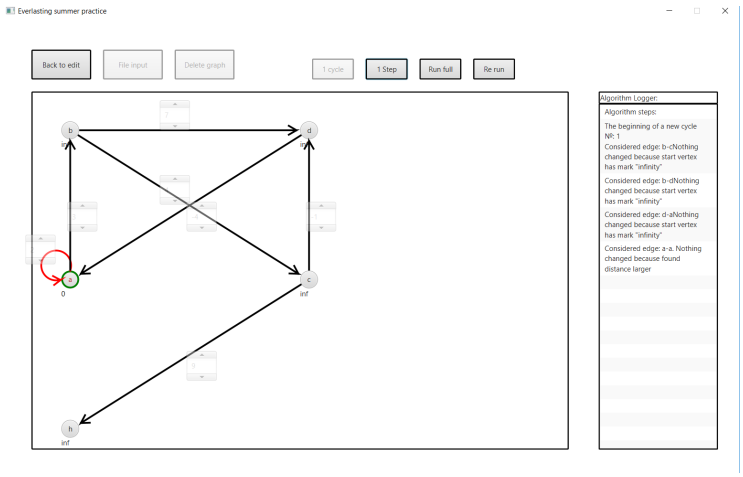


Рисунок 22 - Тест №15. Алгоритм перешел обратно в состояние Watch mode

### 4.7. Проверка корректной визуализации алгоритма.

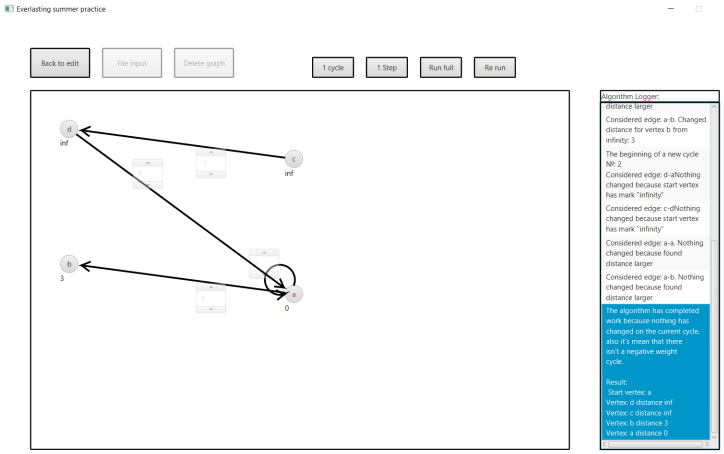


Рисунок 23 - Тест №16. Алгоритм завершил работу.

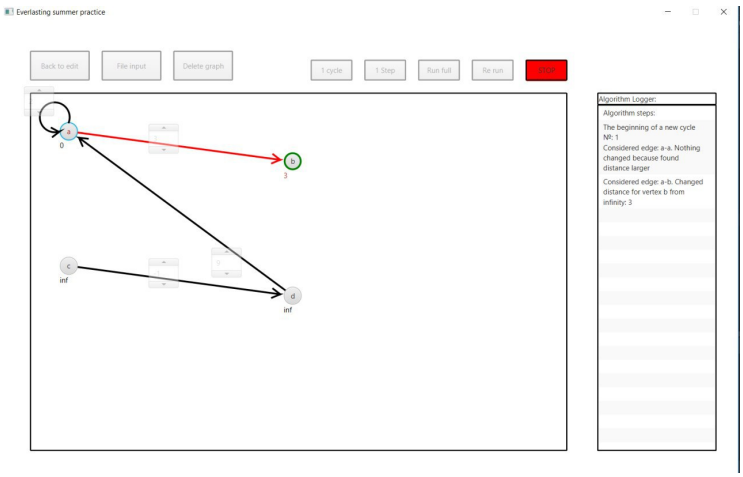


Рисунок 24 - Тест №17. Выделение текущего ребра во время работы алгоритма.

## 4.8. Проверка корректности работы графического интерфейса

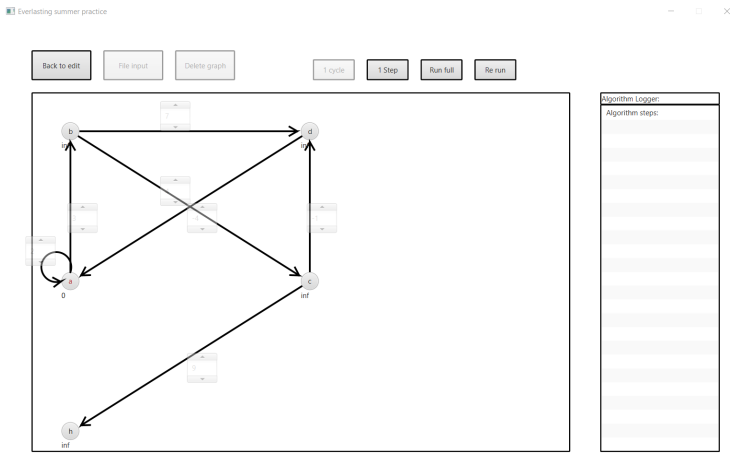


Рисунок 25 - Тест №18. Блокировка кнопок в Watch mode

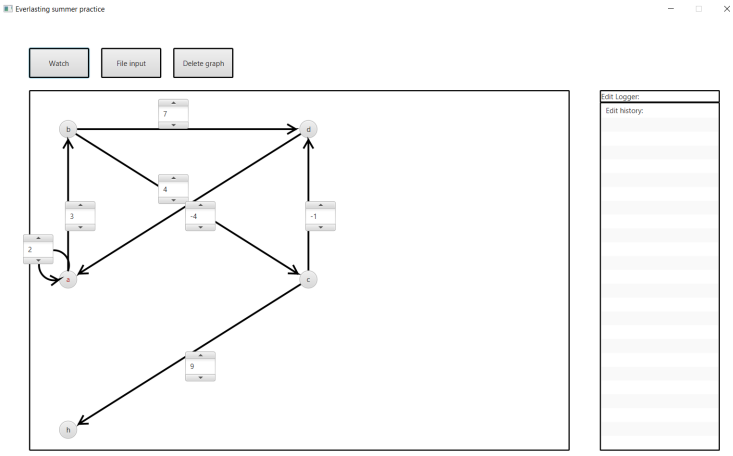


Рисунок 26 - Тест №18. Активные кнопки в Edit mode.

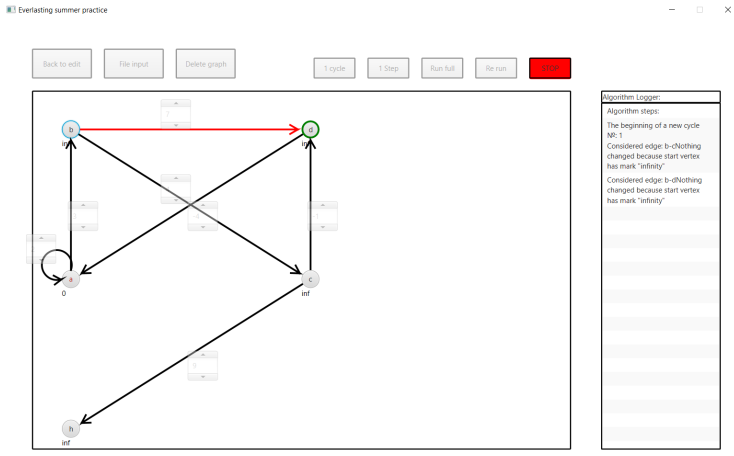


Рисунок 27 - Тест №18. Активные/неактивные кнопки в Run full.

