**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по учебной практике**

Тема: язык программирования Java

|  |  |  |
| --- | --- | --- |
| Студенты гр.4383 | Хафизов И.М., Ахметшина Р.Р. |  |
| Руководитель | Шолохова О.М. |  |

Санкт-Петербург

2016

**ЗАДАНИЕ**

**на Учебную практику**

|  |  |  |
| --- | --- | --- |
| Студенты: Хафизов И.М., Ахметшина Р.Р. | | |
| Группа 4383 | | |
| Тема практики: язык программирования Java | | |
| Задание на практику:  Визуализация алгоритма Дейкстры на языке Java. | | |
| Сроки прохождения практики: 22.06.2016 – 05.07.2016 | | |
| Дата сдачи отчета: 01.07.2016 | | |
| Дата защиты отчета: 04.07.2016 | | |
|  | | |
| Студенты гр.4383 | Хафизов И.М., Ахметшина Р.Р. |
| Руководитель | Шолохова О.М. |

**Аннотация**

Проект реализует и визуализирует алгоритм Дейкстры написанный на языке программирования java. Алгоритм Дейкстры решает задачу нахождения кратчайших путей от определенной вершины до всех остальных вершин в графе. Граф считывается из файла. Для реализации алгоритма был решен определенный ряд задач, например: способ построения графа; выполнены различные тестирования для выявления работоспособности алгоритма. Для визуализации алгоритма были решены задачи: выбор системы визуализации; создание прототипа проекта и др.

**содержание**

[ВВЕДЕНИЕ 5](#_Toc455368321)

[1. Теоретические сведения 6](#_Toc455368322)

[1.1. Описание алгоритма. 6](#_Toc455368323)

[1.2. Сложности алгоритма. 6](#_Toc455368324)

[1.3. Доказательство корректности. 6](#_Toc455368325)

[1.4. Пример работы алгоритма. 7](#_Toc455368326)

[2. Требования к проекту 11](#_Toc455368327)

[2.1 Требования к реализации алгоритма. 11](#_Toc455368328)

[2.2. Решение задач для реализации алгоритма. 11](#_Toc455368329)

[3. Входные и выходные данные 13](#_Toc455368330)

[3.1. Входные данные. 13](#_Toc455368331)

[3.2. Выходные данные. 13](#_Toc455368332)

[4. Структуры данных и алгоритм работы программы 15](#_Toc455368333)

[4.1. Структуры данных алгоритма. 15](#_Toc455368334)

[4.2. Функции алгоритма. 15](#_Toc455368335)

[4.3. Алгоритм работы программы. 16](#_Toc455368336)

[5.Тестирование программы 16](#_Toc455368337)

[ЗАКЛЮЧЕНИЕ 19](#_Toc455368338)

[Список использованных источников 20](#_Toc455368339)

[Приложение: код программы 21](#_Toc455368340)

# **ВВЕДЕНИЕ**

**Цель:** изучить синтаксис языка программирования Java и реализовать алгоритм Дейкстры (нахождение длины минимальных путей от одной вершины до всех остальных вершин).

**Содержание:** данный проект содержит в себе файлы, реализующие работу алгоритма нахождения длин кратчайших путей от вершины до всех остальных вершин. Граф считывается из файла. На экране выводится пошаговая работа алгоритма и графическое представление графа.

# 1. Теоретические сведения

## Описание алгоритма.

Алгоритм Дейкстры — алгоритм на графах. Находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса.

*Формальное объяснение*

В процессе работы алгоритма Дейкстры поддерживается множество S V, состоящее из вершин v, для которых δ(s, v) уже найдено. Алгоритм выбирает вершину u V\S с наименьшим d[u], добавляет u к множеству S и производит релаксацию всех рёбер, выходящих из u, после чего цикл повторяется. Вершины, не лежащие в S, хранятся в очереди Q с приоритетами, определяемыми значениями функции d. Предполагается, что граф задан с помощью списков смежных вершин.

*Не формальное объяснение*

Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до a. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

## 1.2. Сложности алгоритма.

Вершины хранятся в некоторой структуре данных, поддерживающей операции изменения произвольного элемента и извлечения минимального. Каждая вершина извлекается ровно один раз, то есть, требуется O(V) извлечений. В худшем случае, каждое ребро приводит к изменению одного элемента структуры, то есть, O(E) изменений. Если вершины хранятся в простом массиве и для поиска минимума используется алгоритм линейного поиска, временная сложность алгоритма Дейкстры составляет O(V \* V + E) = O(V²). Если же используется очередь с приоритетами, реализованная на основе двоичной кучи (или на основе set), то мы получаем O(V log V + E log E) = O(E log V). Если же очередь с приоритетами была реализована на основе кучи Фибоначчи, получается наилучшая оценка сложности O(V log V + E).

## Доказательство корректности.

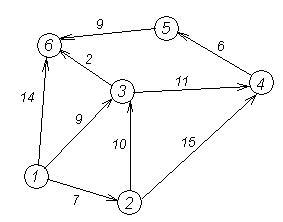
Предположим, алгоритм был запущен на некотором графе из вершины u и выдал неверное значение расстояния для некоторых вершин, причем v — первая из таких вершин (первая в смысле порядка, в котором алгоритм выдавал вершины). Пусть w — ее предок в кратчайшем пути из u в v. Заметим, что расстояние до w подсчитано верно по предположению

Пусть найденное алгоритмом dist'[w] < dist[v]. Тогда рассмотрим последнюю релаксацию ребра, ведущего в v: (s, v). Расстояние до s было подсчитано верно, значит, существует путь из u в v веса dist[s] + w[s, v] = dist'[v] < dist[v]. Противоречие

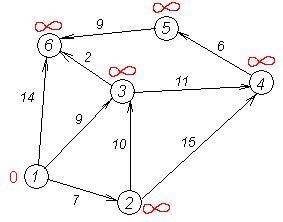
Пусть найденное алгоритмом dist'[w] > dist[v]. Тогда рассмотрим момент обработки вершины w. В этот момент было релаксировано ребро (w, v), и, соответственно, текущая оценка расстояния до вершины v стала равной dist[v], а в ходе следующих релаксаций она не могла уменьшиться. Противоречие Таким образом, алгоритм работает верно. Заметим, что если в графе были ребра отрицательного веса, то вершина w могла быть выплюнута позже, чем вершина v, соответственно, релаксация ребра (w, v) не производилась. Алгоритм Дейкстры работает только для графов без ребер отрицательного веса!

## Пример работы алгоритма.

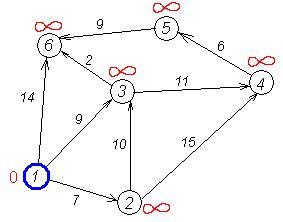
Рассмотрим работу алгоритма на примере графа, показанного на рисунке. Требуется найти расстояния от 1-й вершины до всех остальных.

****

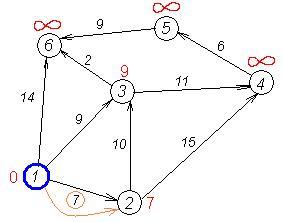
Кружками обозначены вершины, линиями — пути между ними (ребра графа). В кружках обозначены номера вершин, над ребрами обозначена их «цена» — длина пути. Рядом с каждой вершиной красным обозначена метка — длина кратчайшего пути в эту вершину из вершины 1.



