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

ОТЧЕТ

по учебной практике

Тема: Визуализация алгоритма Форда-Беллмана

Студент гр. 9304 _____ Краев Д.В.

Студентка гр. 9304 _____ Аксенова Е.А.

Студент гр. 9304 _____ Ламбин А.В.

Руководитель _____ Фиалковский М.С.

Санкт-Петербург
2021

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Краев Д.В. группы 9304

Студентка Аксенова Е.А. группы 9304

Студент Ламбин А.В. группы 9304

Тема практики: Алгоритм Форда-Беллмана

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на языке программирования Java с графическим интерфейсом.

Алгоритм: алгоритм Форда-Беллмана

Сроки прохождения практики: 1.07.2021 – 14.07.2021

Дата сдачи отчета: 00.07.2021

Дата защиты отчета: 00.07.2021

Студент гр. 9304		Краев Д.В.
Студентка гр. 9304		Аксенова Е.А.
Студент гр. 9304		Ламбин А.В.
Руководитель		Фиалковский М.С.

АННОТАЦИЯ

Основные цели – изучение языка программирования Java, получение опыта работы в команде, разработка визуализации алгоритма Форда-Беллмана на языке программирования Java в виде приложения. В приложении реализовать удобный для пользователя интерфейс для работы с графом.

SUMMARY

The main goals of the hands-on training are to learn the Java programming language, get skills of working in a team, develop the visualization of the Ford-Bellman algorithm in the Java programming language as an application. It is essential to instrument user-friendly interface for working with the grah.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Основные цели – изучение языка программирования Java и получение опыта работы в команде, реализация визуализации алгоритма Форда-Беллмана - поиска минимального пути в графе. Создание удобного для пользователя интерфейса для работы с графом.

Алгоритм Форда-Беллмана – алгоритм поиска кратчайшего пути во взвешенном графе. Алгоритм находит кратчайшие пути от одной вершины до всех остальных путей в графе. В отличие от алгоритма Дейкстры он подходит для графов, имеющих ребра с отрицательным весом. При нахождении отрицательного цикла алгоритм сообщает, что кратчайших путей не существует.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

Приложение будет визуализировать алгоритм Форда-Беллмана, показывать его по шагам. Приложение будет давать возможность редактировать граф с помощью панели инструментов. Созданный граф можно будет сохранить в файл и потом загрузить его обратно в приложение. Программа должна давать возможность разных способов отработки алгоритма: полного и пошагового варианта.

1.1.1. Архитектура проекта.

Приложение будет разработано средствами языка программирования Java. Оно будет состоять из 2 основных частей: внешней и внутренней.

Внешней частью приложения является графический интерфейс и взаимодействие с пользователем. Для реализации графического интерфейса будет использована библиотека JavaFX.

Внутренняя часть приложения – это реализация алгоритмов и механизмов приложения. Работа программы будет заключаться в следующем: приложение предоставляет пользователю интерфейс для работы с графом. С его помощью пользователь задает необходимые требования, и внутренняя часть приложения с помощью реализованных алгоритмов их выполняет и выдает пользователю результат работы.

1.1.2. Формат входных и выходных данных

В качестве входных данных будет выступать созданный пользователем с помощью панели инструментов граф. Также в качестве входных данных может быть файл, содержащий информацию о графе.

В качестве выходных могут выступать полностью отрисованный путь на введённом пользователем графе, а так же длина минимального пути, также действие алгоритма на каждом шаге.

1.1.3. Основные типы данных

В приложении будут использоваться заранее реализованные классы:

1) Vertex

Данный класс будет представлять вершину графа, содержать в себе название и координаты вершин.

2) Line

Класс Line будет представлять ребро графа, содержит в себе вес ребра и информацию о инцидентных вершинах.

3) Graph

Данный класс будет представлять сам граф. Он будет содержать в себе информацию о ребрах и вершинах, добавленных в граф.

1.1.4. Графический интерфейс

Графический интерфейс будет содержать:

1) Холст.

Холст будет занимать большую часть окна приложения. На нем будет происходить построение графа, а также отображаться работа алгоритма. Взаимодействие с холстом происходит с помощью мыши и панели инструментов приложения.

2) Панель инструментов.

Панель инструментов будет располагаться в верхней части окна приложения. С помощью панели инструментов можно будет взаимодействовать с графом: добавлять/удалять вершины и ребра, полностью очищать холст и запускать возможность выбора способа работы алгоритма.