## 4.9. Тесты на нахождение отрицательного цикла в графе

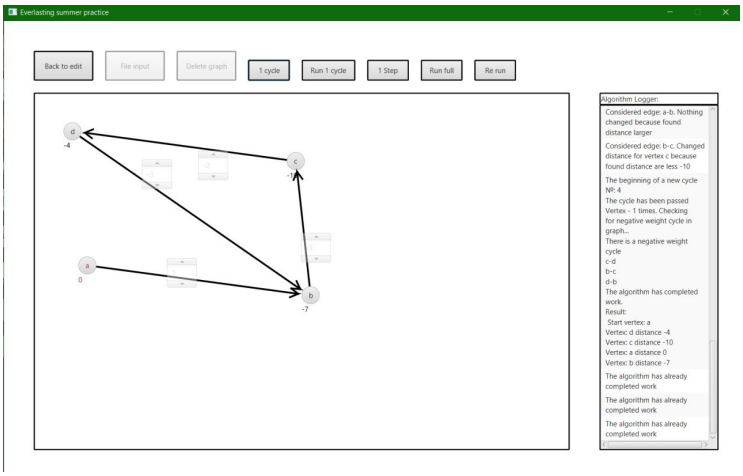


Рисунок 25 - Тест №19.

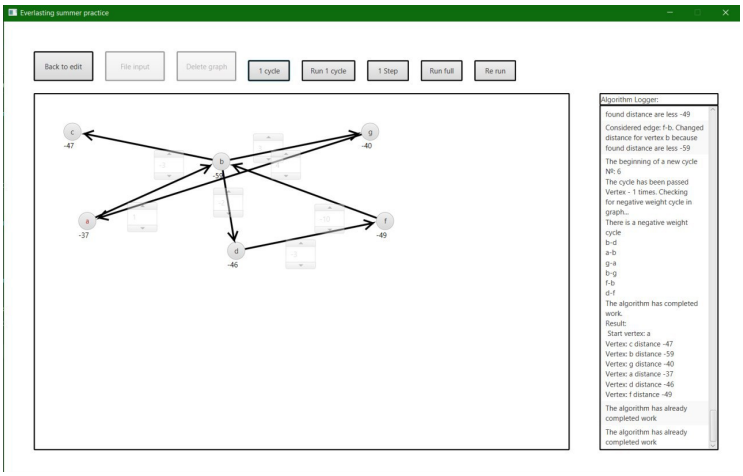


Рисунок 26 - Тест №20.

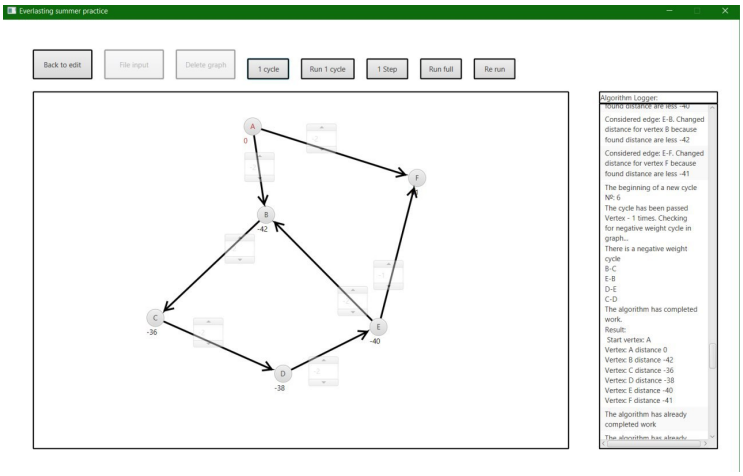


Рисунок 27 - Тест №21

# ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы было успешно реализовано приложение, написанное на языке Java с использованием его инструментов для работы с GUI, которое визуально демонстрирует алгоритм Форда-Беллмана. Так же всеми участниками группы был получен опыт работы в команде с использованием системы контроля версий и распределением обязанностей. Приложение было сделано поэтапно, в четыре версии. Каждая версия является самостоятельной рабочей программой.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. A computer science portal for geeks// www.geeksforgeeks.org. URL: https://www.geeksforgeeks.org/bellman-ford-algorithm-simple-implementation/ (дата обращения: 30.06.2020).

2. Сайт о программировании // metanit.com. URL: https://metanit.com/java/ (дата обращения: 30.06.2020).

3. Java documentation// docs.oracle.com. URL: https://docs.oracle.com/javase/8/javafx/api/index.html (дата обращения: 30.06.2020).