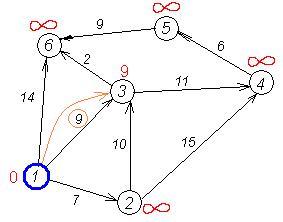
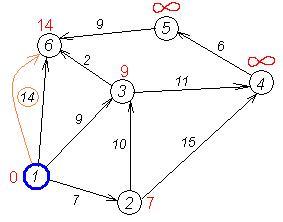
**Первый шаг.** Рассмотрим шаг алгоритма Дейкстры для нашего примера. Минимальную метку имеет вершина 1. Ее соседями являются вершины 2, 3 и 6.



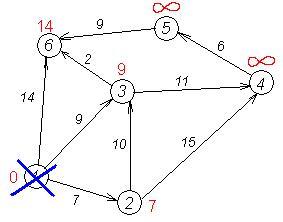
Первый по очереди сосед вершины 1 — вершина 2, потому что длина пути до нее минимальна. Длина пути в нее через вершину 1 равна кратчайшему расстоянию до вершины 1 + длина ребра, идущего из 1 в 2, то есть 0 + 7 = 7. Это меньше текущей метки вершины 2, поэтому новая метка 2-й вершины равна 7.



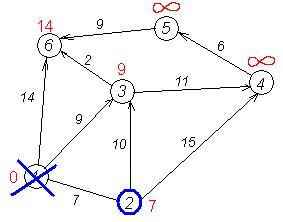
Аналогичную операцию проделываем с двумя другими соседями 1-й вершины — 3-й и 6-й.

Все соседи вершины 1 проверены. Текущее минимальное расстояние до вершины 1 считается окончательным и пересмотру не подлежит (то, что это действительно так, впервые доказал Дейкстра). Вычеркнем её из графа, чтобы отметить, что эта вершина посещена.



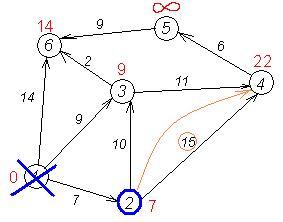
**Второй шаг.** Шаг алгоритма повторяется. Снова находим «ближайшую» из не посещённых вершин. Это вершина 2 с меткой 7.



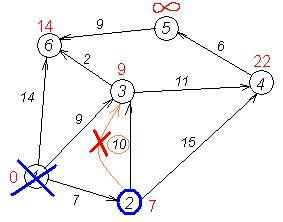
Снова пытаемся уменьшить метки соседей выбранной вершины, пытаясь пройти в них через 2-ю. Соседями вершины 2 являются 1, 3, 4.

Первый (по порядку) сосед вершины 2 — вершина 1. Но она уже посещена, поэтому с 1-й вершиной ничего не делаем.

Следующий сосед вершины 2 — вершины 4 и 3. Если идти в неё через 2-ю, то длина такого пути будет = кратчайшее расстояние до 2 + расстояние между вершинами 2 и 4 = 7 + 15 = 22. Поскольку 22<∞, устанавливаем метку вершины 4 равной 22.



Ещё один сосед вершины 2 — вершина 3. Если идти в неё через 2, то длина такого пути будет = 7 + 10 = 17. Но текущая метка третьей вершины равна 9<17, поэтому метка не меняется.



Все соседи вершины 2 просмотрены, замораживаем расстояние до неё и помечаем ее как посещенную.

**Третий шаг.** Повторяем шаг алгоритма, выбрав вершину 3.

**Дальнейшие шаги.** Повторяем шаг алгоритма для оставшихся вершин (Это будут по порядку 6, 4 и 5).

**Завершение выполнения алгоритма.** Алгоритм заканчивает работу, когда вычеркнуты все вершины. Результат его работы: кратчайший путь от вершины 1 до 2-й составляет 7, до 3-й — 9, до 4-й — 20, до 5-й — 20, до 6-й — 11.

Поскольку алгоритм Дейкстры всякий раз выбирает для обработки вершины с наименьшей оценкой кратчайшего пути, можно сказать, что он относится к жадным алгоритмам.

# 2. Требования к проекту

## 2.1 Требования к реализации алгоритма.

Проект является реализацией алгоритма Дейкстры на языке программирования java. Для его реализации были определены задачи, отвечающие за разные участки разработки.

* Создать wiki с описанием алгоритма
* Создать wiki с требованиями к проекту
* Создать GUI прототип проекта
* Создать класс основной структуры данных (граф)
* Отрисовка основной структуры данных (граф)
* Реализация алгоритма
* Считывание/генерация основной структуры данных

В процессе разработки проекта возникли следующие задачи:

* Разработать функционал под кнопки
* Визуализация поля отрисовки графа
* Разработать отрисовку и метод заполнения прогресс бара
* Реализация функций oneVertex и oneStep
* Написать Java-doc
* Отобразить веса рёбер
* Вывод работы алгоритма в специальное поле

## 2.2. Решение задач для реализации алгоритма.

|  |  |  |  |
| --- | --- | --- | --- |
| **Название задачи** | **Открыто** | **Закрыто** | **Кем решено** |
| Создать wiki с описанием алгоритма | 23.06 | 24.06 | Ахметшина |
| Создать wiki с требованиями к проекту | 23.06 | 24.06 | Ахметшина |
| Создать GUI прототип проекта | 23.06 | 24.06 | Ахметшина |
| Создать класс основной структуры данных (граф) | 23.06 | 24.06 | Ахметшина |
| Отрисовка основной структуры данных (граф) | 23.06 | 24.06 | Хафизов |
| Реализация алгоритма | 23.06 | 24.06 | Ахметшина |
| Считывание/генерация основной структуры данных | 23.06 | 24.06 | Ахметшина |
| Разработать функционал под кнопки | 28.06 | 01.07 | Хафизов |
| Визуализация поля отрисовки графа | 28.06 | 01.07 | Хафизов |
| Разработать отрисовку и метод заполнения прогресс бара | 28.06 | 01.07 | Хафизов |
| Реализация функций oneVertex и oneStep | 28.06 | 29.06 | Хафизов |
| Написать Java-doc | 02.07 | 03.07 | Ахметшина, Хафизов |
| Отобразить веса рёбер | 02.07 | 03.07 | Хафизов |
| Вывод работы алгоритма в специальном поле | 02.07 | 03.07 | Хафизов |

# 3. Входные и выходные данные

## 3.1. Входные данные.

Входные данные графа считываются из файла. Первой строкой считывается количество вершин, следующие строки – список смежности (первое число в тройке – вершина откуда, второе число – вершина куда, третье число – вес ребра).

*Ограничения на входные данные:*

Ребра не могут быть отрицательного веса. Также все числа должны быть неравны нулю.