3) Кнопки запуска и перемотки алгоритма.

Кнопки будут находиться в нижней части окна приложения. С помощью кнопок можно будет выбирать, как будет отображаться алгоритм: выдаст конечный ответ, будет самостоятельно отображать каждый шаг или же предоставит пользователю возможность управлять сменой шагов.

4) Всплывающие окна.

На всплывающих окнах будет отображаться длина минимального пути, а также сообщения о работе пользователя с графом или работе самого алгоритма.

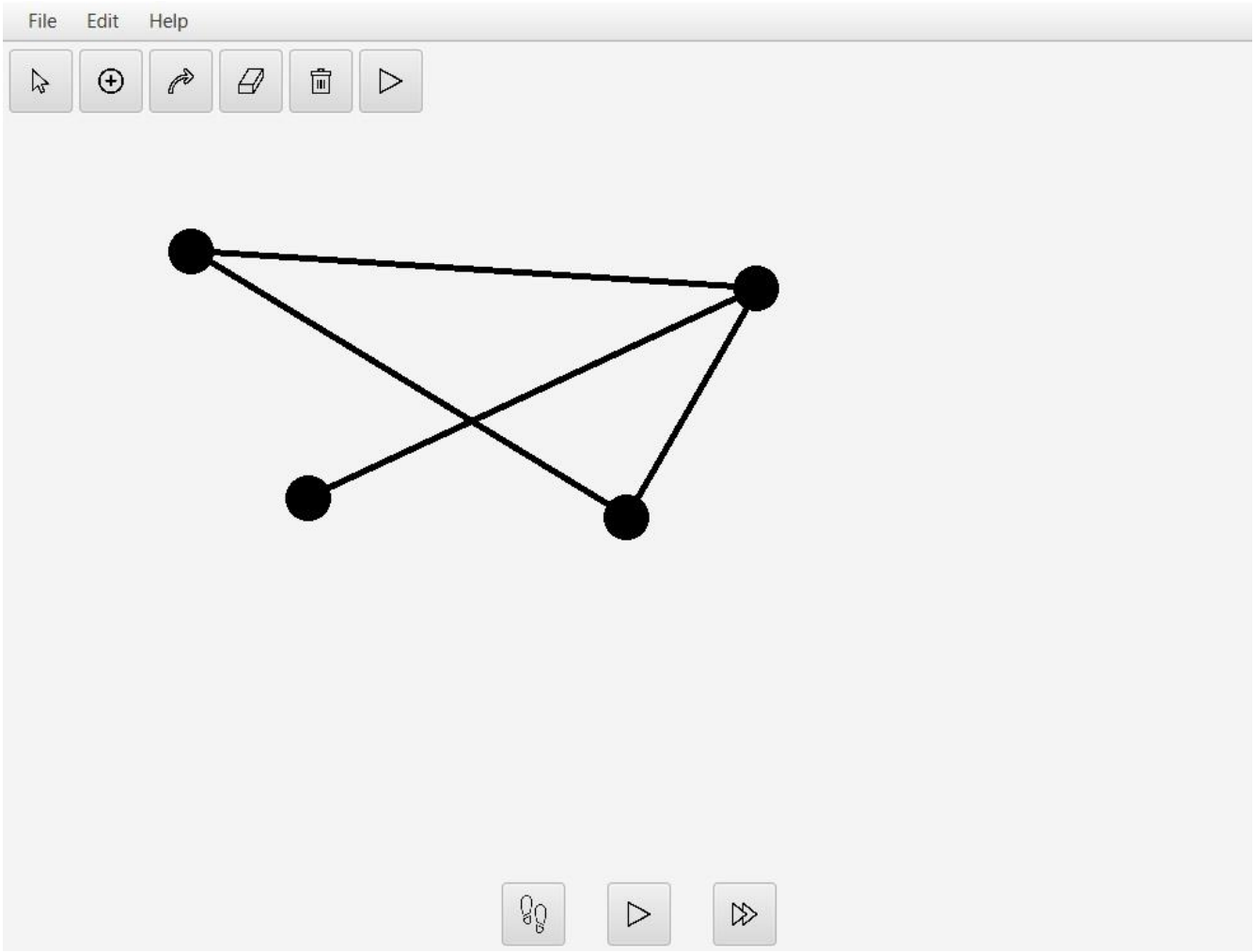

Рисунок 1 – Прототип графического интерфейса.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

До 01.07.2021 – Распределение по бригадам и выбор темы минипроекта.