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

https://github.com/GlebDot/SummerPractice/tree/master/Code/defaultArtifact/src/main/java

```
//Algorithm.java
package algorithm;

import logger.*;
import graph.*;

import java.util.ArrayList;
import java.util.Arrays;

/**Class for Ford-Bellman algorithm. Stores his own version of graph and work
with it.
 * <p>Executes algorithm step by step with function {@link#stepForward}</p>
 */
public class Algorithm implements IAlgorithm {
    protected Graph graph;
    protected int indexOuterLoop;
    protected int indexInnerLoop;
    protected int cycleChangeCounter;
    protected ArrayList<Edge> allEdge;
    public boolean isFinish;

    public Algorithm(){
        graph = null;
        indexOuterLoop = 0;
        indexInnerLoop = 0;
        cycleChangeCounter = 0;
        allEdge = new ArrayList<>();
        isFinish = false;
    }

    @Override
    public AlgorithmMessage stepForward() {
        String mes = "";
        if(indexInnerLoop == 0){
            mes = mes + "The beginning of a new cycle №: " +
(indexOuterLoop+1)+"\n";
        }
        if(isFinish){
            return new AlgorithmMessage("The algorithm has already completed
work");// already done work
        }
        if(indexOuterLoop >= (graph.countOfVertex - 1)){
            isFinish = true;
            mes += "The cycle has been passed Vertex - 1 times. Checking for
negative weight cycle in graph... \n";
            boolean check = false;
            for(Edge tmp: allEdge){
                if (tmp.end.distance > (tmp.start.distance + tmp.weight)){
                    check = true;
                    break;
                }
            }
            if(check){
                mes+="There is a negative weight cycle \n";
```

```
            }else{
                mes+="There isn't a negative weight cycle \n";
            }
            mes = mes+ "The algorithm has completed work."+answer();

            return new AlgorithmMessage(mes, null, true, true); // DONE
        }
        if(indexInnerLoop<allEdge.size()){
            Edge tmp = allEdge.get(indexInnerLoop);
            indexInnerLoop++;
            boolean isEndOfCycle = (indexInnerLoop == allEdge.size())?
true:false;
            if(tmp.start.isCheck == true) {
                if(tmp.end.isCheck == false){
                    tmp.end.isCheck = true;
                    tmp.end.distance = tmp.start.distance + tmp.weight;
                    cycleChangeCounter++;
                    mes = mes + "Considered edge:
"+tmp.start.name+"-"+tmp.end.name+". Changed distance for vertex "+
tmp.end.name+" from infinity: "+ tmp.end.distance;
                    return new AlgorithmMessage(mes, tmp,
false,isEndOfCycle );//changed distance from infinity
                }
                if ((tmp.end.distance > (tmp.start.distance + tmp.weight))) {
                    tmp.end.distance = tmp.start.distance + tmp.weight;
                    cycleChangeCounter++;
                    mes = mes + "Considered edge:
"+tmp.start.name+"-"+tmp.end.name+". Changed distance for vertex "+
tmp.end.name+" because found distance are less "+ tmp.end.distance;
                    return new AlgorithmMessage(mes, tmp, false,
isEndOfCycle);//end v distance changed because found distance are less
                }else{
                    mes = mes +  "Considered edge:
"+tmp.start.name+"-"+tmp.end.name+". Nothing changed because found distance
larger";
                    return new AlgorithmMessage(mes, tmp, false,
isEndOfCycle);// nothing changed because found distance larger
                }
            }else{
                mes = mes +  "Considered edge:
"+tmp.start.name+"-"+tmp.end.name+"Nothing changed because start vertex has mark
\"infinity\"";
                return new AlgorithmMessage(mes, tmp, false, isEndOfCycle); //
nothing changed because startV "infinity"
            }
        }else{
            indexOuterLoop++;
            indexInnerLoop = 0;
            if(cycleChangeCounter == 0){
                isFinish = true;
                mes += "The algorithm has completed work because nothing has
changed on the current cycle, also it's mean that there isn't a negative weight
cycle. \n"+answer();
                return new AlgorithmMessage(mes, null, true, true);
            }
            cycleChangeCounter = 0;
            return stepForward();
        }
    }


    @Override
    public void initAlgorithm(Graph g) {
        indexInnerLoop = 0;
        indexOuterLoop = 0;
```

```
        isFinish = false;
        allEdge.clear();
        try {
            this.graph = g.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        Vertex[] allVertex = graph.graph.keySet().toArray(new Vertex[0]);
        Arrays.sort(allVertex);
        for(Vertex v: allVertex){
            allEdge.addAll(graph.graph.get(v));
        }
        if(graph.startVertex == null){
            if(!graph.graph.isEmpty()){
                Vertex[] allVertexNew = graph.graph.keySet().toArray(new
Vertex[0]);
                for(Vertex v: allVertexNew){
                    if(v.number == 0){
                        graph.startVertex = v;
                        break;
                    }
                }
            }
        }
        if(graph.startVertex != null) {
            graph.startVertex.isStart = true;
            graph.startVertex.isCheck = true;
        }
    }

    public String answer(){
        Vertex[] allVertexNew = graph.graph.keySet().toArray(new Vertex[0]);
        Arrays.sort(allVertexNew);
        String ans = "\nResult: \n Start vertex: " +
graph.startVertex.name+"\n";
        for(Vertex v: allVertexNew){
            boolean isInf = v.isCheck;
            ans= ans + "Vertex: " + v.name + " distance " +(isInf?
v.distance:"inf") + "\n";

        }
        return ans;
    }
}

//Ialgorith,.java
package algorithm;

import logger.*;
import graph.*;

public interface IAlgorithm {
    public abstract AlgorithmMessage stepForward();
    public abstract void initAlgorithm(Graph g);
}

//App.java
package app;

import graphEditor.GraphEditor;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
```

```
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.*;
import javafx.scene.canvas.Canvas;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.input.Clipboard;
import javafx.scene.input.ClipboardContent;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import java.util.concurrent.TimeUnit;
import java.io.File;
import java.io.FileReader;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;


import algorithm.*;
import fileReader.fileReader;
import graphEditor.*;
import graph.*;
import javafx.util.Duration;
import logger.AlgorithmMessage;
import logger.Logger;



/**
 * JavaFX App
 */
public class App extends Application {

    private Logger logger;
    private IAlgorithm algorithmSolver;
    private fileReader reader;
    private Timeline timeline;
    private IGraphEditor graphEditor;

    public App(){
        logger = Logger.getInstance();
        algorithmSolver = new Algorithm();
        timeline = null;
    }


    @Override
    public void start(Stage stage) {
        Border defaultBorder = new Border(new BorderStroke(Color.BLACK,
BorderStrokeStyle.SOLID,
        new CornerRadii(1), new BorderWidths(2)));

        Button fileEditButton = setButtonConfiguration(170.0, 50.0, 100.0, 50.0,
"File input");
        Button clearGraphButton = setButtonConfiguration(290.0, 50.0, 100.0,
50.0, "Delete graph");
        Button onWatchModeButton = setButtonConfiguration(50.0, 50.0, 100.0,
50.0, "Watch");
        Button makeAlgStepButton = setButtonConfiguration(610.0, 65.0, 70.0,
35.0, "1 Step");
```

```
        Button runFullAlgButton = setButtonConfiguration(700.0, 65.0, 70.0,
35.0, "Run full");
        Button reRunAlgButton = setButtonConfiguration(790.0, 65.0, 70.0, 35.0,
"Re run");
        Button killRunButton = setButtonConfiguration(880.0, 65.0, 70.0, 35.0,
"STOP");
        Button runOneCycleButton = setButtonConfiguration(520.0, 65.0, 70.0,
35.0, "1 cycle");

        runOneCycleButton.setVisible(false);
        killRunButton.setStyle("-fx-background-color: #ff0000; ");
        killRunButton.setVisible(false);
        reRunAlgButton.setVisible(false);
        makeAlgStepButton.setVisible(false);
        runFullAlgButton.setVisible(false);
        onWatchModeButton.setDisable(true);

        Label loggerLabel = new Label("Edit Logger:");
        loggerLabel.setPrefWidth(200);
        loggerLabel.setLayoutX(1000.0);
        loggerLabel.setLayoutY(120.0);
        loggerLabel.setBorder(defaultBorder);

        ListView<String> loggerTable = new ListView<String>(logger.strList);
        loggerTable.setPrefWidth(200);
        loggerTable.setPrefHeight(580);
        loggerTable.setLayoutX(1000.0);
        loggerTable.setLayoutY(140.0);
        loggerTable.setBorder(defaultBorder);
            loggerTable.setOnMouseClicked(new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent event) {
                ObservableList<String> stList = loggerTable.getItems();
                StringBuilder outStr = new StringBuilder(stList.toString());
                outStr.deleteCharAt(0);
                outStr.deleteCharAt(outStr.length() - 1);
                for(int i = 0; i < outStr.length(); i++){
                    if(outStr.charAt(i) == ','){
                        outStr.setCharAt(i, '\n');
                        outStr.setCharAt(i+1, '\n');
                    }
                }
                Clipboard clipboard = Clipboard.getSystemClipboard();
                ClipboardContent content = new ClipboardContent();
                content.putString(outStr.toString());
                clipboard.setContent(content);
            }
        });

        Pane graphEditorBox = new Pane();
        graphEditorBox.setBorder(defaultBorder);
        graphEditorBox.setLayoutX(50);
        graphEditorBox.setLayoutY(120);
        graphEditorBox.setPrefHeight(600);
        graphEditorBox.setPrefWidth(900);

        Canvas canvas = new Canvas();
        canvas.setHeight(600);
        canvas.setWidth(900);

        graphEditorBox.getChildren().add(canvas);

        graphEditor = new GraphEditor(canvas, onWatchModeButton);
        //handlers
        fileEditButton.setOnAction(new EventHandler<ActionEvent>() {
```

```
            @Override
            public void handle(ActionEvent event) {
                FileChooser fileChooser = new FileChooser();
                fileChooser.setTitle("Open Resource File");
                File file = fileChooser.showOpenDialog(stage);
                if (file != null) {
                    reader = new fileReader(file);
                    Graph g = reader.readFromFile();
                    if (g != null) {
                        graphEditor.clearEditor();
                        graphEditor.loadGraph(g);
                    }
                }

            }
        });

        clearGraphButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                logger.clear();
                graphEditor.clearEditor();
                logger.logEvent("Graph cleared");
            }
        });

        onWatchModeButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                if(onWatchModeButton.getText().equals("Watch")){
                    logger.clear();
                    logger.logEvent("Algorithm steps:");
                    loggerLabel.setText("Algorithm Logger:");
                    fileEditButton.setDisable(true);
                    clearGraphButton.setDisable(true);
                    onWatchModeButton.setText("Back to edit");
                    makeAlgStepButton.setVisible(true);
                    runFullAlgButton.setVisible(true);
                    runOneCycleButton.setVisible(true);
                    reRunAlgButton.setVisible(true);