*Пример:*

6

1 2 4

2 3 12

3 4 3

4 5 9

5 1 8

1 3 5

2 4 5

3 5 8

5 6 1

6 1 9

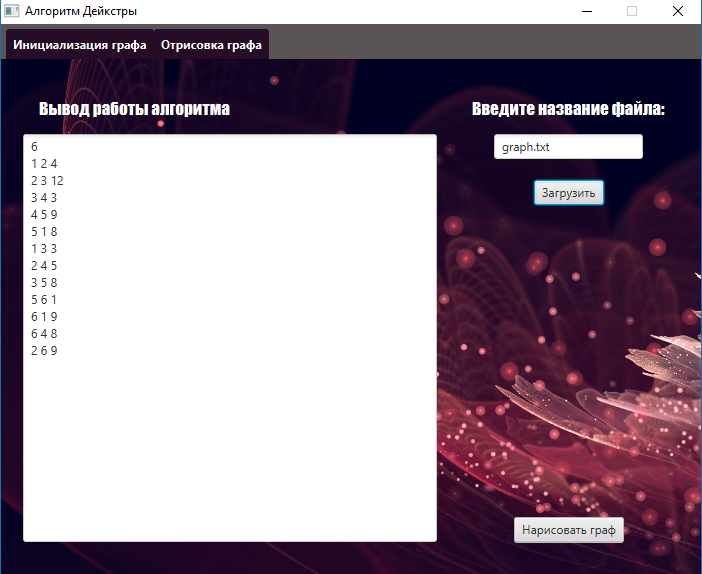
6 4 8

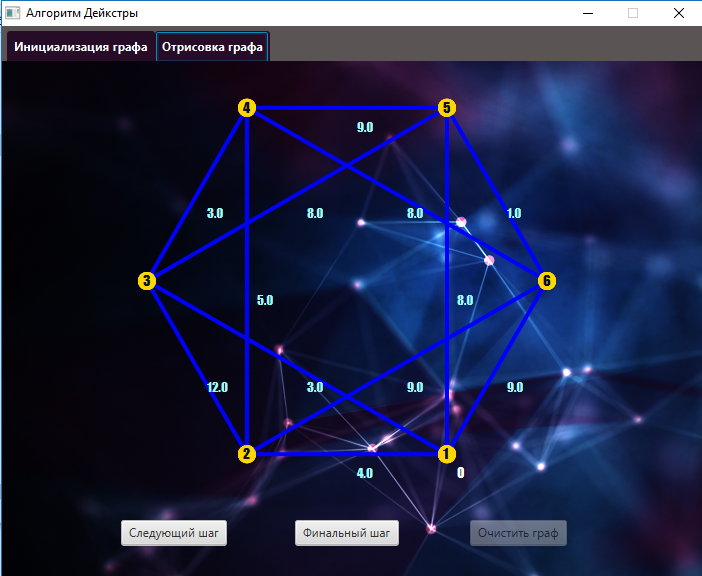
2 6 9

## 3.2. Выходные данные.

Выходные данные представлены графически (после нажатия кнопки «Нарисовать граф») и в текстовой панели (вкладка инициализация графа) в виде списка смежности.

Программа проверяет все рёбра и, если из одной вершины в другую несколько рёбер, выбирает минимальную из них.

**

**

# 4. Структуры данных и алгоритм работы программы

## 4.1. Структуры данных алгоритма.

|  |  |
| --- | --- |
| **Структура** | **Описание** |
| public int numV; | Количество вершин |
| public double matrix[][]; | Матрица смежности |
| public double dist[]; | Массив хранящий минимальные расстояния до вершин |
| public int parent[]; | Массив хранящий историю вершин |
| public boolean in\_tree[]; | Массив пройденных вершин |
| public static int cur = 0; | Вершина, с которой начинается работа алгоритма |

## 4.2. Функции алгоритма.

|  |  |
| --- | --- |
| Функция | Описание |
| public void init | Метод обрабатывающий входной вектор и преобразующий его в матрицу смежности. Также инициализирует все массивы в классе. |
| public void dijkstra | Метод выполняющий алгоритм с текущей точки(вершины) до конца |
| public void oneStep | Метод выполняющий один шаг алгоритма с текущего состояния |
| public void initVertex | Метод записывающий в первые numV ячеек ребра с минимальными путями |
| public boolean isEnd | Метод проверяющий закончился ли алгоритм на данный момент |
| public double getDist | Метод возвращает расстояние от начальной вершины до i-ой |
| public void clear | Метод, очищающий все данные в классе |

## 

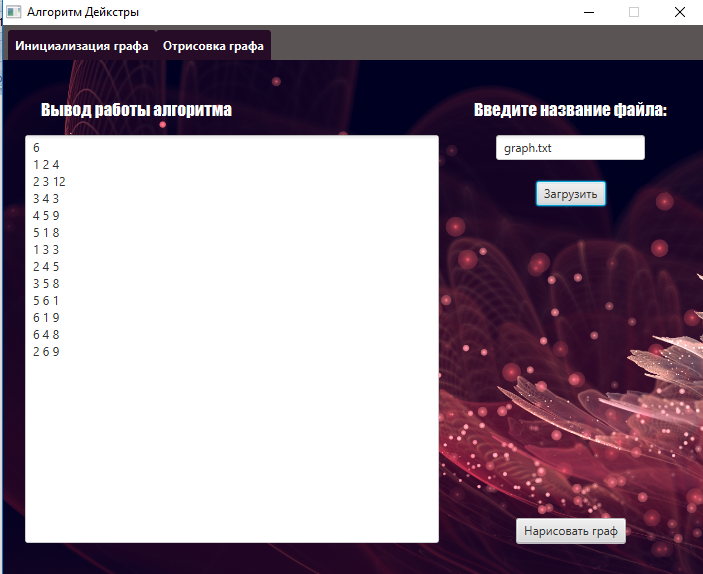
## 4.3. Алгоритм работы программы.

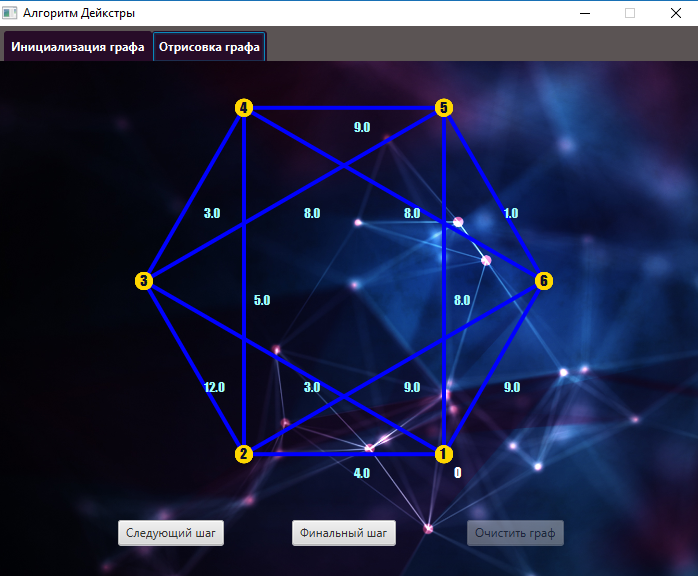
Для того чтобы начать работу с программой, требуется вписать название файла из которого необходимо считать граф. Далее нажать кнопку "Загрузить" и в поле "Вывод работы алгоритма" отобразятся входные данные. Затем нажать кнопку "Нарисовать граф" и программа автоматически переключится на другую вкладку. В этой вкладке будет нарисован граф, который был считан из файла. Далее есть несколько путей:

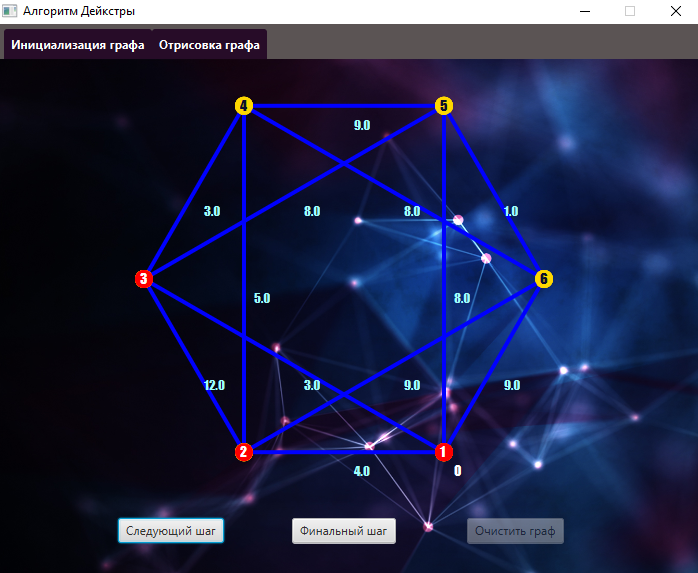
* рассматривать алгоритм пошагово
* сразу показать результат работы алгоритма

В первом случае, есть возможность нажимать кнопку "Следующий шаг" до тех пор, пока алгоритм не закончит работу, либо на каком-то шаге, если алгоритм не закончил работу, можно нажать кнопку "Финальный шаг" и алгоритм завершит работу. После завершения в поле "Вывод работы алгоритма" отобразятся списки смежности. Нажав кнопку "Очистить граф", можно ввести другой тестовый файл и заново просмотреть работу алгоритма.

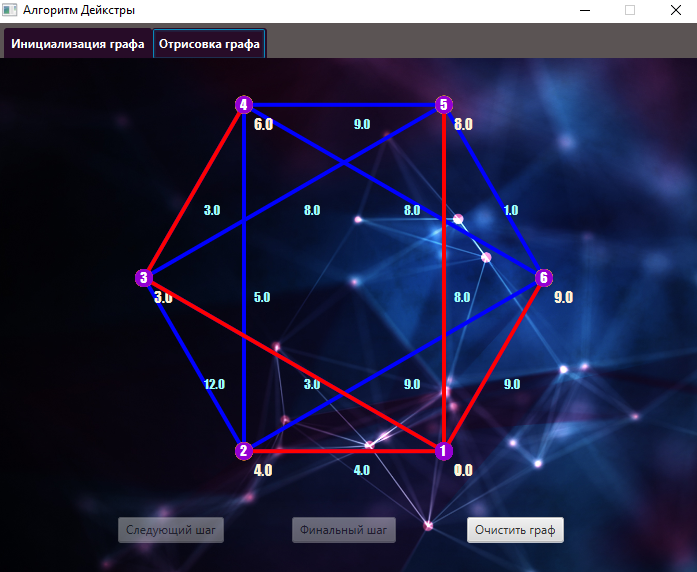
# 5.Тестирование программы



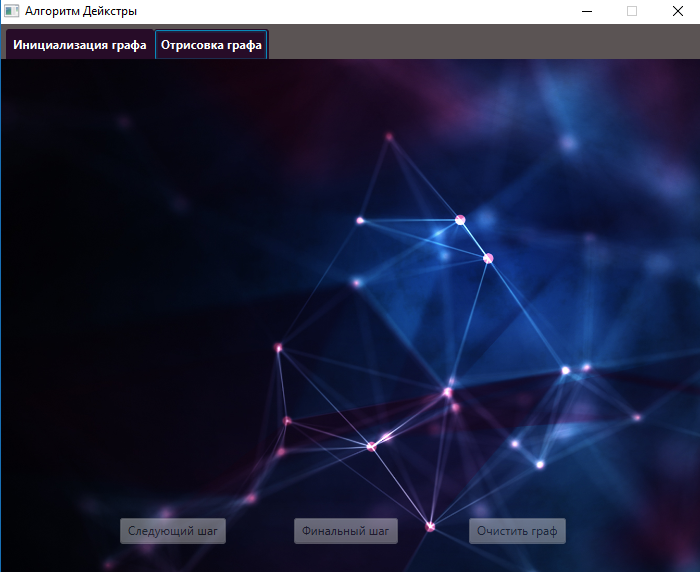


Пошаговая обработка:

После нажатия кнопки «Финальный шаг»:



Очистка графа:

****

# ЗАКЛЮЧЕНИЕ

В результате разработки данного проекта был изучен синтаксис языка java.

Был реализован и визуализирован алгоритм Дейкстры – нахождение минимальных путей от определенной вершины графа до всех остальных.

Изучена среда визуализации awt, swing и Javafx (использовалась последняя).

# Список использованных источников

1. Алгоритмы на Java (Автор: Роберт Седжвик, Кевин Уэйн)
2. <http://ru.stackoverflow.com/>
3. <https://ru.wikipedia.org/wiki/java/>
4. Java 8. Полное руководство. (Автор: Герберт Шилдт)
5. <https://habrahabr.ru/>
6. <https://youtube.com/> видеоуроки по Javafx

# Приложение: код программы

**main.java**

|  |
| --- |
| **package** graphics;  **import** javafx.application.Application; **import** javafx.fxml.FXMLLoader; **import** javafx.scene.Parent; **import** javafx.scene.Scene; **import** javafx.stage.Stage;  **public class** Main **extends** Application {   @Override  **public void** start(Stage primaryStage) **throws** Exception {  Parent root = FXMLLoader.*load*(getClass().getResource(**"sample.fxml"**));  primaryStage.setTitle(**"Алгоритм Дейкстры"**);  primaryStage.setScene(**new** Scene(root));  primaryStage.setResizable(**false**);  primaryStage.show();  }    **public static void** main(String[] args) {  *launch*(args);  } } |