До 07.07.2021 – Сдача вводного задания.

До 07.07.2021 – Согласование спецификации. Создание прототипа графического интерфейса.

До 10.07.2021 – Сдача второго этапа

До 14.07.2021 – Сдача финальной версии мини-проекта

2.2. Распределение ролей в бригаде

Роли:

1. Лидер — Краев Д.В. — определяет детали проекта, имеет решающее право выбора.

2. Алгоритмист — Аксёнова Е.А. — реализация алгоритма Форда-Беллмана и формирование необходимых данных для фронтенд-разработчиков.

3. Фронтенд — Ламбин А.В. — создание удобного для пользователя графического интерфейса, визуализация работы алгоритма.

4. Документация — Краев Д.В. — создание и ведение отчёта.

5. Тестировщик — Аксёнова Е.А. — unit-тестированиие написанных функций для алгоритма, создание тестовых данных.

6. Мотиватор — Ламбин А.В. — поддержание командного духа.

7. Реализация взаимодействия через файл – Краев Д.В. — реализация возможности работы приложения с помощью загружаемых файлов.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Для реализации интерфейса программы и реализации внутренних алгоритмов были написаны классы, выполняющие разные функции.

3.1.1. Класс Coordinates

Класс *Coordinates* представляет координату вершины графа на поле. Он имеет 2 поля:

- *private double x* – координата графа на оси абсцисс.
- *private double y* – координата графа на оси ординат.

У класса есть 1 конструктор:

- *Coordinates(double x, double y)*

Также класс имеет следующие методы:

- *public void setCord(double x, double y)*

  Устанавливает координату (*x*,*y*).
- *public void moveCord(double dx, double dy)*

  Устанавливает координату (*x+dx*,*y+dy*).
- *public double getX()*

  Возвращает поле *x*.
- *public double getY()*

  Возвращает поле *y*.
- *public double distance(Coordinates a)*

  Возвращает расстояние между этой координатой и координатой *a*.
- *public boolean equals(Coordinates a)*

  Сравнивает эту координату с координатой *a*. Возвращает true, если они примерно равны с погрешностью 10^{-6}, false в ином случае.
- *public String* toString*()*

  Возвращает строковое представление класса.

3.1.2. Класс Line

Класс *Line* представляет ребро графа. Он имеет 4 поля:

- *private final int weight* – вес ребра.
- *private final Vertex startVertex* – начальная вершина ребра.
- *private final Vertex endVertex* – конечная вершина ребра.
- *private final String nameOfLine* - название ребра.

У класса есть 1 конструктор:

- *public Line(int weight, Vertex start, Vertex end)*

Также у класса есть следующие методы:

- *public Vertex getStartVertex()*

  Возвращает начальную вершину ребра.
- *public Vertex getEndVertex()*

  Возвращает конечную вершину ребра.
- *public int getWeight()*

  Возвращает вес ребра.
- *public String getName()*

  Возвращает имя ребра.
- *public Boolean equals(Line obj)*

  Сравнивает это ребро с ребром *obj*. Возвращает true, если id их начальных и конечных вершин совпадают.
- *public int compareTo(Line obj)*

  Сравнивает это ребро с ребром *obj*, возвращает 1, если id конечной вершины этого ребра меньше чем id конечно вершины ребра *obj*, возвращает -1, если больше, и возвращает 0 во всех других случаях.

3.1.3. Класс Vertex

Класс *Vertex* представляет вершину ребра. Он имеет 3 поля:

- *private final int idOfVertex* – идентификатор вершины.
- *private Coordinates cordOfVertex* – координата вершины.

- *private final static double DIAMETER = 30*

  У класса есть 1 конструктор:

- *public Vertex(int id, double x, double y)*

  Также у класса есть следующие методы:

- *public double getDiameter()*

  Возвращает значение поля DIAMETER.

- *public int getId()*

  Возвращает идентификатор вершины.

- *public Coordinates get()*

  Возвращает координату вершины.

- *public double getX()*

  Возвращает координату вершины на оси абсцисс.

- *public double getY()*

  Возвращает координату вершины на оси ординат.

- *public void setCordOfVertex(double x, double y)*

  Устанавливает координату вершины.

- *public double distance(Vertex a)*

  Возвращает расстояние между этой вершиной и вершиной *a*.

- *public double equals(Vertex a)*

  Сравнивает эту вершину с вершиной *a*. Возвращает true, если их идентификаторы и координаты совпадают, false в ином случае.