                    graphEditor.setEditState(false);

algorithmSolver.initAlgorithm((Graph)graphEditor.getGraph());
                }
                else{
                    loggerLabel.setText("Edit Logger:");
                    logger.clear();
                    logger.logEvent("Edit history:");
                    fileEditButton.setDisable(false);
                    clearGraphButton.setDisable(false);
                    onWatchModeButton.setText("Watch");
                    makeAlgStepButton.setVisible(false);
                    runFullAlgButton.setVisible(false);
                    runOneCycleButton.setVisible(false);
                    reRunAlgButton.setVisible(false);
                    graphEditor.setEditState(true);
                }
            }
        });

        makeAlgStepButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                AlgorithmMessage mes = algorithmSolver.stepForward();
```

```
                logger.logEvent(logger.prepare(mes.getMessage()));
                graphEditor.setCurrentEdge(mes.getViewingEdge());
                if (mes.getViewingEdge() != null) {
                    graphEditor.setCurrentVertex(mes.getViewingEdge().end);
                } else {
                    graphEditor.setCurrentVertex(null);
                }
            }
        });

        runFullAlgButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                algorithmSolver.initAlgorithm((Graph)graphEditor.getGraph());
                graphEditor.rerunEditor();
                timeline = new Timeline();
                makeAlgStepButton.setDisable(true);
                runFullAlgButton.setDisable(true);
                runOneCycleButton.setDisable(true);
                reRunAlgButton.setDisable(true);
                onWatchModeButton.setDisable(true);
                killRunButton.setVisible(true);
                timeline.getKeyFrames().add(new KeyFrame(Duration.seconds(2),
new EventHandler<ActionEvent>() {
                    @Override public void handle(ActionEvent actionEvent) {
                        AlgorithmMessage mes = algorithmSolver.stepForward();
                        logger.logEvent(logger.prepare(mes.getMessage()));
                        graphEditor.setCurrentEdge(mes.getViewingEdge());
                        if (mes.getViewingEdge() != null) {

graphEditor.setCurrentVertex(mes.getViewingEdge().end);
                        } else {
                            graphEditor.setCurrentVertex(null);
                        }
                        if(mes.isFinish()){
                            makeAlgStepButton.setDisable(false);
                            runFullAlgButton.setDisable(false);
                            reRunAlgButton.setDisable(false);
                            runOneCycleButton.setDisable(false);
                            onWatchModeButton.setDisable(false);
                            killRunButton.setVisible(false);
                            timeline.stop();
                        }
                    }
                }));
                timeline.setCycleCount(Timeline.INDEFINITE);
                timeline.play();
            }
        });

        reRunAlgButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                logger.clear();
                algorithmSolver.initAlgorithm((Graph)graphEditor.getGraph());
                graphEditor.rerunEditor();
            }
        });

        killRunButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                timeline.stop();
                logger.logEvent("Algorithm execution paused\nContinue via \"1
step\"");
```

```
                makeAlgStepButton.setDisable(false);
                runFullAlgButton.setDisable(false);
                reRunAlgButton.setDisable(false);
                onWatchModeButton.setDisable(false);
                killRunButton.setVisible(false);
                runOneCycleButton.setDisable(false);
            }
        });

        runOneCycleButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                timeline = new Timeline();
                makeAlgStepButton.setDisable(true);
                runFullAlgButton.setDisable(true);
                runOneCycleButton.setDisable(true);
                reRunAlgButton.setDisable(true);
                onWatchModeButton.setDisable(true);
                killRunButton.setVisible(true);
                timeline.getKeyFrames().add(new KeyFrame(Duration.seconds(2),
new EventHandler<ActionEvent>() {
                    @Override public void handle(ActionEvent actionEvent) {
                        AlgorithmMessage mes = algorithmSolver.stepForward();
                        logger.logEvent(logger.prepare(mes.getMessage()));
                        graphEditor.setCurrentEdge(mes.getViewingEdge());
                        if (mes.getViewingEdge() != null) {

graphEditor.setCurrentVertex(mes.getViewingEdge().end);
                        } else {
                            graphEditor.setCurrentVertex(null);
                        }

                        if(mes.isEndOfCycle() || mes.getMessage().indexOf("The
algorithm has already com") != -1){
                            makeAlgStepButton.setDisable(false);
                            runFullAlgButton.setDisable(false);
                            reRunAlgButton.setDisable(false);
                            runOneCycleButton.setDisable(false);
                            onWatchModeButton.setDisable(false);
                            killRunButton.setVisible(false);
                            timeline.stop();
                        }
                    }
                }));
                timeline.setCycleCount(Timeline.INDEFINITE);
                timeline.play();
            }
        });

        Group root = new Group(loggerLabel, fileEditButton, clearGraphButton,
onWatchModeButton,
                makeAlgStepButton, runFullAlgButton, reRunAlgButton,
graphEditorBox,
                loggerTable, killRunButton, runOneCycleButton);
        Scene scene = new Scene(root);

        stage.setScene(scene);
        stage.setTitle("Everlasting summer practice");
        stage.setWidth(1250);
        stage.setHeight(800);
        stage.setScene(scene);
        stage.setResizable(false);
        stage.show();
    }
```

```
    private Button setButtonConfiguration(double offsetX, double offsetY, double
width, double height, String text){
        Border defaultBorder = new Border(new BorderStroke(Color.BLACK,
BorderStrokeStyle.SOLID,
                new CornerRadii(1), new BorderWidths(2)));
        Button button = new Button(text);
        button.setLayoutX(offsetX);
        button.setLayoutY(offsetY);
        button.setPrefWidth(width);
        button.setPrefHeight(height);
        button.setBorder(defaultBorder);
        return button;
    }

    public static void main(String[] args) {
        launch(args);
    }
}

//FakeMain.java
package app;

public class FakeMain {
    public static void main(String[] args) {
        App.main(args);
    }
}

//fileReader.java
package fileReader;

import graph.Edge;
import graph.Graph;
import graph.Vertex;
import logger.Logger;

import java.io.File;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class fileReader {
    protected File file;
    protected Scanner scn;
    protected Logger log;

    public fileReader(File file) {
        this.file = file;
        log = Logger.getInstance();
        scn = null;
    }

    public Graph readFromFile() {
        String nameF = file.getName();
        if(!nameF.matches(".+\\.txt$")){
            log.logEvent(log.prepare("Read file error: wrong file extension"));
            return null;
        }
        try{
            scn = new Scanner(file);
        }catch (IOException e){
            //mes for log
            log.logEvent(log.prepare("Error open file! "+e.getMessage()));
            return null;
```

```
        }
        Graph gr = new Graph();
        Map<Character, Vertex> allVertex = new HashMap<>();

        Character verStart;
        Character verEnd;

        Vertex tmpStart = new Vertex();
        Vertex tmpEnd = new Vertex();
        int weight = 0;
        int countEdge = 0;

        String patternForVertex = "[a-zA-Z]";

        if(scn.hasNextInt()){
            countEdge = scn.nextInt();
        }else{
            log.logEvent(log.prepare("\nRead file error: wrong count of edge
(not a number)"));
            return null;
        }
        if(scn.hasNext(patternForVertex)){
            verStart = scn.next().charAt(0);
            tmpStart = new Vertex(verStart.toString());
            tmpStart.isStart = true;
            allVertex.put(verStart, tmpStart);
            gr.addVertex(tmpStart);
            log.logEvent(log.prepare("Start vertex for algorithm: "+verStart));

        }
        else{
            log.logEvent(log.prepare("Read file error: wrong start vertex name
(not [a-z])"));
            return null;
        }


        for(int i = 0; i<countEdge; i++){
            //reading start v
            if(scn.hasNext(patternForVertex)){
                verStart =  scn.next().charAt(0);
            }else{
                log.logEvent(log.prepare("Read file error: read edge - wrong
start vertex name (not [a-z]) in line №"+(i+3)));
                return null;
            }
            //reading end v
            if(scn.hasNext(patternForVertex)){
                verEnd = scn.next().charAt(0);
            }
            else{
                log.logEvent(log.prepare("Read file error:  read edge - wrong
end vertex name (not [a-z]) in line №"+(i+3)));
                return null;
            }
            //reading weight
            if(scn.hasNextInt()){
                weight = scn.nextInt();
            }
            else{
                log.logEvent(log.prepare("Read file error:  read edge - wrong
weight (not a number) in line №"+(i+3)));
                return null;
            }
            //checking for next line
```

```
            if(!scn.hasNextLine()){
                if((i+1) < countEdge){
                    log.logEvent(log.prepare("Read file error: not enough
data"));
                    return null;
                }
            }else{
                //check for junk in end line
                if(scn.nextLine().length() != 0){
                    log.logEvent(log.prepare("Read file error:  read edge -
Wrong end line №"+(i+3)));
                    return null;
                }
            }
            //end reading 1 line. Add to graph process:
            if(allVertex.containsKey(verStart)){
                tmpStart = allVertex.get(verStart);
            }else{
                tmpStart = new Vertex(verStart.toString());
                allVertex.put(verStart, tmpStart);
                gr.addVertex(tmpStart);
            }

            if(allVertex.containsKey(verEnd)){
                tmpEnd = allVertex.get(verEnd);
            }else{
                tmpEnd = new Vertex(verEnd.toString());
                allVertex.put(verEnd, tmpEnd);
                gr.addVertex(tmpEnd);
            }
            gr.addEdge(new Edge(weight, tmpStart, tmpEnd));
            log.logEvent(log.prepare("Add new edge:\n"+verStart+" "+verEnd+"
"+weight+"\n"));
        }

        if(scn.hasNextLine()){
            log.logEvent(log.prepare("Warning read file: the number of edges is
less than the data in the file, only the right amount is read \n"));
        }

        scn.close();
        return gr;
    }

}

//Edge.java
package graph;

import java.util.Objects;

/**Class for storing information about edge of the graph */
public class Edge {
    public Vertex start;
    public Vertex end;
    public int weight;


    public Edge(int weight, Vertex start, Vertex end){
        this.start = start;
        this.end = end;
        this.weight = weight;
    }
```

