**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

### Отчет

### по учебной практике

**«Визуализация алгоритмов на графах на Java»**

Студент гр. 6383 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Быков И. В.

Студент гр. 6383 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лавренкова Е. Л.

Студент гр. 6381 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кухарев М. А.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фирсов М.А.

Санкт-Петербург

2018

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студент Быков И. В. группы 6383 | | |
| Студентка Лавренкова Е. Л. группы 6383 | | |
| Студент Кухарев М.А. группы 6381  Тема практики: Визуализация алгоритмов на графах на Java | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.  Алгоритм: Форда – Беллмана. | | |
| Сроки прохождения практики: 27.06.2018 – 10.07.2017 | | |
| Дата сдачи отчета: ??.07.2018 | | |
| Дата защиты отчета: ??.07.2018 | | |
|  | | |
| Студент |  | Быков И. В. |
| Студентка |  | Лавренкова Е. Л. |
| Студент |  | Кухарев М. А. |
| Руководитель |  | Фирсов М.А. |

**АННОТАЦИЯ**

В данной работе разрабатывается визуализатор алгоритма Форда — Беллмана на языке Java с использованием графического интерфейса. Данный алгоритм производит поиск кратчайшего пути во взвешенном графе с возможными ребрами отрицательного веса. В результате выполнения работы программы, пользователь получает граф, на котором отображен кратчайший путь от одной вершины к другой.

**annotation**

In this work, there is the Bellman — Ford algorithm developed in the Java language using a graphical interface. This algorithm searches for the shortest path in a weighted graph with possible edges of negative weight. As a result of the execution of the program, the user receives a graph on which the shortest path from one vertex to another is displayed.

**Содержание**

[1. ИСХОДНОЕ ЗАДАНИЕ 5](#_Toc486917569)

[2. ОБЪЯСНЕНИЕ АЛГОРИТМА 5](#_Toc486917570)

[2.1. Теоретическая справка 5](#_Toc486917571)

[2.2.Описание алгоритма 5](#_Toc486917572)

[3. ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ 6](#_Toc486917573)

[4. СПЕЦИФИКАЦИЯ 6](#_Toc486917574)

[4.1. Исходные Требования к программе 6](#_Toc486917575)

[4.2. Графический интерфейс 7](#_Toc486917576)

[4.3. Ход работы 9](#_Toc486917577)

[5. План разработки и распределение ролей в бригаде](#_Toc486917578) 11

[5.1. План разработки](#_Toc486917579) 11

[5.2. Распределение ролей в бригаде](#_Toc486917580) 11

[6. ТЕСТИРОВАНИЕ ПРОГРАММЫ](#_Toc486917581) 12

[6.1. Тестирование работы программы при вводе из файла in.txt:](#_Toc486917582) 12

[6.2. Тестирование работы программы с помощью случайной генерации.](#_Toc486917583) 12

[7. ЗАКЛЮЧЕНИЕ](#_Toc486917584) 12

[8. ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД](#_Toc486917585) 12

# 1. **ИСХОДНОЕ ЗАДАНИЕ**

В данной практической работе требуется написать программу на языке JAVA, выполняющую поиск Кратчайших путей в графе с помощью алгоритма Беллмана — Форда и выполнить визуализацию алгоритма.

# 2. **ОБЪЯСНЕНИЕ АЛГОРИТМА**

## 2.1. **Теоретическая справка**

Алгоритм Беллмана — Форда — алгоритм поиска кратчайшего пути во взвешенном графе. За время O (|V| × |E|) алгоритм находит кратчайшие пути от одной вершины графа до всех остальных. В отличие от алгоритма Дейкстры, алгоритм Беллмана-Форда допускает рёбра с отрицательным весом. Предложен независимо Ричардом Беллманом и Лестером Фордом.

## 2.2. **Описание алгоритма**

Мы считаем, что граф не содержит цикла отрицательного веса. Заведём массив расстояний , который после отработки алгоритма будет содержать ответ на задачу. В начале работы мы заполняем его следующим образом:  , а все остальные элементы   равны бесконечности ∞.