- *public int compareTo(Vertex obj)*

  Сравнивает эту вершину с вершиной obj. Возвращает 1, если идентификатор этой вершины больше идентификатора вершины *obj*. Возвращает 0 во всех других случаях.

3.1.4. Класс Graph

Класс *Graph* представляет граф. У этого класса есть единственное поле:

- *private TreeMap<Vertex, TreeSet<Line>> matrix* – матрица смежности.

У класса есть 1 конструктор:

- *public Graph()*

Также у класса есть следующие методы:

- *public Boolean addVertex(Vertex newNode)*

  Добавляет вершину в граф. Возвращает true, если добавление произошло успешно, возвращает false в ином случае.
- *public Vertex getVertex(double x, double y)*

  Возвращает вершину находящуюся в заданных координатах.
- *public boolean addLine(Line newLine)*

  Добавляет ребро в граф. Возвращает true, если добавление произошло успешно, возвращает false в ином случае.
- *public Set<Vertex> allVertexes()*

  Возвращает все вершины графа.
- *public HashSet<Line> allLines()*

  Возвращает все ребра графа.
- *public Line getLine(Vertex start, Vertex end)*

  Возвращает ребро, идущее из вершины *start* в вершину *end*, если такого ребра нет, то возвращает null.
- *public TreeMap<Vertex, TreeSet<Line>> getMatrix()*

  Возвращает матрицу смежности.
- *public int getNumOfVertexes()*

  Возвращает количество вершин.
- *public TreeMap<Integer, Integer> makeStartArrayList(Vertex start)*

  Возвращает Map, в котором проинициализированы пути до всех вершин в графе от начальной.
- *public public void deleteAll()*

  Очищает матрицу смежности.
- *public void deleteLine(Vertex start, Vertex end)*

Удаляет ребро, идущее из вершины *start* в вершину *end*.

- *public void deleteVertex(double x, double y)*
  Удаляет вершину в заданной координате.
- *public TreeMap<Integer, String> makePathList(Vertex start)*
  Возвращает Map, в котором проинициализированы вершины, с помощью которых были улучшены пути до данных вершин.
- *public Integer getMinPath(Vertex end, TreeMap<Integer, Integer> destinations, TreeMap<Integer, String> path) throws UnsupportedOperationException*
  Возвращает величину минимального пути из вершины *start* в вершину *end*.
- *public ArrayList<Vertex> minPathArray(Vertex start, Vertex end, TreeMap<Integer, String> path)*
  Возвращает минимальный путь из вершины *start* в вершину *end*.
- *private String minPathString(Vertex start, Vertex end, TreeMap<Integer, String> path)*
  Возвращает строковое представление минимального пути из вершины *start* в вершину *end*.

3.1.5. Класс MainWindowController

Класс *MainWindowController* – это класс, с помощью которого происходит управление приложением. Он содержит в себе реализацию интерфейса программы.

Он содержит следующие поля:

- *private Graph graph* – поле, содержащее граф.
- *private Button cursorButton,*
  addNodeButton,
  addLineButton,
  deleteLineButton,
  deleteAllButton,
  playButton,

oneStepForwardButton,

playForwardButton,

getResultButton - поля, содеражщие кнопки приложения.

- *private Canvas canvas* – поле, содержащее холст.
- *private short state* – поле, содержащее номер состояния программы.
- *private int idVertex* – поле, содержащее id последней созданной вершины.
- *private Vertex startLine* – поле, содержащее вершину нового ребра.
- *private Vertex startVertex* – поле, содержащее начальную вершину
- *private Vertex endVertex* – поле, содержащее конечную вершину.

Также граф содержит следующие методы:

- *private void cursorButtonClick(ActionEvent event)*

  Обрабатывает нажатие курсора.
- *private void addNodeButtonClick(ActionEvent event)*

  Переключает программу в режим добавления вершин.
- *private void addLineButtonClick(ActionEvent event)*

  Переключает программу в режим добавления ребер.
- *private void deleteLineButtonClick(ActionEvent event)*

  Переключает программу в режим удаления ребер.
- *private void deleteAllButtonClick(ActionEvent event)*

  Удаляет граф.
- *private void playButtonClick(ActionEvent event)*

  Запускает алгоритм.
- *private void oneStepForwardButtonClick(ActionEvent event)*

  Перематывает алгоритм на шаг вперед.
- *private void playForwardButtonClick(ActionEvent event)*

  Полностью показывает все шаги работы алгоритма.
- *private void getResultButtonClick(ActionEvent event)*

  Запускает алгоритм и выводит результат работы.
- *private void CanvasClick(MouseEvent event)*

Взаимодействие с холстом.

