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 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 и получение опыта работы в команде. Во время практики необходимо в команде разработать приложение на данном языке. Приложение должно визуализировать алгоритм Форда-Беллмана.

SUMMARY

The main goal is to learn the Java programming language and gain experience working in a team. During the practice, it is necessary to develop an application in this language in the team. The application should visualize the Ford-Bellman algorithm.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Основные цели – изучение языка программирования Java и получение опыта работы в команде. Во время практики необходимо разработать приложение, визуализирующее алгоритм Форда-Беллмана.

Алгоритм Форда-Беллмана – алгоритм поиска кратчайшего пути во взвешенном графе. В отличие от алгоритма Дейкстры он подходит для графов, имеющих ребра с отрицательным весом.

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.

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 интерфейс и взаимодействие с пользователем. Для реализации 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.

1.1.3. Основные типы данных

В приложении будут использоваться заранее реализованные классы:

1) Node

Данный класс будет представлять вершину графа, содержать в себе название и информацию о ребрах.

2) Line

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

3) Graph

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

1.1.4. Графический интерфейс

Графический интерфейс будет содержать:

1) Поле

Поле будет занимать большую часть окна приложения. Оно будет показывать ход алгоритма и сам граф, с которым можно будет взаимодействовать с помощью мыши и панели инструментов.

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, указывать вершину, от которой нужно найти наименьшие пути.

3) Кнопки запуска и перемотки алгоритма.

Кнопки будут находиться в нижней части окна приложения. С помощью кнопок можно будет запускать и перематывать алгоритм.

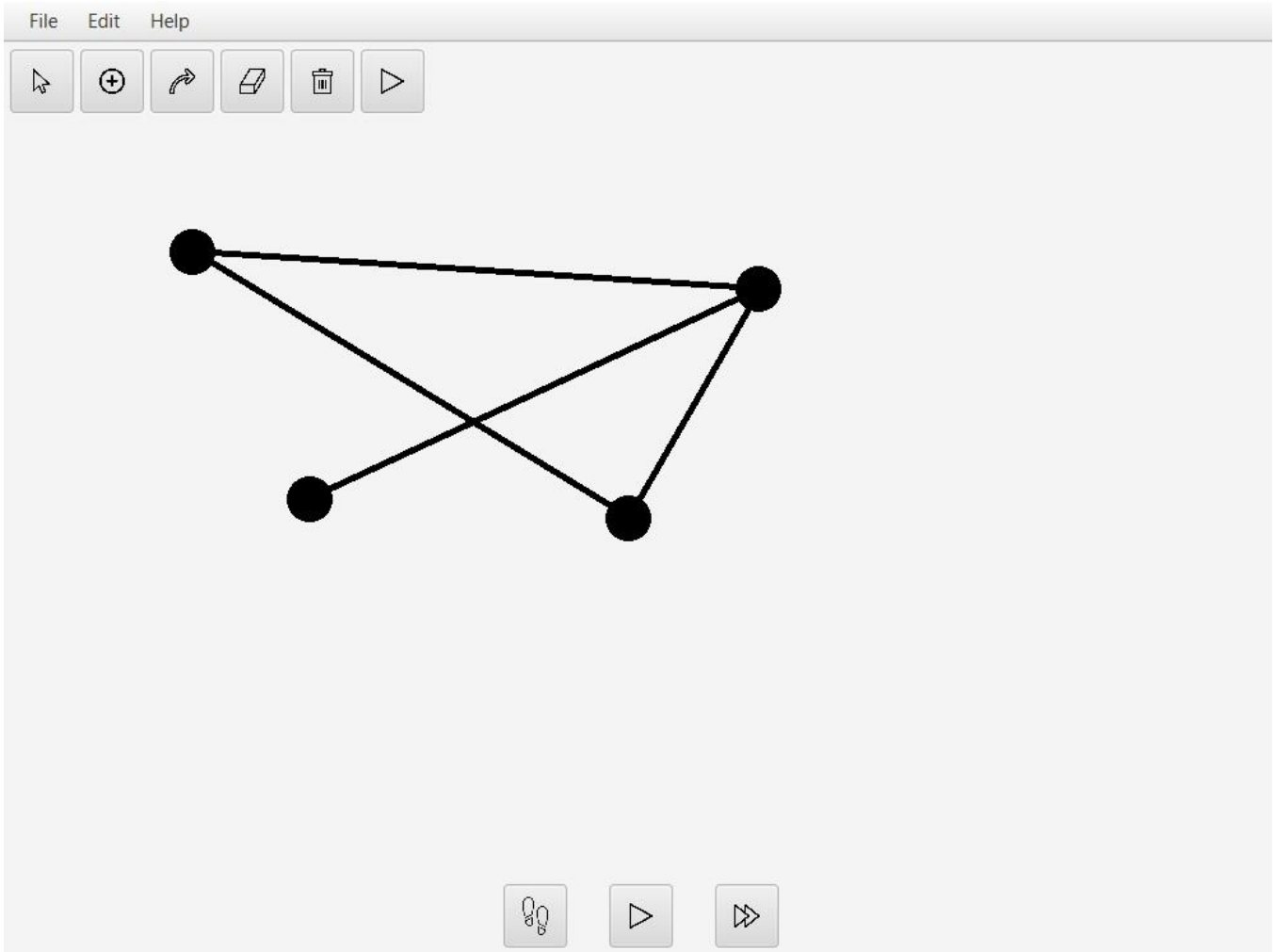

Рисунок 1 – Прототип графического интерфейса.

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.
2. Алгоритмист — Аксёнова Е.А.
3. Фронтэнд — Ламбин А.В.
4. Документация — Краев Д.В.
5. Тестировщик — Аксёнова Е.А.
6. Мотиватор — Ламбин А.В.
7. Реализация взаимодействия через файл – Краев Д.В.

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 finalString nameOfLine* - нахвние 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)*

- *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, Boolean> makeVisitedList(Vertex start)*

- *public TreeMap<Integer, String> makePathList(Vertex start)*

- *public Integer getMinPath(Vertex start, Vertex end, TreeMap<Integer, Integer> destinations, TreeMap<Integer, Boolean> visited, Integer valuePath, 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.*