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

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Edge edge = (Edge) o;
        return weight == edge.weight && (this.start == edge.start) &&
                this.end == edge.end;
    }



}

//Graph.java
package graph;

import java.util.*;

/**Class for storing information about graph and working with it */
public class Graph implements IGraph {

    public Map<Vertex, ArrayList<Edge> > graph;
    public Vertex startVertex;
    public int countOfVertex;
    public int countOfEdge;

    public Graph(){
        graph = new HashMap<>();
        startVertex = null;
        countOfEdge = 0;
        countOfVertex = 0;
    }


    @Override
    public Graph clone() throws CloneNotSupportedException  {
        Graph newG = new Graph();
        Vertex[] allVertexOld = graph.keySet().toArray(new Vertex[0]);
        for(Vertex v: allVertexOld){
            newG.countOfVertex++;
            newG.graph.put(v.clone(), new ArrayList<>());
        }

        Vertex[] allVertexNew = newG.graph.keySet().toArray(new Vertex[0]);
        Vertex start = null;
        Vertex end = null;
        for(Vertex v: allVertexOld){
            ArrayList<Edge> edgesOld = graph.get(v);
            for(Edge e: edgesOld){
                for(Vertex newVertx: allVertexNew){
                    newG.startVertex = (startVertex!= null && newVertx.number ==
startVertex.number)?newVertx:newG.startVertex;
                    if(newVertx.number == e.start.number){
                        start = newVertx;
                    }
                    if(newVertx.number == e.end.number){
                        end = newVertx;
                    }
                    if(start != null && end != null){
                        newG.addEdge(new Edge(e.weight, start, end));
                        start = end = null;
```

```
                    break;
                }
            }
        }

    }

    return newG;
}

@Override
public void addVertex(Vertex v) {
    countOfVertex++;
    v.setNumber(countOfVertex-1);
    graph.put(v, new ArrayList<>());
}

@Override
public void addEdge(Edge e) {
    boolean isAlreadyExists = false;
    if(graph.containsKey(e.start)&&graph.containsKey(e.end)){
        ArrayList<Edge> forCheck = graph.get(e.start);
        for(Edge tmp: forCheck){
            if(tmp.equals(e)){
                isAlreadyExists = true;
                break;
            }
        }
        if(!isAlreadyExists){
            forCheck.add(e);
            countOfEdge++;
        }
    }
}

@Override
public void deleteVertex(Vertex v) {
    if(!graph.containsKey(v)){
        return;
    }
    ArrayList<Edge> edges = graph.get(v);
    int size = edges.size();
    for(int i = 0; i<size;i++){
        deleteEdge(edges.get(0));
    }
    Vertex[] allV = graph.keySet().toArray(new Vertex[0]);
    ArrayList<Edge> tmp = null;
    ArrayList<Edge> edgeForDel = new ArrayList<>();

    for (Vertex tmpV:allV) {
        tmp = graph.get(tmpV);
        for(Edge tmpE: tmp){
            if(tmpE.end == v){
                edgeForDel.add(tmpE);
            }
        }
    }
    for(int i = 0; i<edgeForDel.size();i++){
        deleteEdge(edgeForDel.get(i));
    }
    graph.remove(v);
    countOfVertex--;
    allV = graph.keySet().toArray(new Vertex[0]);
    for(Vertex tmpV:allV){
        if(tmpV.number == countOfVertex){
```

```
                tmpV.setNumber(v.number);
            }
        }
        if(startVertex == v){
            startVertex = null;
        }
    }

    @Override
    public void deleteEdge(Edge e) {

        if(graph.containsKey(e.start)&&graph.containsKey(e.end)){
            Vertex start = e.start;
            graph.get(start).remove(e);
            countOfEdge--;
        }
    }

    @Override
    public void setStartVertex(Vertex v) {
        if(graph.containsKey(v)) {
            this.startVertex = v;
        }else{
            //some error
        }
    }
}


//Igraph.java
package graph;

public interface IGraph {
    public abstract void addVertex(Vertex v);
    public abstract void addEdge(Edge e);
    public abstract void deleteEdge(Edge e);
    public abstract void deleteVertex(Vertex v);
    public abstract void setStartVertex(Vertex v);
}


//Vertex.java
package graph;

/**class for storing information about vertex of the graph */
public class Vertex implements Comparable<Vertex> {
    public int number;
    public int distance;
    public boolean isStart;
    public boolean isCheck;
    public String name;
    public Vertex(){
        number = -1;
        distance = 0;
        isStart = false;
        isCheck = false;

    }

    public Vertex(String name){
        this();
        this.name = name;
    }

    public void setNumber(int number){
```

```
        this.number = number;
    }

    @Override
    public Vertex clone() throws CloneNotSupportedException {
        Vertex v = new Vertex(name);
        v.setNumber(number);
        return v;
    }

    @Override
    public int compareTo(Vertex o) {
        return this.number - o.number;
    }
}


//GraphEditor.java
package graphEditor;

import java.util.ArrayList;

import graph.*;
import javafx.beans.value.ChangeListener;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.beans.value.ObservableValue;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.Button;
import javafx.scene.control.Spinner;
import javafx.scene.control.SpinnerValueFactory;
import javafx.scene.control.Label;
import javafx.scene.input.*;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.shape.*;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.util.*;
import logger.AlgorithmMessage;
import logger.ILogger;
import logger.Logger;


enum EdgeDrawingStates {DRAW_EDGE, NOT_DRAW_EDGE}

class EdgeVisual extends Group {
    Path line;
    Spinner<Integer> textWeigth;
    Text edgeWeigthLabel;

    protected Button source;
    protected Button finish;
    protected Edge edgeRef;

    private Border selectBorder;

    public Edge getEdgeRef() {
```

```
        return edgeRef;
    }

    @Override
    public boolean equals(Object o) {

        if (o == this) {
            return true;
        }

        if (o instanceof EdgeVisual) {
            EdgeVisual edgeObj = (EdgeVisual)o;
            if (source.getLayoutX() == edgeObj.source.getLayoutX() &&
source.getLayoutY()
            == edgeObj.source.getLayoutY() && finish.getLayoutX() ==
edgeObj.finish.getLayoutX() &&
            finish.getLayoutY() == edgeObj.finish.getLayoutY()) {
                return true;
            }
        }

        return false;
    }

    public EdgeVisual(NodeVisual start, NodeVisual end, boolean isDoubleEdge) {
        source = start;
        finish = end;

        edgeRef = new Edge(0, start.getVertexRef(), end.getVertexRef());

        selectBorder = new Border(new BorderStroke(Color.YELLOW,
BorderStrokeStyle.SOLID,
        new CornerRadii(1), new BorderWidths(2)));

        line = new Path();
        edgeWeigthLabel = new Text("0");
        edgeWeigthLabel.setFont(Font.font("Colibri", FontWeight.SEMI_BOLD, 14));
        //edgeWeigthLabel.setPrefSize(50, 50);
        edgeWeigthLabel.setVisible(false);
        line.setStrokeWidth(3);
        textWeigth = new Spinner<Integer>();