- *private void drawVertex(Canvas canvas, double x, double y, Color color)*
  Рисует вершину.
- *private void drawLine(Canvas canvas, Vertex start, Vertex end, Color color)*
  Рисует ребро.
- *private void drawText(Canvas canvas, String text, double x, double y, Color color)*
  Показывает текст.
- *private void drawGraph(Canvas canvas, Graph graph, Color color)*
  Рисует граф.
- *private void removeSelected()*
  Удаляет выбранный элемент.

3.1.6. Класс MainWindow

Данный класс представляет основное окно приложения. Он содержит в себе метод *main*. В нем программа запускает приложение. Также класс содержит в себе еще 1 метод:

Public void start(Stage stage)

Метод запускает приложение.

3.1.7. Класс GraphWriterReader

Класс представляет собой инструмент, который позволяет сохранять граф в файл и читать граф с файла. Он содержит следующие методы:

- *public static void write(Graph graph, Path path)*
- *public static void write(Graph graph, String pathStr)*
  Методы, записывающие граф в файл.
- *public static Graph read(Path path)*
- *public static Graph read(String pathStr)*
  Методы, считывающие граф с файла.

3.1.8. Класс GraphState

Класс представляет с собой состояние графа. Он содержит 1 поле:

- *private byte[] grState* - граф в байтовом представлении.

Класс содержит 1 конструктор:

- *public GraphState(Graph graph)*

Также класс содержит 1 метод:

- *public Graph get()*

Создает из байтового представления граф и возвращает его.

3.1.9. Класс GraphStates

Класс представляет собой инструмент, хранящий состояния графа. Он содержит 2 поля:

- *private ArrayList<GraphState> states* – поле, хранящее состояния графа.
- *private int curIndex* – поле, содержащее индекс текущего состояния.

Класс содержит 1 конструктор:

- *public GraphStates()*

Также граф содержит следующие методы:

- *public Graph next()*

Переключает текущее состояние на следующее, возвращает его.

- *public Graph getCur()*

Возвращает текущее состояние

- *public Graph prev()*

Переключает текущее состояние на предыдущее, возвращает его

- *public void add(Graph graph)*

Если текущее состояние – последнее в массиве, то добавляет состояние в конец массива, если текущее состояние – не последнее, то удаляет все состояния после текущего, далее добавляет состояние в конец массива. В конце работы переключает текущее состояние на добавленное.

3.2. Графический интерфейс

Для взаимодействия с пользователем и удобного создания графа был разработан удобный графический интерфейс.

3.2.1. Холст

Холст занимает большую часть окна. На нем происходит создание и редактирование графа, а также воспроизводится алгоритм Форда-Фалкерсона.

3.2.2. Панель инструментов и кнопки запуска алгоритма

Панель инструментов находится в верхней части окна. С ее помощью можно создать и редактировать граф. Панель имеет следующие функции:

- Перетаскивание вершин.
- Создание/удаление вершин.
- Создание/удаление ребер.
- Удаление графа.
- Отмена/возвращение изменений.

  Также 2 предыдущие функции можно запустить с помощью сочетаний клавиш Ctrl+Z/Ctrl+Y соответственно.

В нижней части окна интерфейс имеет кнопки запуска алгоритма. С их помощью можно:

- Запустить алгоритм и вывести результат.
- Полностью отобразить алгоритм по шагам.
- Отобразить следующий шаг.

3.2.3. Работа с файловой системой.

С помощью графического интерфейса можно взаимодействовать с файловой системой. Были реализованы следующие функции:

- Сохранение/загрузка файла

  Можно запустить с помощью сочетаний клавиш Ctrl+S/Ctrl+O соответственно.
- Сохранение изображения графа.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

Было проведено тестирование графического интерфейса. В его ходе были протестированы все элементы управления в приложении. Было выявлено, что интерфейс соответствует ожидаемой производительности и функциональности. Не было выявлено ни 1 дефекта, даже в неожидаемых сценариях работы.

4.2 Тестирование кода алгоритма

С помощью библиотеки JUnit было проведено тестирование кода алгоритма и основных его классов. Для этого были написаны следующие unit-тесты, проверяющие корректную работу отдельных элементов алгоритма:

- testVertex - проверяет корректную работу методов класса Vertex.
- testCoordinates - проверяет корректную работу методов класса Coordinates.
- testLine - проверяет корректную работу методов класса Line.
- testGraph - проверяет корректную работу методов класса Graph.