**dijkstra.java**

|  |
| --- |
| **package** algoritm;   **import** graphics.Controller;  **import** java.util.\*;  */\*\*  \* Класс Dijkstra  \* В данном классе реализован алгоритм Дейкстры  \*  \** ***@author*** *Ахметшина Резеда  \*/* **public class** Dijkstra {  **public int numV**; *//количество вершин* **public double matrix**[][]; *//матрица смежности* **public double dist**[]; *//массив с растояниями* **public int parent**[]; *//массив родителей* **public static final double *INF*** = Integer.***MAX\_VALUE*** / 10; *//бесконечность* **public boolean in\_tree**[]; *//массив хранящий состояние вершин: обработано или нет* **public static int** *cur* = 0; *//вершина, с которой работаем, всегда начинаем с первой* **public int min\_dist**; *//минимальное расстояние на текущий момент. Необходимо для пошаговой реализации   /\*\*  \* Конструктор  \*  \** ***@param numV*** *колличество вершин  \** ***@param vertex*** *вектор с ребрами  \*/* **public** Dijkstra(**int** numV, Vector<Controller.GraphStruct> vertex) {  **this**.**numV** = numV;  **dist** = **new double**[numV];  **parent** = **new int**[numV];  **in\_tree** = **new boolean**[numV];  init(vertex);  }   */\*\*  \* Метод обрабатывающий входной вектор и преобразующий его в матрицу смежности  \* Так же инициализирует все массивы в классе  \*  \** ***@param vertex*** *\*/* **public void** init(Vector<Controller.GraphStruct> vertex) {  **matrix** = **new double**[**numV**][**numV**];  **for** (**int** i = 0; i < **numV**; i++) **in\_tree**[i] = **false**;  **for** (**int** i = 0; i < **numV**; i++) {  **for** (**int** j = 0; j < **numV**; j++) {  **if** (i == j) **matrix**[i][j] = 0;  **else matrix**[i][j] = ***INF***;  }  }  **for** (**int** i = 0; i < vertex.size(); i++) {  **if** (vertex.get(i).**weight** < **matrix**[vertex.get(i).**first** - 1][vertex.get(i).**second** - 1]) {  **matrix**[vertex.get(i).**first** - 1][vertex.get(i).**second** - 1] = vertex.get(i).**weight**;  **matrix**[vertex.get(i).**second** - 1][vertex.get(i).**first** - 1] = vertex.get(i).**weight**;  }  }  **for** (**int** i = 0; i < **numV**; i++)  **dist**[i] = ***INF***; *// машинная бесконечность,  // т. е. любое расстояние будет меньше данного* **dist**[0] = 0;  }   */\*\*  \* Метод выполняющий алгоритм с текущей точки до конца  \*  \** ***@param vertex*** *вектор с ребрами  \*/* **public void** dijkstra(Vector<Controller.GraphStruct> vertex) {  *// in\_tree[i] == true, если для вершины i  // уже посчитано минимальное расстояние  // пока есть необработанная вершина* **while** (!**in\_tree**[*cur*]) {  **in\_tree**[*cur*] = **true**;   **for** (**int** i = 0; i < **numV**; i++) {  *// если между cur и i есть ребро* **if** (**matrix**[*cur*][i] != ***INF***) {  *// считаем расстояние до вершины i:  // расстояние до cur + вес ребра* **double** d = **dist**[*cur*] + **matrix**[*cur*][i];  *// если оно меньше, чем уже записанное* **if** (d < **dist**[i]) {  **dist**[i] = d; *// обновляем его* **parent**[i] = *cur*; *// и "родителя"* }  }  }   *// ищем нерассмотренную вершину  // с минимальным расстоянием  //unsigned min\_dist(INT\_MAX);* **int** min\_dist = (**int**) ***INF***;  **for** (**int** i = 0; i < **numV**; i++) {  **if** (!**in\_tree**[i] && (**dist**[i] < min\_dist)) {  *cur* = i;  min\_dist = (**int**) **dist**[i];  }  }  }    *// Теперь:  // в dist[i] минимальное расстояние от start до i  // в parent[i] вершина, из которой лежит оптимальный путь в i* initVertex(vertex);  }   */\*\*  \* Метод выполняющий один шаг алгоритма с текущего состояния  \*  \** ***@param vertex*** *вектор с ребрами  \*/* **public void** oneStep(Vector<Controller.GraphStruct> vertex) {  **in\_tree**[*cur*] = **true**;   **for** (**int** i = 0; i < **numV**; i++) {  *// если между cur и i есть ребро* **if** (**matrix**[*cur*][i] != ***INF***) {  *// считаем расстояние до вершины i:  // расстояние до cur + вес ребра* **double** d = **dist**[*cur*] + **matrix**[*cur*][i];  *// если оно меньше, чем уже записанное* **if** (d < **dist**[i]) {  **dist**[i] = d; *// обновляем его* **parent**[i] = *cur*; *// и "родителя"* }  }  }   *// ищем нерассмотренную вершину  // с минимальным расстоянием  //unsigned min\_dist(INT\_MAX);* **min\_dist** = (**int**) ***INF***;  **for** (**int** i = 0; i < **numV**; i++) {  **if** (!**in\_tree**[i] && **dist**[i] < **min\_dist**) {  *cur* = i;  **min\_dist** = (**int**) **dist**[i];  }  }  *// Теперь:  // в dist[i] минимальное расстояние от start до i  // в parent[i] вершина, из которой лежит оптимальный путь в i* **if** (!**in\_tree**[*cur*]) initVertex(vertex);  }   */\*\*  \* Метод записывающий в первые numV ячеек ребра с минимальнымми путями  \*  \** ***@param vertex*** *вектор с ребрами  \*/* **public void** initVertex(Vector<Controller.GraphStruct> vertex) {  **for** (**int** counter = 0; counter < **numV**; counter++) {  vertex.get(counter).**first** = counter + 1;  vertex.get(counter).**second** = **parent**[counter] + 1;  vertex.get(counter).**weight** = **dist**[counter];  }  }   */\*\*  \* Метод проверяющий закончился ли алгоритм на данный момент  \*  \** ***@return*** *true если закончен, иначе false  \*/* **public boolean** isEnd() {  **return in\_tree**[*cur*];  }   */\*\*  \* Метод возвращает расстояние от начальной вершины до i-ой  \*  \** ***@param i*** *номер вершины  \** ***@return*** *расстояние до i-ой вершины  \*/* **public double** getDist(**int** i) {  **return dist**[i - 1];  }   */\*\*  \* Метод, очищающий все данные в классе  \*/* **public void** clear() {  **numV** = 0;  **matrix** = **null**;  **dist** = **null**;  **parent** = **null**;  **in\_tree** = **null**;  *cur* = 0;  } } |