Сам алгоритм Форда-Беллмана представляет из себя несколько фаз. На каждой фазе просматриваются все рёбра графа, и алгоритм пытается произвести **релаксацию** (relax, ослабление) вдоль каждого ребра  стоимости c. Релаксация вдоль ребра — это попытка улучшить значение   значением . Фактически это значит, что мы пытаемся улучшить ответ для вершины , пользуясь ребром  и текущим ответом для вершины .

Утверждается, что достаточно  фазы алгоритма, чтобы корректно посчитать длины всех кратчайших путей в графе (повторимся, мы считаем, что циклы отрицательного веса отсутствуют). Для недостижимых вершин расстояние  останется равным бесконечности ∞.

Алгоритм Беллмана-Форда позволяет очень просто определить, существует ли в графе отрицательный цикл, достижимый из вершины. Достаточно произвести внешнюю итерацию цикла не {\displaystyle |V|-1}, a ровно {\displaystyle |V|} раз. Если при исполнении последней итерации длина кратчайшего пути до какой-либо вершины строго уменьшилась, то в графе есть отрицательный цикл, достижимый из данной вершины. На основе этого можно предложить следующую оптимизацию: отслеживать изменения в графе и, как только они закончатся, сделать выход из цикла (дальнейшие итерации будут бессмысленны).

# 3. **ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ**

Задан исходный ориентированный граф G = (V, E), где V – количество вершин в графе, для данного графа, в котором каждая вершина пронумерована от 0 до V. Необходимо реализовать алгоритм Беллмана — Форда для данного графа.

Визуализацию необходимо проделать для небольших по размерам графов для того, чтобы убедится в правильности визуализации алгоритма. После выполнения алгоритма будет получен минимальный путь, такой, чтобы пройти из стартовой точки в конечную с минимальными затратами по стоимости.

# 4. **СПЕЦИФИКАЦИЯ**

## 4.1. **Исходные Требования к программе**

#### **4.1.1.Требования к вводу исходных данных.**

Программа предоставляет пользователю графический интерфейс. Входные данные считываются либо из файла, либо с помощью случайной генерации графа (критериями построения графа будут являться количество вершин и количество ребер, исходящих из одной вершины). Тестовый файл содержит количество вершин, количество ребер, вершины из которых исходят ребра, вершины в которые входят ребра и веса ребер графа. После запуска программы, пользователь наблюдает меню программы, в котором он может считать граф из файла или сгенерировать граф самостоятельно.

#### **4.1.2.Требования к визуализации.**

Визуализация должна представлять собой окно с меню. Граф можно либо считать из файла in.txt, при нажатии кнопки «Read from the file», либо сгенерировать, при нажатии «Generate graph».

Эскиз окна с меню выглядит следующим образом (рис.1, 2):

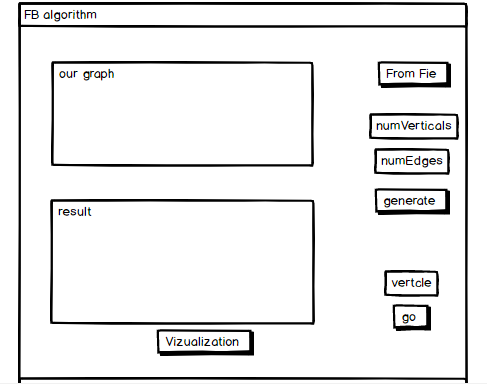


Рисунок 1. Эскиз главного окна.

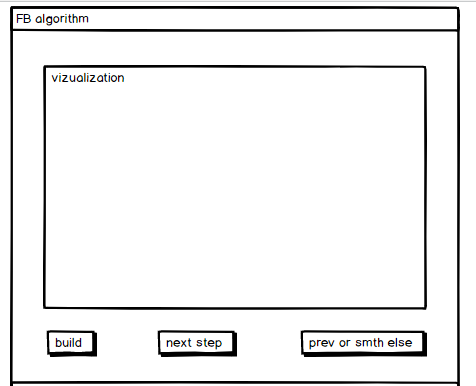


Рисунок 2. Эскиз окна с графом.