Ниже приведены результаты тестирования.

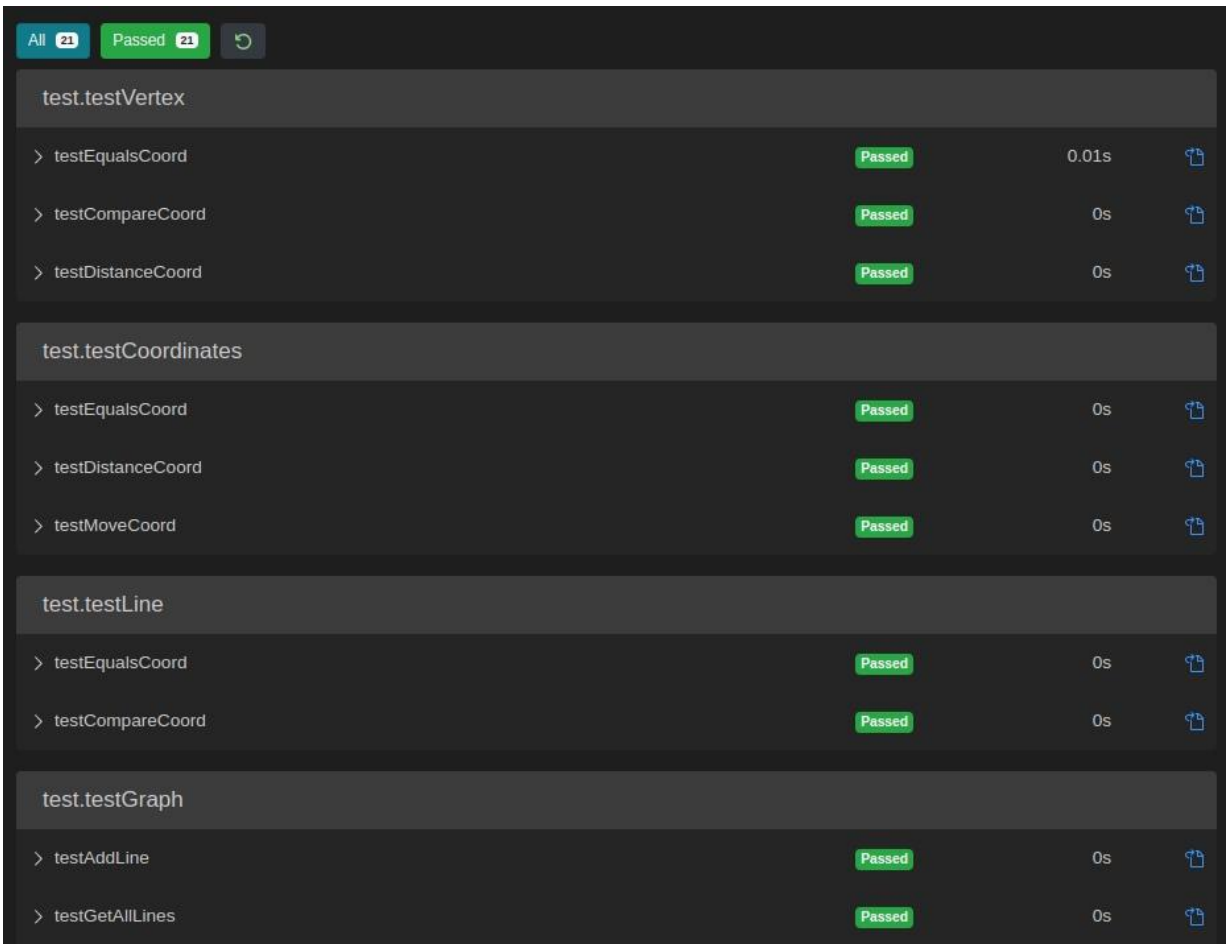

Рисунок 2 - Результаты тестирования.

ЗАКЛЮЧЕНИЕ

В течение практики был изучен язык программирования Java. Разработано приложение, визуализирующее алгоритм Форда-Беллмана. Для приложения был разработан графический интерфейс. Изучены средства создания графического интерфейса. Произведено тестирование алгоритма и графического интерфейса. Реализованы все задуманные функции, а также ряд новых.