**controller.java**

|  |
| --- |
| **package** graphics;   **import** algoritm.Dijkstra; **import** javafx.event.ActionEvent; **import** javafx.fxml.FXML; **import** javafx.fxml.Initializable; **import** javafx.scene.control.\*; **import** javafx.scene.layout.AnchorPane; **import** javafx.scene.layout.Pane; **import** javafx.scene.paint.Color; **import** javafx.scene.shape.Circle; **import** javafx.scene.shape.Line;  **import** java.io.DataInputStream; **import** java.io.FileInputStream; **import** java.io.IOException; **import** java.net.URL; **import** java.util.\*;  */\*\*  \* Класс Controller наследованный от Initializable  \* В данном классе находится GUI  \* Визуализация выполнена на Javafx  \*  \** ***@author*** *Хафизов Ильнур  \*/* **public class** Controller **implements** Initializable {  @FXML *//все переменные с пометками @FXML являются элементами GUI* Label **outNameLabel**;  @FXML  TextField **fileField**;  @FXML  TextArea **outputTextArea**;  @FXML  Button **startButton**;  @FXML  Button **loadButton**;  @FXML  Button **nextStepButton**;  @FXML  Pane **Pane1**;  @FXML  Circle **CircleGraph**;  @FXML  AnchorPane **AnchPane**;  @FXML  Button **clearButton**;  @FXML  TabPane **tabPane1** = **new** TabPane();  @FXML  Tab **tab1** = **new** Tab();  @FXML  Tab **tab2** = **new** Tab();  @FXML  Button **finalButton**;  **int**[] **inpInt** = **null**; *//массив для считывания из файла* **int**[] **readyArr** = **null**; *//массив для обработки данных* **int numV**; *//количество вершин* **int size** = 0; *//размер файла* GraphStruct **temp**; *//вспомогательная переменная* Dijkstra **algoritm**; *//переменная в которой будет обрабатываться алгоритм* **public static final double *INF*** = Integer.***MAX\_VALUE*** / 10;  **double x1**, **y1**, **x2**, **y2**, **l1**, **l2**, **m1**, **m2**; *//переменные для координат* **private** Vector<GraphStruct> **vertex**; *//массив рёбер   /\*\*  \* Класс GraphStruct  \* Хранит в себе все имеющиеся рёбра  \*/* **public class** GraphStruct {  **public int first**;  **public int second**;  **public double weight**;   **public** GraphStruct(**int** first, **int** second, **double** weight) {  **this**.**first** = first;  **this**.**second** = second;  **this**.**weight** = weight;  }  }   */\*\*  \* Метод реагирующий на нажатие кнопки "Финальный шаг"  \* В данном методе вызывается алгоритм Дейкстры, который проводится с данной точки и работает до конца(до завершения)  \* После выполнения алгоритма прорисовывается граф с минимальными путями (paint\_GRAPH\_RED)  \* Так же выводится список смежности в поле outputTextArea  \*  \** ***@param event*** *событие нажатия кнопки  \*/* @FXML  **public void** finalButtonClicked(ActionEvent event) {  **algoritm**.dijkstra(**vertex**);  paint\_GRAPH\_RED();  **clearButton**.setDisable(**false**);  **nextStepButton**.setDisable(**true**);  **finalButton**.setDisable(**true**);  printToArea();  }   */\*\*  \* Метод реагирующий на нажатие кнопки "Очистить граф"  \* В этом методе обнуляются все переменные, освобождается память, также очищается панель с рисовкой графа  \* Так же после нажатия программа переключается между вкладками  \*  \** ***@param event*** *событие нажатия кнопки  \*/* @FXML  **public void** clearButtonClicked(ActionEvent event) {  **algoritm**.clear();  **algoritm** = **null**;  **readyArr** = **null**;  **vertex** = **null**;  **numV** = 0;  **size** = 0;  **temp** = **null**;  **x1** = 0;  **x2** = 0;  **y1** = 0;  **y2** = 0;  **l1** = 0;  **l2** = 0;  **m1** = 0;  **m2** = 0;  **Pane1**.getChildren().clear();  **clearButton**.setDisable(**true**);  SingleSelectionModel<Tab> selectionModel = **tabPane1**.getSelectionModel();   **if** (selectionModel.getSelectedIndex() == 0) {  selectionModel.select(1);  } **else** {  selectionModel.select(0);  }  **outputTextArea**.setText(**""**);  }   */\*\*  \* Метод реагирующий на нажатие кнопки "Следующий шаг"  \* В этом методе запускается один шаг алгоритма Дейкстры  \* Также прорисовывается вершина которая была обработана  \* Если алгоритм заканчивается после нажатия кнопки, вырисовывается граф с минимальными путями(paint\_GRAPH\_RED)  \*  \** ***@param event*** *событие нажатия кнопки  \*/* @FXML  **public void** nextStepButtonClicked(ActionEvent event) {  **algoritm**.oneStep(**vertex**);  **if** (**algoritm**.isEnd()) {  paint\_GRAPH\_RED();  **clearButton**.setDisable(**false**);  **nextStepButton**.setDisable(**true**);  **finalButton**.setDisable(**true**);  printToArea();  } **else** paint\_Vertex();  }   */\*\*  \* Метод реагирующий на нажатие кнопки "Загрузить"  \* Вызывает метод fileOk() который выполняет считывание с файла  \* Название файла вводится в соответствующее поле  \*  \** ***@param event*** *событие нажатия кнопки  \*/* @FXML  **public void** loadButtonClicked(ActionEvent event) {  fileOk();  }   */\*\*  \* Метод реагирующий на нажатие кнопки "Нарисовать граф"  \* Метод выполняет обработку поступивших из файла данный и рисует начальный граф (paint\_GRAPH)  \* После нажания на кнопку выполняется переключения между вкладками программы  \*  \** ***@param event*** *событие нажатия кнопки  \*/* @FXML  **public void** startButtonClicked(ActionEvent event) {  **startButton**.setDisable(**true**);  init(**inpInt**);  paint\_GRAPH();  **nextStepButton**.setDisable(**false**);  **finalButton**.setDisable(**false**);  SingleSelectionModel<Tab> selectionModel = **tabPane1**.getSelectionModel();   **if** (selectionModel.getSelectedIndex() == 0) {  selectionModel.select(1);  } **else** {  selectionModel.select(0);  }  }   */\*\*  \* Метод рисующий граф введенный из файла  \* Также помечает начальную вершину рисую под ней вес  \*/* @FXML  **public void** paint\_GRAPH() {  Label V1, V2, V3;  Line edge;  *// Отрисовка графа* **for** (**int** i = 0; i < **vertex**.size(); i++) {  **x1** = **CircleGraph**.getRadius() \* Math.*cos*(2 \* Math.***PI*** / **numV** \* **vertex**.get(i).**first**) + **CircleGraph**.getCenterX();  **y1** = **CircleGraph**.getRadius() \* Math.*sin*(2 \* Math.***PI*** / **numV** \* **vertex**.get(i).**first**) + **CircleGraph**.getCenterY();  **x2** = **CircleGraph**.getRadius() \* Math.*cos*(2 \* Math.***PI*** / **numV** \* **vertex**.get(i).**second**) + **CircleGraph**.getCenterX();  **y2** = **CircleGraph**.getRadius() \* Math.*sin*(2 \* Math.***PI*** / **numV** \* **vertex**.get(i).**second**) + **CircleGraph**.getCenterY();  V3 = **new** Label(**""** + **vertex**.get(i).**weight**);  V3.setLayoutX((**x1** + **x2**) / 2 + 10);  V3.setLayoutY((**y1** + **y2**) / 2 + 10);  edge = **new** Line(**x1**, **y1**, **x2**, **y2**);  edge.setStrokeWidth(4);  edge.setStroke(Color.***BLUE***);  **Pane1**.getChildren().addAll(edge, **new** Circle(**x1**, **y1**, 9, Color.***GOLD***), **new** Circle(**x2**, **y2**, 9, Color.***GOLD***));  **x1** += -4;  **y1** += -9;  **x2** += -4;  **y2** += -9; *//рисование имен вершин* V1 = **new** Label(**""** + **vertex**.get(i).**first**);  V1.setLayoutX(**x1**);  V1.setLayoutY(**y1**);  V2 = **new** Label(**""** + **vertex**.get(i).**second**);  V2.setLayoutX(**x2**);  V2.setLayoutY(**y2**);  V1.setStyle(**"-fx-font-weight: bold; -fx-font-size: 14px; -fx-text-fill: rgb(12,16,45); -fx-font-family: \"Impact\";"**);  V2.setStyle(**"-fx-font-weight: bold; -fx-font-size: 14px; -fx-text-fill: rgb(12,16,45); -fx-font-family: \"Impact\";"**);  V3.setStyle(**"-fx-font-weight: bold; -fx-font-size: 13px; -fx-text-fill: rgb(157,255,254); -fx-font-family: \"Impact\";"**);  **Pane1**.getChildren().addAll(V1, V2, V3);  }  **x1** = **CircleGraph**.getRadius() \* Math.*cos*(2 \* Math.***PI*** / **numV** \* **vertex**.get(0).