## 4.2. **Графический интерфейс**

При выборе генерации графа, пользователя просят ввести количество вершин в графе (до 30) и количество ребер, исходящих из одной вершины (не более 10). При создании графа, считанного из текстового файла, расположение вершин задаются автоматически, ребра – создаются между i-ой вершиной и j-ой. Веса рёбер указываются пользователем, непосредственно в самом текстовом файле. При создании сгенерированного графа расположение вершин, ребер и веса ребер задаются автоматически.

Пользователю, после построения графа, предоставится выбор вершины. Ему необходимо ввести вершину графа в текстовое поле и нажать кнопку «Start the algorithm».

Затем к созданному графу применяется алгоритм Беллмана — Форда. Пользователь сможет просмотреть работу алгоритма в текстовом поле и запустить визуализацию работы алгоритма, нажав «Visualization of the algorithm’s work». Далее пользователю предлагается ввести конечную вершину графа с помощью текстового поля и кнопки «Path».

Пользователь может сделать выбор, в каком виде получить результат выполнения алгоритма: в виде конечного результата или пошагового выполнения.

Выходными данными является визуализированный алгоритм: построенный кратчайший путь между начальной и конечной вершинами.

Кратчайший путь в итоговом графе выделяется красным цветом. Ребро графа, которое в ходе выполнения подверглось релаксации, выделяется зелёным цветом, а ребро, которое не подверглось релаксации – синим. Если до точки, в которую входит ребро, нет пути, то оно выделяется оранжевым цветом. Программа реализована на языке Java с использованием Фреймворка JavaFX. Далее представлен план Меню (рис. 3, 4):

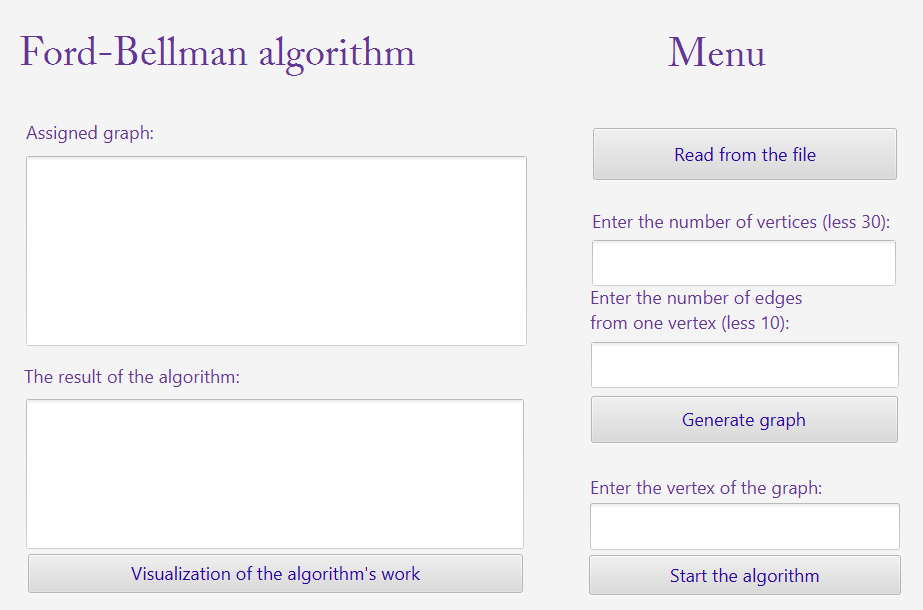


Рисунок 3. Стартовое окно меню.