        SpinnerValueFactory<Integer> valueFactory = new
SpinnerValueFactory.IntegerSpinnerValueFactory(-50, 50, 0);
        textWeigth.setValueFactory(valueFactory);


        line.setStroke(Color.BLACK);
        line.setFill(Color.TRANSPARENT);

        line.addEventHandler(MouseEvent.MOUSE_ENTERED, new
EventHandler<MouseEvent>(){
            @Override
            public void handle(MouseEvent event) {
                toFront();
                if (!textWeigth.isDisable()){
                    line.setStroke(Color.YELLOW);
                }
                textWeigth.setBorder(selectBorder);
                edgeWeigthLabel.setLayoutX(event.getX());
                edgeWeigthLabel.setLayoutY(event.getY() - 14);

                edgeWeigthLabel.setVisible(true);
            }
        });
```

```
        line.addEventHandler(MouseEvent.MOUSE_EXITED, new
EventHandler<MouseEvent>(){
            @Override
            public void handle(MouseEvent event) {
                if (!textWeigth.isDisable()){
                    line.setStroke(Color.BLACK);
                }
                textWeigth.setBorder(Border.EMPTY);
                edgeWeigthLabel.setVisible(false);
            }
        });

        if (source.getLayoutX() == finish.getLayoutX() && finish.getLayoutY() ==
source.getLayoutY()) {
            drawLoop();
        } else {
            drawEdge(isDoubleEdge);
        }


textWeigth.getStyleClass().add(Spinner.STYLE_CLASS_SPLIT_ARROWS_VERTICAL);
        textWeigth.setEditable(true);

        textWeigth.valueProperty().addListener(new ChangeListener<Integer>() {
            @Override
            public void changed(ObservableValue<? extends Integer> value,
Integer oldValue, Integer newValue) {
                edgeRef.changeWeight(newValue);
                edgeWeigthLabel.setText(newValue.toString());
            }
        });

        setWeigthTextPosition();

        textWeigth.toFront();
        line.toBack();


        getChildren().add(edgeWeigthLabel);
        getChildren().add(line);
        getChildren().add(textWeigth);
    }

    private void drawLoop() {
        double startX = source.getLayoutX() +  source.getMaxWidth() / 2.0;;
        double startY = source.getLayoutY();

        double endX = source.getLayoutX();
        double endY = source.getLayoutY() + source.getMaxHeight() / 2.0;;

        line.getElements().add(new MoveTo(startX, startY));

        line.getElements().add(new ArcTo(25, 25, 0, endX, endY, true, false));

        double sin = Math.sin(-1.7);
        double cos = Math.cos(-1.7);

        double x1 = (- 1.0 / 2.0 * cos + Math.sqrt(3) / 2 * sin) * 15.0 + endX;
        double y1 = (- 1.0 / 2.0 * sin - Math.sqrt(3) / 2 * cos) * 15.0 + endY;

        double x2 = (1.0 / 2.0 * cos + Math.sqrt(3) / 2 * sin) * 15.0 + endX;
        double y2 = (1.0 / 2.0 * sin - Math.sqrt(3) / 2 * cos) * 15.0 + endY;

        line.getElements().add(new LineTo(x1, y1));
```

```
        line.getElements().add(new MoveTo(endX - line.getStrokeWidth(), endY));
        line.getElements().add(new LineTo(x2, y2));
    }

    private void drawEdge(boolean isDouble) {
        double startX = source.getLayoutX() + source.getMaxWidth() / 2.0;
        double startY = source.getLayoutY() + source.getMaxHeight() / 2.0;

        double endX = finish.getLayoutX() + finish.getMaxWidth() / 2.0;
        double endY = finish.getLayoutY() + finish.getMaxHeight() / 2.0;

        double angle = Math.atan2((endY - startY), (endX - startX));
        double sin = Math.sin(angle);
        double cos = Math.cos(angle);

        startX += (line.getStrokeWidth() + source.getMaxWidth()) / 2.0 * cos;
        startY += (line.getStrokeWidth() + source.getMaxWidth()) / 2.0 * sin;

        endX -= (line.getStrokeWidth() * 2 + source.getMaxWidth() / 2.0) * cos;
        endY -= (line.getStrokeWidth() * 2 + source.getMaxWidth()) / 2.0 * sin;

        angle -= Math.PI / 2;
        sin = Math.sin(angle);
        cos = Math.cos(angle);

        double x1 = (- 1.0 / 2.0 * cos + Math.sqrt(3) / 2 * sin) * 15.0 + endX;
        double y1 = (- 1.0 / 2.0 * sin - Math.sqrt(3) / 2 * cos) * 15.0 + endY;

        double x2 = (1.0 / 2.0 * cos + Math.sqrt(3) / 2 * sin) * 15.0 + endX;
        double y2 = (1.0 / 2.0 * sin - Math.sqrt(3) / 2 * cos) * 15.0 + endY;

        line.getElements().add(new MoveTo(startX, startY));
        if (!isDouble) {
            line.getElements().add(new LineTo(endX, endY));
        } else {
            line.getElements().add(new MoveTo(endX, endY));
        }

        line.getElements().add(new LineTo(x1, y1));
        line.getElements().add(new MoveTo(endX, endY));
        line.getElements().add(new LineTo(x2, y2));
    }

    private void setWeigthTextPosition() {
        double textX = 0.0;
        double textY = 0.0;
        if (source.getLayoutX() == finish.getLayoutX() && source.getLayoutY() ==
finish.getLayoutY()) {

            textX = source.getLayoutX() - 60;
            textY = source.getLayoutY() - 60;

        } else {

            double angle = Math.atan2((finish.getLayoutY() -
source.getLayoutY()),
            (finish.getLayoutX() - source.getLayoutX()));

            double sin = Math.sin(angle);
            double cos = Math.cos(angle);

            double textStartX = source.getLayoutX();
            double textStartY = source.getLayoutY();
```

```
            textX = Math.abs(source.getLayoutX() - finish.getLayoutX()) * 2.0 /
5.0 * cos + textStartX;
            textY = Math.abs(source.getLayoutY() - finish.getLayoutY()) * 2.0 /
5.0 * sin + textStartY;
        }


        textWeigth.setMaxWidth(50);
        textWeigth.setLayoutX(textX);
        textWeigth.setLayoutY(textY);
    }
}


class NodeVisual extends Button {
    private ArrayList<EdgeVisual> edgeRefs;

    private Vertex nodeRef;
    private Label labelRef;

    public Vertex getVertexRef() {
        return nodeRef;
    }

    public void setLabelRef(Label label) {
        labelRef = label;
    }

    public void setNewName(String name) {
        setText(name);
        nodeRef.name = name;
    }

    public Label getLabelRef() {
        return labelRef;
    }

    public void setNewLabelValue(String info) {
        labelRef.setTextFill(Color.FIREBRICK);

        if (info.equals(labelRef.getText())) {
            final Timeline timeline = new Timeline();
            timeline.setCycleCount(1);
            timeline.setAutoReverse(false);
            final KeyValue kv = new KeyValue(labelRef.textFillProperty(),
Color.BLACK, Interpolator.EASE_BOTH);
            final KeyFrame kf = new KeyFrame(Duration.millis(1500), kv);
            timeline.getKeyFrames().add(kf);
            timeline.play();
        } else {
            labelRef.setText(info);
        }
    }

    public NodeVisual(String name) {
        super(name);
        edgeRefs = new ArrayList<EdgeVisual>();
        setShape(new Circle(15));
        setMaxSize(30, 30);
        setMinSize(30, 30);

        nodeRef = new Vertex(name);
    }

    @Override
```

```
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }

        if (o instanceof Vertex) {
            Vertex obj = (Vertex)o;
            return nodeRef.equals(obj);
        }

        return false;
    }

    public void setEdge(EdgeVisual edge) {
        edgeRefs.add(edge);
    }

    public ArrayList<EdgeVisual> getEdges() {
        return edgeRefs;
    }
}
/**Class for editing and bulding graph */
public class GraphEditor implements IGraphEditor {
    private IGraph graph;
    private boolean isEditing;

    private NodeVisual startNode;

    private Canvas canvas;
    private GraphicsContext context;
    private Border hightlightBorder;
    private Pane parentBox;

    private ILogger logger;
    private Button controlButtonRef;


    private ArrayList<NodeVisual> graphNodes;
    private ArrayList<Label> nodeLabels;
    private ArrayList<EdgeVisual> graphEdges;

    private EdgeDrawingStates edgeState;

    private NodeVisual edgeStart;
    private NodeVisual edgeEnd;

    private NodeVisual hightligthedNode;
    private EdgeVisual hightligthedEdge;

    private int graphNodesCount;

    public GraphEditor(Canvas canvas, Button b) {
        hightlightBorder = new Border(new BorderStroke(Color.GREEN,
BorderStrokeStyle.SOLID,
        new CornerRadii(1), new BorderWidths(3)));
        isEditing = true;
        this.canvas = canvas;
        context = canvas.getGraphicsContext2D();
        parentBox = (Pane)canvas.getParent();
        graphNodesCount = 0;
        edgeState = EdgeDrawingStates.NOT_DRAW_EDGE;
        graph = new Graph();

        nodeLabels = new ArrayList<Label>();
        graphNodes = new ArrayList<NodeVisual>();
```