**first**) + **CircleGraph**.getCenterX();*//отрисовка 1ой вершины* **y1** = **CircleGraph**.getRadius() \* Math.*sin*(2 \* Math.***PI*** / **numV** \* **vertex**.get(0).**first**) + **CircleGraph**.getCenterY();  V2 = **new** Label(**"0"**);  **x1** += 10;  **y1** += 10;  V2.setLayoutX(**x1**);  V2.setLayoutY(**y1**);  V2.setStyle(**"-fx-font-weight: bold; -fx-font-size: 14px; -fx-text-fill: rgb(255,255,255); -fx-font-family: \"Impact\";"**);  **Pane1**.getChildren().addAll(V2);  }   */\*\*  \* Метод рисующий минимальные пути на графах(красными линиями)  \* Также под каждой вершиной рисуется расстояние от начальной вершины  \*/* @FXML  **private void** paint\_GRAPH\_RED() {  Label V1, V2, W;  Line edge;  **for** (**int** i = 0; i < **numV**; i++) {  **x1** = **CircleGraph**.getRadius() \* Math.*cos*(2 \* Math.***PI*** / **numV** \* **vertex**.get(i).**first**) + **CircleGraph**.getCenterX();  **y1** = **CircleGraph**.getRadius() \* Math.*sin*(2 \* Math.***PI*** / **numV** \* **vertex**.get(i).**first**) + **CircleGraph**.getCenterY();  **x2** = **CircleGraph**.getRadius() \* Math.*cos*(2 \* Math.***PI*** / **numV** \* **vertex**.get(i).**second**) + **CircleGraph**.getCenterX();  **y2** = **CircleGraph**.getRadius() \* Math.*sin*(2 \* Math.***PI*** / **numV** \* **vertex**.get(i).**second**) + **CircleGraph**.getCenterY();  **if** (**algoritm**.getDist(**vertex**.get(i).**first**) != ***INF***) {  **l1** = **x1** + 10;  **l2** = **y1** + 10;  W = **new** Label(**algoritm**.getDist(**vertex**.get(i).**first**) + **""**);  W.setLayoutX(**l1**);  W.setLayoutY(**l2**);  W.setStyle(**"-fx-font-weight: bold; -fx-font-size: 15px; -fx-text-fill: rgb(255,242,212); -fx-font-family: \"Impact\";"**);  **Pane1**.getChildren().addAll(**new** Circle(**x1**, **y1**, 9, Color.***GOLD***), W);  } **else if** (**algoritm**.getDist(**vertex**.get(i).**second**) != ***INF***) {  **m1** = **x2** + 10;  **m2** = **y2** + 10;  W = **new** Label(**algoritm**.getDist(**vertex**.get(i).**second**) + **""**);  W.setLayoutX(**m1**);  W.setLayoutY(**m2**);  W.setStyle(**"-fx-font-weight: bold; -fx-font-size: 15px; -fx-text-fill: rgb(255,242,212); -fx-font-family: \"Impact\";"**);  **Pane1**.getChildren().addAll(W);  }  **if** ((**algoritm**.getDist(**vertex**.get(i).**second**) != ***INF***) && (**algoritm**.getDist(**vertex**.get(i).**first**) != ***INF***)) {  edge = **new** Line(**x1**, **y1**, **x2**, **y2**);  edge.setStrokeWidth(4);  edge.setStroke(Color.***RED***);  **Pane1**.getChildren().addAll(edge, **new** Circle(**x2**, **y2**, 9, Color.***DARKVIOLET***), **new** Circle(**x1**, **y1**, 9, Color.***DARKVIOLET***));  **x1** += -4;  **y1** += -9;  **x2** += -4;  **y2** += -9; *//рисование имен вершин* V1 = **new** Label(**""** + **vertex**.get(i).**first**);  V1.setLayoutX(**x1**);  V1.setLayoutY(**y1**);  V2 = **new** Label(**""** + **vertex**.get(i).**second**);  V2.setLayoutX(**x2**);  V2.setLayoutY(**y2**);  V1.setStyle(**"-fx-font-weight: bold; -fx-font-size: 14px; -fx-text-fill: white; -fx-font-family: \"Impact\";"**);  V2.setStyle(**"-fx-font-weight: bold; -fx-font-size: 14px; -fx-text-fill: white; -fx-font-family: \"Impact\";"**);  **Pane1**.getChildren().addAll(V1, V2);  }  }  }   */\*\*  \* Метод рисует вершину, которая была обработана за последний шаг  \*/* @FXML  **private void** paint\_Vertex() {  Label V1;  **for** (**int** i = 0; i < **numV**; i++) {  **if** (**algoritm**.**in\_tree**[i]) {  **x1** = **CircleGraph**.getRadius() \* Math.*cos*(2 \* Math.***PI*** / **numV** \* (i + 1)) + **CircleGraph**.getCenterX();  **y1** = **CircleGraph**.getRadius() \* Math.*sin*(2 \* Math.***PI*** / **numV** \* (i + 1)) + **CircleGraph**.getCenterY();  **Pane1**.getChildren().addAll(**new** Circle(**x1**, **y1**, 9, Color.***RED***));  **x1** += -4;  **y1** += -9;  i += 1;  V1 = **new** Label(**""** + i);  i--;  V1.setLayoutX(**x1**);  V1.setLayoutY(**y1**);  V1.setStyle(**"-fx-font-weight: bold; -fx-font-size: 14px; -fx-text-fill: white; -fx-font-family: \"Impact\";"**);  **Pane1**.getChildren().addAll(V1);  }  }  }   */\*\*  \* Метод обрабатывающий входные данные из файла  \*  \** ***@param inpInt*** *массив, который был считан из файла  \*/* **public void** init(**int**[] inpInt) {  **vertex** = **new** Vector<GraphStruct>();  **int** j;  **for** (j = 0; j < inpInt.**length**; j++) **if** (inpInt[j] == 0) **break**;  **readyArr** = **new int**[j];  **for** (**int** i = 0; i < j; i++) **readyArr**[i] = inpInt[i];  **numV** = **readyArr**[0];  **for** (**int** i = 1; i < **readyArr**.**length**; i += 3) {  **temp** = **new** GraphStruct(**readyArr**[i], **readyArr**[i + 1], **readyArr**[i + 2]);  **vertex**.add(**temp**);  }  **for** (**int** i = 0; i < (**vertex**.size() - 1); i++) {  **if** ((**vertex**.get(i + 1).**second** == **vertex**.get(i).**first**) && (**vertex**.get(i).**second** == **vertex**.get(i + 1).**first**))  **if** (**vertex**.get(i).**weight** > **vertex**.get(i + 1).**weight**) {  **vertex**.remove(i);  i--;  } **else** {  **vertex**.remove(i + 1);  i--;  }  }  **readyArr** = **null**;  **algoritm** = **new** Dijkstra(**numV**, **vertex**);  }   */\*\*  \* Метод, выводяший списки смежности в поле, находящейся в первой вкладке программы  \*/* **public void** printToArea() {  String buff = **new** String();  ArrayList[] adj = **new** ArrayList[**numV**];  **for** (**int** i = 0; i < **numV**; ++i) {  adj[i] = **new** ArrayList<Integer>();  }  **for** (**int** i = 0; i < **numV**; i++) {  adj[**vertex**.get(i).**first** - 1].add(**vertex**.get(i).**second**);  adj[**vertex**.get(i).**second** - 1].add(**vertex**.get(i).**first**);  }  **for** (**int** i = 0; i < **numV**; i++) {  **for** (**int** j = 0; j < adj[i].size(); j++) {  **int** n = i + 1;  Integer a = (Integer) adj[i].get(j);  **if** (n == a) {  adj[i].remove(j);  }  }  }  **for** (**int** i = 0; i < **numV**; i++) {  buff = buff + (i + 1) + **": "**;  **for** (**int** j = 0; j < adj[i].size(); j++) {  **int** n = i + 1;  Integer a = (Integer) adj[i].get(j);  **if** (n != a) {  buff = buff + **" -> "** + adj[i].get(j);  }  }  buff += **"\n"**;  }  **outputTextArea**.setText(buff);  }   */\*\*  \* Метод, считывающий информацию из файла  \*/* @FXML  **public void** fileOk() {  **int** i = 0;  String buff, buff2 = **new** String();  DataInputStream input = **null**;  **try** (DataInputStream streambuff = **new** DataInputStream(**new** FileInputStream(**fileField**.getText()))) {  **size** = streambuff.available();  } **catch** (IOException ex2) {  **outputTextArea**.setText(**"Wrong path"**);  **outputTextArea**.setVisible(**true**);  **return**;  }  **inpInt** = **new int**[**size**];  **try** {  input = **new** DataInputStream(**new** FileInputStream(**fileField**.getText()));   **while** ((buff = input.readLine()) != **null**) {  StringTokenizer stoken = **new** StringTokenizer(buff);  **while** (stoken.hasMoreTokens()) {  **int** ibuff = Integer.*parseInt*(stoken.nextToken());  **inpInt**[i] = ibuff;  buff2 += Integer.*toString*(ibuff) + **" "**;  **if** (i % 3 == 0) buff2 += **"\n"**;  i++;  }  }  } **catch** (IOException ex) {   }   **outputTextArea**.setText(buff2);  **outputTextArea**.setVisible(**true**);  **startButton**.setDisable(**false**);  }   */\*\*  \* Инициализация программы перед запуском  \*  \** ***@param location*** *\** ***@param resources*** *\*/* @Override  **public void** initialize(URL location, ResourceBundle resources) {  **startButton**.setDisable(**true**);  **finalButton**.setDisable(**true**);  **fileField**.setText(**"graph.txt"**);  **nextStepButton**.setDisable(**true**);  **clearButton**.setDisable(**true**);  } } |