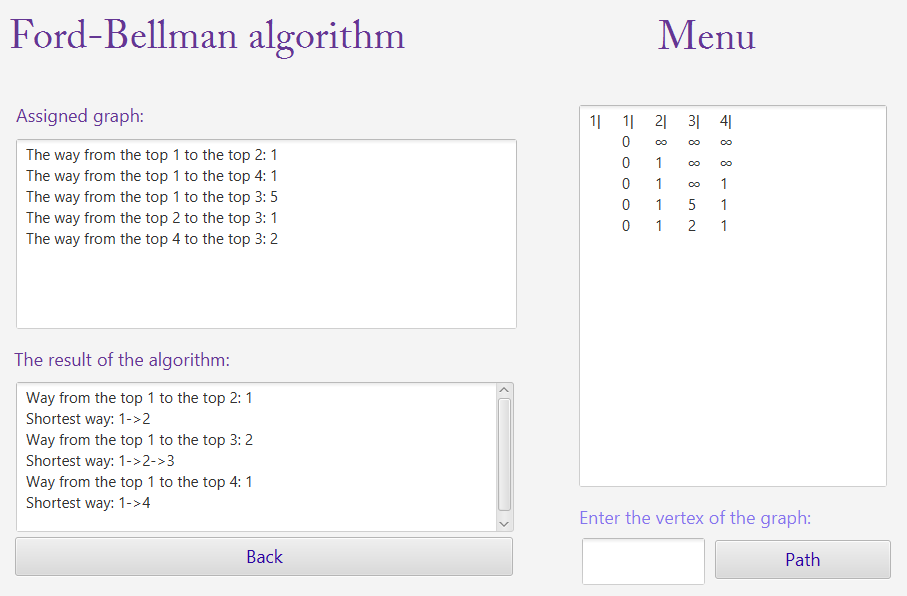


Рисунок 4. Стартовое окно меню.

Далее представлен план дочернего окна (рис. 5).



Рисунок 5. Дочернее окно меню с графом.

4.3.Ход работы

Ход работы над разработкой программы состоит из четырёх частей:   
*1) Реализация графа.*

Граф определяется классом Graph, который состоит из класса ElementGraphWay и класса WaysPoint.   
Класс ElementGraphWay содержит поля: вершина-исток, вершина-финиш и вес ребра. Класс WaysPoint содержит координаты и имена вершин. Таким образом, граф представляется в виде списка смежности.  
*2) Реализация алгоритма Беллмана — Форда.*

Для реализации алгоритма Беллмана — Форда используется функция searchAlgorithm, которая производит поиск кратчайшего пути из заданной вершины в графе. А с помощью функции outputWays производится вывод кратчайшего пути из заданной вершины в графе.

*3) Построение графа.*   
Построение будет состоять из двух функций. Прямое построение графа, заданного с помощью списка смежности пользователем в файле, и генерация случайного графа с указанным пользователем количеством вершин.

Построение графа пользователем из файла осуществляется функцией inputFile. В случае нехватки данных, файла, не содержащего данных или отсутствия файла, будут брошены соответствующие исключения.

Построение случайно сгенерированного графа производится функцией inputGeneration(). Пользователю нужно будет ввести в поля меню количество вершин и количество исходящих из каждой вершины ребер. Весами ребер будут случайно сгенерированные числа.  
*4) Визуализация.*

Визуализация разработана с помощью JavaFX - платформы на основе Java для создания приложений с насыщенным графическим интерфейсом.   
Файл "sample.fxml" содержит визуализацию главного меню. Файл "GUI\_prototype.fxml" содержит визуализацию дочернего окна для отрисовки итогового графа.

Вывод на экран конечного результата и пошаговое выполнение алгоритма содержится в файле GraphController.java

# 5. **План разработки и распределение ролей в бригаде**

## 5.1. **План разработки**

1) 29.06.2018 – предоставление UML-диаграммы классов программы, демонстрация демо-версии визуального интерфейса алгоритма.

2) 03.07.2018 – реализация алгоритма Беллмана – Форда, демонстрация пользовательского интерфейса.