```
        graphEdges = new ArrayList<EdgeVisual>();

        logger = Logger.getInstance();

        controlButtonRef = b;

        //add Vertex
        this.canvas.addEventHandler(MouseEvent.MOUSE_CLICKED, new
EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent event) {
                if (event.getButton().compareTo(MouseButton.PRIMARY) == 0 &&
isEditing) {
                    NodeVisual graphNode = createGraphNodeButton();
                    graphNode.setLayoutX(event.getX());
                    graphNode.setLayoutY(event.getY());
                    parentBox.getChildren().add(graphNode);
                    graphNodes.add(graphNode);

                    logger.logEvent(new AlgorithmMessage("Created vertex: " +
graphNode.getText()));
                    edgeState = EdgeDrawingStates.NOT_DRAW_EDGE;

                    if (graphNodes.size() == 1) {
                        setStartVertex(graphNode);
                    }
                }
            }
        });
    }

    private void setStartVertex(NodeVisual node) {
        if (startNode != null) {
            startNode.setTextFill(Color.BLACK);;
        }

        startNode = node;
        graph.setStartVertex(startNode.getVertexRef());
        startNode.setTextFill(Color.FIREBRICK);

        logger.logEvent(new AlgorithmMessage("New start vertex: " +
startNode.getText()));
    }

    private NodeVisual createGraphNodeButton() {
        String buttonName = "" + (char)('A' + graphNodesCount);
        NodeVisual graphNode = new NodeVisual(buttonName);

        graphNodesCount++;

        graph.addVertex(graphNode.getVertexRef());


        graphNode.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                if (isEditing) {
                    if (edgeState == EdgeDrawingStates.NOT_DRAW_EDGE) {
                        edgeState = EdgeDrawingStates.DRAW_EDGE;
                        edgeDrawBegin(graphNode);
                    } else {
                        edgeState = EdgeDrawingStates.NOT_DRAW_EDGE;
                        edgeDrawEnd(graphNode);
                    }
                }
```

```
            }
        });

        //delete vertex event
        graphNode.addEventHandler(MouseEvent.MOUSE_PRESSED, new
EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent event) {
                if (event.getButton().compareTo(MouseButton.SECONDARY) == 0 &&
isEditing) {
                    graph.deleteVertex(graphNode.getVertexRef());

                    logger.logEvent(new AlgorithmMessage("Deleted vertex: " +
graphNode.getText()));
                    ArrayList<EdgeVisual> edgesToRemove = graphNode.getEdges();
                    for (EdgeVisual edge : edgesToRemove) {
                        parentBox.getChildren().remove(edge);
                        graphEdges.remove(edge);
                    }
                    if (graphEdges.isEmpty()) {
                        controlButtonRef.setDisable(true);
                    }
                    graphNodes.remove(graphNode);
                    parentBox.getChildren().remove(graphNode);
                    if (graphNode.equals(startNode) && !graphNodes.isEmpty()) {
                        setStartVertex(graphNodes.get(0));
                    }
                }
            }
        });

        graphNode.addEventHandler(MouseEvent.MOUSE_PRESSED, new
EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent event) {
                if (event.getButton().compareTo(MouseButton.MIDDLE) == 0 &&
isEditing) {
                    setStartVertex(graphNode);
                }
            }
        });

        return graphNode;
    }

    private EdgeVisual createEdge(NodeVisual start, NodeVisual end) {
        EdgeVisual tryEdge = new EdgeVisual(end, start, true);
        EdgeVisual edge;
        if (graphEdges.contains(tryEdge)) {
            edge = new EdgeVisual(start, end, true);
        } else {
            edge = new EdgeVisual(start, end, false);
        }
        logger.logEvent(new AlgorithmMessage("Created edge: " + start.getText()
+ " - " +
        end.getText()));


        graph.addEdge(edge.getEdgeRef());

        graphEdges.add(edge);
        if (controlButtonRef.isDisable()) {
            controlButtonRef.setDisable(false);
        }
        //Delete edge event
```

```
        edge.addEventHandler(MouseEvent.MOUSE_CLICKED, new
EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent event) {
                if (event.getButton().compareTo(MouseButton.SECONDARY) == 0 &&
isEditing) {
                    logger.logEvent(new AlgorithmMessage("Deleted edge: " +
edge.source.getText() + " - " +
                    edge.finish.getText()));

                    graph.deleteEdge(edge.getEdgeRef());
                    parentBox.getChildren().remove(edge);
                    graphEdges.remove(edge);
                }
            }
        });

        return edge;
    }

    private void edgeDrawBegin(NodeVisual clickedButton) {
        edgeStart = clickedButton;
    }

    private void edgeDrawEnd(NodeVisual clickedButton) {
        edgeEnd = clickedButton;


        EdgeVisual edgeVis = createEdge(edgeStart, edgeEnd);

        if (!parentBox.getChildren().contains(edgeVis)) {
            edgeEnd.setEdge(edgeVis);
            edgeStart.setEdge(edgeVis);

            parentBox.getChildren().add(edgeVis);
        }
    }

    private void prepareGraphEditor() {
        for (NodeVisual node : graphNodes) {

            Label nodeLabel = new Label();
            nodeLabel.setLayoutY(node.getLayoutY() + node.getMinHeight());
            nodeLabel.setLayoutX(node.getLayoutX());

            nodeLabel.setText("inf");

            parentBox.getChildren().add(nodeLabel);
            nodeLabels.add(nodeLabel);
            node.setLabelRef(nodeLabel);
        }

        if (startNode != null) {
            startNode.setNewLabelValue("0");
        }


        for (EdgeVisual edge : graphEdges) {
            edge.textWeigth.setDisable(true);
        }
    }

    private void clearGraphEditorLabels() {
        for (Label label : nodeLabels) {
            parentBox.getChildren().remove(label);
```

```
    }

    for (EdgeVisual edge : graphEdges) {
        edge.line.setStroke(Color.BLACK);
    }

    nodeLabels.clear();

    for (EdgeVisual edge : graphEdges) {
        edge.textWeigth.setDisable(false);
    }

    if (hightligthedEdge != null) {
        hightligthedEdge.line.setStroke(Color.BLACK);
    }

    if (hightligthedNode != null) {
        hightligthedNode.setBorder(Border.EMPTY);
    }
}

private void setEndStartNodesForEdge(Edge e) {
    for (NodeVisual node : graphNodes) {
        if (e.start.name.equals(node.getVertexRef().name)) {
            edgeStart = node;
        }

        if (e.end.name.equals(node.getVertexRef().name)) {
            edgeEnd = node;
        }
    }
}

@Override
public void setEditState(boolean isEditState) {
    isEditing = isEditState;
    if (!isEditing) {
        prepareGraphEditor();
    } else {
        clearGraphEditorLabels();
    }
}

@Override
public IGraph getGraph() {
    return graph;
}


@Override
public void setCurrentEdge(Edge e) {
    if (e == null) {
        if (hightligthedEdge != null) {
            hightligthedEdge.line.setStroke(Color.BLACK);
            hightligthedEdge = null;
        }

        return;
    }

    if (hightligthedEdge != null) {
        hightligthedEdge.line.setStroke(Color.BLACK);
    }
    for (EdgeVisual edgeVis : graphEdges) {
```

```
            if (e.start.name == edgeVis.edgeRef.start.name && e.end.name ==
edgeVis.edgeRef.end.name) {
                hightligthedEdge = edgeVis;
                edgeVis.line.setStroke(Color.RED);
                break;
            }
        }
    }