**sample.fxml**

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>  <?***import javafx.scene.shape.\****?> <?***import javafx.scene.text.\****?> <?***import javafx.scene.image.\****?> <?***import javafx.scene.control.\****?> <?***import javafx.scene.layout.\****?>* <**AnchorPane minHeight="-Infinity" minWidth="-Infinity" prefHeight="540.0" prefWidth="690.0" stylesheets="@../gui.css"  xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="graphics.Controller"**>  <**children**>  <**TabPane fx:id="tabPane1" layoutX="88.0" layoutY="47.0" prefHeight="540.0" prefWidth="700.0"  tabClosingPolicy="UNAVAILABLE" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"  AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0"**>  <**tabs**>  <**Tab fx:id="tab1" text="Инициализация графа"**>  <**content**>  <**AnchorPane**>  <**children**>  <**AnchorPane**>  <**children**>  <**ImageView fitHeight="549.0" fitWidth="1078.0" pickOnBounds="true"  preserveRatio="true"**>  <**image**>  <**Image url="@../14532028780463.png"**/>  </**image**>  </**ImageView**>  <**TextArea fx:id="outputTextArea" layoutX="22.0" layoutY="75.0"  prefHeight="408.0" prefWidth="414.0"**/>  <**Label fx:id="outNameLabel" layoutX="38.0" layoutY="38.0"  text="Вывод работы алгоритма" textFill="WHITE"**>  <**font**>  <**Font name="Impact" size="17.0"**/>  </**font**>  </**Label**>  <**Button fx:id="startButton" layoutX="513.0" layoutY="458.0"  mnemonicParsing="false" onAction="#startButtonClicked"  text="Нарисовать граф"**/>  <**TextField fx:id="fileField" layoutX="493.0" layoutY="75.0"**/>  <**Label fx:id="nameInputLabel" layoutX="471.0" layoutY="38.0"  text="Введите название файла:" textFill="WHITE"**>  <**font**>  <**Font name="Impact" size="17.0"**/>  </**font**>  </**Label**>  <**Button fx:id="loadButton" layoutX="533.0" layoutY="121.0"  mnemonicParsing="false" onAction="#loadButtonClicked" text="Загрузить"**/>  </**children**>  </**AnchorPane**>  </**children**>  </**AnchorPane**>  </**content**>  </**Tab**>  <**Tab fx:id="tab2" text="Отрисовка графа"**>  <**content**>  <**AnchorPane fx:id="AnchPane" minHeight="0.0" minWidth="0.0" prefHeight="180.0"  prefWidth="200.0"**>  <**children**>  <**ImageView fitHeight="657.0" fitWidth="1000.0" pickOnBounds="true" preserveRatio="true"**>  <**image**>  <**Image url="@../14532026843321.png"**/>  </**image**>  </**ImageView**>  <**Button fx:id="nextStepButton" layoutX="119.0" layoutY="459.0" mnemonicParsing="false"  onAction="#nextStepButtonClicked" text="Следующий шаг"**/>  <**Pane fx:id="Pane1" prefHeight="400.0" prefWidth="690.0"**>  <**children**>  <**Circle fx:id="CircleGraph" centerX="345.0" centerY="220.0" fill="DODGERBLUE"  layoutX="345.0" layoutY="203.0" radius="200.0" stroke="BLACK"  strokeType="INSIDE" visible="false"**/>  </**children**>  </**Pane**>  <**Button fx:id="clearButton" layoutX="468.0" layoutY="459.0" mnemonicParsing="false"  onAction="#clearButtonClicked" text="Очистить граф"**/>  <**Button fx:id="finalButton" layoutX="293.0" layoutY="459.0" mnemonicParsing="false"  onAction="#finalButtonClicked" text="Финальный шаг"**/>  </**children**>  </**AnchorPane**>  </**content**>  </**Tab**>  </**tabs**>  </**TabPane**>  </**children**> </**AnchorPane**> |