3) 05.07.2018 –

## 5.2. **Распределение ролей в бригаде**

В бригаде три человека, между которыми распределена работа над проектом.

|  |  |
| --- | --- |
| Кухарев М.А. | Разработка графического интерфейса на JavaFX. |
| Быков И. В. | Реализация алгоритма Беллмана — Форда. |
| Лавренкова Е.Л. | Тестирование программы, составление отчёта. |

**Использованные структуры данных:**

Граф представлен списком двух вершин и весом ребра между ними.

*Класс для хранения ребер графа:*

**public class** ElementGraphWay {  
 **int from**; //начальная вершина  
 **int to**; //конечная вершина  
 **int l**; //вес ребра  
}

*Класс для хранения координат узлов графа:***public class** WaysPoint {  
 **int x**; //х-координата вершины  
 **int y**; //у-координата вершины  
 **int name**;//имя вершины  
}

*UML-диаграмма классов (рис. 6):*

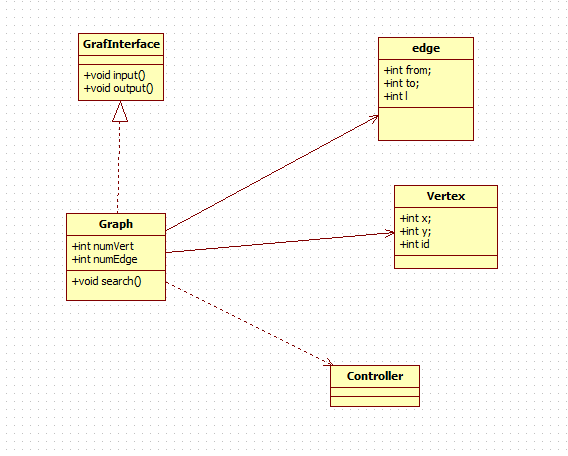
**

Рисунок 6. UML-диаграмма.

# 6. **ТЕСТИРОВАНИЕ ПРОГРАММЫ**

## **6.1. Тестирование работы программы при вводе из файла in.txt**:

## 6.2. **Тестирование работы программы с помощью случайной генерации.**

# 7. **ЗАКЛЮЧЕНИЕ**

По завершению учебной практики была написана программа, выполняющая поиск кратчайших путей в графе с помощью алгоритма Беллмана — Форда для неотрицательных графов, а также была выполнена визуализация алгоритма.

Программа реализована на языке Java в интегрированной среде разработки IntelliJ IDEA, с использованием Фреймворка JavaFX.

# 8. ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

***1) Файл Main.java.***

package sample;  
  
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("Ford-Bellman algorithm");  
 primaryStage.setScene(new Scene(root, 1400, 700));  
 primaryStage.show();  
 }  
  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
}

***2) Файл Controller.java.***

package sample;  
  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
  
  
*/\*\*  
 \* A class for handling events of the elements of the main window.  
 \* \*/*public class Controller {  
 private Alert alert;  
 @FXML  
 private TextField textField1;  
 @FXML  
 private TextArea amountVertex;  
 @FXML  
 private TextArea amountEdges;  
 @FXML  
 public TextArea result;  
 @FXML  
 private TextArea textGraph;  
 @FXML  
 private Label text3;  
 @FXML  
 private Label text4;  
 @FXML  
 private Label text5;  
 @FXML  
 private Label text6;  
 @FXML  
 private Button next;  
 @FXML  
 private Button fileGraph;  
 @FXML  
 private Button generateGraph;  
 @FXML  
 private Button begAlg;  
 @FXML  
 private Button back;  
 @FXML  
 private Button graphWay;  
 @FXML  
 private Label vertex;  
 @FXML  
 public TextArea graphAlg;  
 @FXML  
 private TextArea graphV;  
  
  
 private static int *k*;  
  
  
 public static Graph *P* = new Graph();  
 private GraphController graphWindow;  
 public static boolean *Setting* = true;  
  