    @Override
    public void setCurrentVertex(Vertex v) {
        if (v == null) {
            if (hightligthedNode != null) {
                hightligthedNode.setBorder(Border.EMPTY);
                hightligthedNode.getLabelRef().setTextFill(Color.BLACK);
                hightligthedNode = null;
            }

            return;
        }


        if (hightligthedNode != null) {
            hightligthedNode.getLabelRef().setTextFill(Color.BLACK);
            hightligthedNode.setBorder(Border.EMPTY);
        }
        for (NodeVisual node : graphNodes) {
            if (v.name == node.getVertexRef().name) {
                hightligthedNode = node;
                node.setBorder(hightlightBorder);

                if (v.isCheck) {
                    hightligthedNode.setNewLabelValue(new
Integer(v.distance).toString());
                }
                break;
            }
        }
    }

    @Override
    public void clearEditor() {
        graph = new Graph();
        graphNodesCount = 0;
        edgeStart = null;
        edgeEnd = null;
        for (NodeVisual node : graphNodes) {
            parentBox.getChildren().remove(node);
        }

        for (EdgeVisual edge : graphEdges) {
            parentBox.getChildren().remove(edge);
        }

        graphNodes.clear();
        graphEdges.clear();
        controlButtonRef.setDisable(true);
    }

    @Override
    public void rerunEditor() {
        if (hightligthedEdge != null) {
            hightligthedEdge.line.setStroke(Color.BLACK);
        }
```

```
        if (hightligthedNode != null) {
            hightligthedNode.setBorder(Border.EMPTY);
        }

        for (NodeVisual node : graphNodes) {
            node.setNewLabelValue("inf");
        }

        if (startNode != null) {
            startNode.setNewLabelValue("0");
        }
    }

    @Override
    public void loadGraph(Graph graph) {
        double stepX = (parentBox.getWidth() - 150) /
Math.ceil(graph.graph.size() * 1.0 / 2);
        double stepY = (parentBox.getHeight() - 150) /
Math.ceil(graph.graph.size() * 1.0 / 2);

        double coordX = 50;
        double coordY = 50;

        int xOrder = 0;
        int yOrder = 0;

        for(Vertex vertex : graph.graph.keySet()) {
            NodeVisual graphNode = createGraphNodeButton();
            graphNode.setLayoutX(coordX);

            if (xOrder % 2 == 0) {
                graphNode.setLayoutY(coordY);
            } else {
                graphNode.setLayoutY(coordY + 50);
            }
            graphNode.setNewName(vertex.name);
            parentBox.getChildren().add(graphNode);
            graphNodes.add(graphNode);

            if (vertex.isStart) {
                setStartVertex(graphNode);
            }

            coordX += stepX;
            xOrder++;
            if (coordX > parentBox.getWidth() - stepX) {
                coordY += stepY;
                yOrder++;
                xOrder = 0;
                if (yOrder % 2 == 0) {
                    coordX = 75;
                } else {
                    coordX = 50;
                }
            }
        }

        for (ArrayList<Edge> edges : graph.graph.values()) {
            for (Edge e : edges) {
                setEndStartNodesForEdge(e);
                EdgeVisual edgeVis = createEdge(edgeStart, edgeEnd);
                edgeEnd.setEdge(edgeVis);
                edgeStart.setEdge(edgeVis);
                parentBox.getChildren().add(edgeVis);
```

```
                edgeVis.textWeigth.getValueFactory().setValue(e.weight);
            }
        }
    }
}


//IgraphEditor.java
package graphEditor;

import graph.*;

public interface IGraphEditor {
    public abstract void setEditState(boolean isEditState);
    public abstract IGraph getGraph();
    public abstract void setCurrentEdge(Edge e);
    public abstract void setCurrentVertex(Vertex v);

    public abstract void clearEditor();
    public abstract void loadGraph(Graph graph);
    public abstract void rerunEditor();
}


//AlgorithmMessage.java
package logger;

import graph.Edge;
import graph.Vertex;

/**This is class for sending special message to logger from Algorithm
 * Rigth now it contains only a string of information and should be extended
*/
public class AlgorithmMessage {
    private String message;
    private Edge viewingEdge;
    private Vertex changeV;
    private Vertex startV;
    private boolean isFinish;
    private boolean isEndOfCycle;

    public AlgorithmMessage(){
        message = null;
        viewingEdge = null;
        changeV = null;
        startV = null;
        isFinish = false;
        isEndOfCycle = false;
    }

    public AlgorithmMessage(String mes) {
        message = mes;
        viewingEdge = null;
        changeV = null;
        startV = null;
        isFinish = false;
        isEndOfCycle = false;
    }

    public AlgorithmMessage(String mes, Edge viewingEdge, boolean isFinish,
boolean isEndOfCycle) {
        message = mes;
        this.viewingEdge = viewingEdge;
        if(viewingEdge!= null) {
            this.changeV = viewingEdge.end;
```

```
            this.startV = viewingEdge.start;
        }
        else{
            this.changeV = null;
            this.startV = null;
        }
        this.isFinish = isFinish;
        this.isEndOfCycle = isEndOfCycle;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String mes) {
        message = mes;
    }

    public Vertex getChangeV(){
        return this.changeV;
    }

    public  Vertex getStartV(){
        return this.startV;
    }

    public  Edge getViewingEdge(){
        return  this.viewingEdge;
    }

    public boolean isFinish(){
        return this.isFinish;
    }

    public boolean isEndOfCycle() {
        return this.isEndOfCycle;
    }
}

//Ilogger.java
package logger;

import javafx.collections.ObservableList;

public interface ILogger {
    public abstract void logEvent(AlgorithmMessage message);

    public abstract void logEvent(String message);

    public abstract void clear();

    public abstract String prepare(String message);
}


//Logger.java
package logger;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;

//should use singleton and proxy patterns
public class Logger implements ILogger {

    private static Logger instance;
```

```
    public ObservableList<String> strList;

    private Logger(){
        this.strList = FXCollections.observableArrayList("Edit history:");
    }

    public static Logger getInstance() {
        if (instance == null) {
            instance = new Logger();
        }
        return instance;
    }

    @Override
    public void logEvent(AlgorithmMessage message) {
        strList.add(message.getMessage());
    }

    @Override
    public void logEvent(String message) {
        strList.add(message);
    }

    @Override
    public void clear() {
        strList.clear();
    }

    @Override
    public  String prepare(String message){
        StringBuilder str = new StringBuilder(message);
        int i = 0;
        int currLen = 0;
        while((i+currLen) != str.length()){
            if(str.charAt(i+currLen) == '\n'){
                i+=currLen;
                i++;
                currLen = 0;
                continue;
            }
            if(currLen == 29){
                if(str.charAt(i+currLen) == ' ' || str.charAt(i+currLen) ==
'\n'){
                    str.setCharAt(i+currLen, '\n');
                }
                else{
                    while(str.charAt(i+currLen) != ' ' || str.charAt(i+currLen)
== '\n'){
                        currLen--;
                        if(str.charAt(i+currLen) == ' '){
                            str.setCharAt(i+currLen, '\n');
                            break;
                        }
                    }
                }
            }
            else{
                currLen++;
                continue;
            }
            i+=currLen;
            i++;
            currLen = 0;
        }
```

```
            return str.toString();
        }
    }
```

## ПРИЛОЖЕНИЕ Б
## ТЕСТ 7 ЛОГИРОВАНИЕ

Algorithm steps:

The beginning of a new cycle
є: 1
Considered edge: d-a Nothing
changed because start vertex
has mark "infinity"

Considered edge: c-d Nothing
changed because start vertex
has mark "infinity"

Considered edge: a-a. Nothing
changed because found
distance larger

Considered edge: a-b. Changed
distance for vertex b from
infinity: 3

The beginning of a new cycle
є: 2
Considered edge: d-a Nothing
changed because start vertex
has mark "infinity"

Considered edge: c-d Nothing
changed because start vertex

has mark "infinity"

Considered edge: a-a. Nothing
changed because found
distance larger

Considered edge: a-b. Nothing
changed because found
distance larger

The algorithm has completed
work because nothing has
changed on the current cycle

also it's mean that there
isn't a negative weight
cycle.

Result:
Start vertex: a
Vertex: d distance inf
Vertex: c distance inf
Vertex: b distance 3
Vertex: a distance 0