 */\*\*  
 \*  
 \* Event processing function "Generating a random graph".  
 \* Checks the correct filling of the fields "Number of vertices" and "Number of edges".  
 \* Handles the "Generate Count" key.  
 \*/* @FXML  
 public void generateGraph() {  
 if (amountVertex.getText() == null || amountVertex.getText().length() == 0) {  
 error("Enter the number of vertices");  
 int x = Integer.*parseInt*(amountVertex.getText());  
 } else {  
 if (amountEdges.getText() == null || amountEdges.getText().length() == 0) {  
 error("Enter the number of edges");  
 int y = Integer.*parseInt*(amountEdges.getText());  
 }  
 try {  
 int x = Integer.*parseInt*(amountVertex.getText());  
 int y = Integer.*parseInt*(amountEdges.getText());  
 if (x > 30 || x < 0 || y > 10 || y < 0) {  
 error("The graph can not be displayed");  
 } else {  
 *P*.list.clear();  
 *P*.ways.clear();  
 *P*.road.clear();  
 *k* = 0;  
 *P*.n = x;  
 *P*.m = y;  
 *P*.V=-1;  
 textGraph.clear();  
 *P*.inputGeneration();  
 for (int i = 0; i < *P*.list.size(); i++) {  
 textGraph.appendText("Way from the top " + (*P*.list.elementAt(i).from + 1) + " to the top " + (*P*.list.elementAt(i).to + 1) + ": " + *P*.list.elementAt(i).l + "\n");  
 }  
 Stage stageWindow = new Stage();  
 }  
 } catch (NumberFormatException ex) {  
 ex.printStackTrace();  
 error("An incorrect value was entered in one of the fields");  
 }  
 }  
 }  
  
  
 */\*\*  
 \* Function for outputting errors when filling in fields.  
 \*  
 \** ***@param*** *s A string that tells you what error is committed when filling out the fields.  
 \*/* public void error(String s) {  
 alert = new Alert(Alert.AlertType.*ERROR*);  
 alert.setTitle("Invalid input");  
 alert.setHeaderText(s);  
 alert.showAndWait();  
 }  
  
 */\*\*  
 \* The function of starting the algorithm for finding the most granular paths of the graph.  
 \*/* public void beginAlgorithm() {  
 *k* = 1;  
 *P*.searchAlgorithm(this);  
 *P*.outputWays(this);  
 }  
  
 */\*\*  
 \* The function of processing the keystroke "Start the algorithm".  
 \*/* @FXML  
 public void workBegin() {  
 if (*P*.n == 0) {  
 error("Graph not generated");  
 } else {  
 if (textField1.getText() == null || textField1.getText().length() == 0) {  
 error("Enter the vertex");  
 } else {  
 try {  
 int m = Integer.*parseInt*(textField1.getText());  
 *P*.v = m;  
 if (*P*.v > *P*.n || *P*.v <= 0)  
 error("The vertex is not set correctly");  
 else beginAlgorithm();  
 } catch (NumberFormatException ex) {  
 ex.printStackTrace();  
 error("Incorrect value of the vertex");  
 }  
 }  
 }  
 }  
  
  
 */\*\*  
 \*  
 \* The function of processing the keystroke "Read graph from file".  
 \*  
 \** ***@throws*** *IOException  
 \*/* @FXML  
 public void fileGeneration () {  
 *k* = 0;  
 *P*.list.clear();  
 *P*.ways.clear();  
 *P*.road.clear();  
 *P*.V=-1;  
 textGraph.clear();  
 *P*.inputFile();  
 for (int i = 0; i < *P*.list.size(); i++) {  
 textGraph.appendText("The way from the top " + (*P*.list.elementAt(i).from + 1) + " to the top " + (*P*.list.elementAt(i).to + 1) + ": " + *P*.list.elementAt(i).l + "\n");  
 }  
 Stage stageWindow = new Stage();  
 }  
  
 */\*\*  
 \* Function of calling the child window to draw the graph.  
 \*/* public void FXMLDocumentController(Stage stageWindow) throws IOException {  
 Parent root = FXMLLoader.*load*(getClass().getResource("GUI\_prototype.fxml"));  
 stageWindow.setTitle("Graph");  
 Scene scene = new Scene(root);  
 stageWindow.setScene(scene);  
 stageWindow.show();  
 }  
  
 */\*\*  
 \* The function to display the components of the main panel when you press the "Visualize the algorithm" button.  
 \*/* @FXML  
 public void nextMenu() {  
 if (*k* == 0)  
 error("Algorithm did not start its work");  
 else {  
 next.setVisible(false);  
 text3.setVisible(false);  
 text4.setVisible(false);  
 text5.setVisible(false);  
 text6.setVisible(false);  
 amountEdges.setVisible(false);  
 amountVertex.setVisible(false);  
 textField1.setVisible(false);  
 begAlg.setVisible(false);  
 generateGraph.setVisible(false);  
 fileGraph.setVisible(false);  
 back.setVisible(true);  
 graphWay.setVisible(true);  
 vertex.setVisible(true);  
 graphAlg.setVisible(true);  
 graphV.setVisible(true);  
 *Setting* = false;  
 }  
 }  
  
 */\*\*  
 \* The function returns to the previous menu when the algorithm is visualized.  
 \*/* @FXML  
 public void backMenu() {  
 next.setVisible(true);  
 text3.setVisible(true);  
 text4.setVisible(true);  
 text5.setVisible(true);  
 text6.setVisible(true);  
 amountEdges.setVisible(true);  
 amountVertex.setVisible(true);  
 textField1.setVisible(true);  
 begAlg.setVisible(true);  
 generateGraph.setVisible(true);  
 fileGraph.setVisible(true);  
 back.setVisible(false);  
 graphWay.setVisible(false);  
 vertex.setVisible(false);  
 graphAlg.setVisible(false);  
 graphV.setVisible(false);  
 *Setting* = true;  
 }  
  
 */\*\*  
 \* Function of output of the shortest path to a given vertex.  
 \*/* @FXML  
 public void getWay() {  
 if (graphV.getText() == null || graphV.getText().length() == 0) {  
 error("Enter the number of vertices");  
 } else {  
  
 try {  
 int x = Integer.*parseInt*(graphV.getText());  
 x--;  
 *P*.V = x;  
 if (x + 1 > *P*.n || x + 1 < 0 || x == (*P*.v)) {//?  
  
 error("The graph can not be displayed");  
 } else {  
 Stage stageWindow = new Stage();  
 try {  
 FXMLDocumentController(stageWindow);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 } catch (NumberFormatException ex) {  
 ex.printStackTrace();  
 error("Incorrect value of the final vertex is entered.");  
 }  
 }  
 }  
}

***3) Файл Graph.java.***

package sample;  
*/\*\*  
 \* Ford-Bellman algorithm  
 \** ***@author*** *Bykov  
 \** ***@author*** *Lavrenkova  
 \** ***@author*** *Kuharev  
 \** ***@version*** *1.0  
 \*/*import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.Random;  
import java.util.Scanner;  
import java.util.Vector;  
  
*/\*\*  
 \* A class for storing the graph and displaying actions on it.  
 \*/*public class Graph implements GraphInterface {  
 Graph() {  
 negativeCircle = 0;  
 }  
  
 */\*\*  
 \* Class for storing graph edges.  
 \*/* public class ElementGraphWay {  
 int from;  
 int to;  
 int l;  
 }  
  
 */\*\*  
 \* A class for storing the coordinates of graph nodes.  
 \*/* public class WaysPoint {  
 int x;  
 int y;  
 int name;  
 }  
  
 private final Random random = new Random();  
 Vector<ElementGraphWay> list = new Vector<>();  
 Vector<Integer> ways = new Vector<>();  
 Vector<Integer> road = new Vector<>();  
 Vector<WaysPoint> visual = new Vector<>();  
 int n; // number of vertex  
 int m = 0; // number of edges  
 int v; // vertxe - beginner  
 int V = -1;  
 private int negativeCircle;  
  
 */\*\*  
 \* Reading the graph from the file  
 \*/* public void inputFile() {  
 negativeCircle = 0;  
 list.clear();  
 visual.clear();  
 Scanner sc;  
 try {  
 sc = new Scanner(new File("in.txt"));  
 try {  
 if (sc.hasNextInt()) {   
 n = sc.nextInt();  
 } else System.*out*.println("There is not enough data in the file");  
 if (sc.hasNextInt()) {  
 m = sc.nextInt();  
 } else System.*out*.println("There is not enough data in the file");  
 XY();  
 for (int j = 0; j < m; j++) {  
 ElementGraphWay Q = new ElementGraphWay();  
 if (sc.hasNextInt()) {  
 Q.from = sc.nextInt();  
 Q.from--;  
 }  
 if (sc.hasNextInt()) {  
 Q.to = sc.nextInt();  
 Q.to--;  
 }  
 if (sc.hasNextInt()) {  
 Q.l = sc.nextInt();  
 }  
 list.add(Q);  
  
 }  
 } catch (Exception ex) {  
 System.*out*.println("File is empty!");  
 }  
 } catch (FileNotFoundException ex) {  
 System.*out*.println("File is not exist!");  
 }  
  
 }  
  
 */\*\*  
 \*   
 \* Function of generating a random graph.  
 \*/* public void inputGeneration() {  
 negativeCircle = 0;  
 list.clear();  
 visual.clear();  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < m; j++) {  
 ElementGraphWay Q = new ElementGraphWay();  
 Q.from = i;  
 int q;// way to  
 do {  
 q = random.nextInt(n) + 1;  
 --q;  
 } while (q == i);  
 Q.to = q;  
 Q.l = random.nextInt(100);  
 list.add(Q);  
 }  
 }  
 m = n \* m;  
 XY();  
 }  
  
 */\*\*  
 \* The function of finding the shortest paths from a given vertex in a graph.  
 \*  
 \** ***@param*** *P controller, in which the results will be displayed.  
 \*/* public void searchAlgorithm(Controller P) {  
 int inf = Integer.*MAX\_VALUE*;  
 for (int i = 0; i < n; i++) {  
 ways.add(inf);  
 road.add(-1);  
 }  
 v--;// vertex from  
 ways.set(v, 0);  
 P.graphAlg.clear();  
 P.graphAlg.appendText((v + 1) + "|\t");  
 for (int k = 1; k <= n; k++) {  
 P.graphAlg.appendText(k + "|\t");  
 }  
 P.graphAlg.appendText("\n\t");  
 for (int k = 0; k < n; k++) {  
 if (ways.elementAt(k) == inf)  
 P.graphAlg.appendText("∞" + "\t");  
 else {  
 if (ways.elementAt(k) > 9)  
 P.graphAlg.appendText(ways.elementAt(k) + "\t");  
 else P.graphAlg.appendText(ways.elementAt(k) + "\t");  
 }  
 }  
 P.graphAlg.appendText("\n");  
 for (int i = 1; i <= n; ++i) {  
 for (int j = 0; j < m; ++j) {  
 if ((ways.elementAt(list.elementAt(j).from) < inf) & ((ways.elementAt(list.elementAt(j).from) + list.elementAt(j).l) < ways.elementAt(list.elementAt(j).to))) {  
 if (i == n) {  
 negativeCircle = 1;  
 P.graphAlg.clear();  
 P.result.clear();  
 P.graphAlg.appendText("The graph has negative cycles");  
 P.result.appendText("The graph has negative cycles");  
 return;  
 } else {  
 ways.set(list.elementAt(j).to, (ways.elementAt(list.elementAt(j).from) + list.elementAt(j).l));  
 road.set(list.elementAt(j).to, list.elementAt(j).from);  
 P.graphAlg.appendText("\t");  
 for (int k = 0; k < n; k++) {  
 if (ways.elementAt(k) == inf)  
 P.graphAlg.appendText("∞"+"\t");  
 else {  
 if (ways.elementAt(k) > 99)  
 P.graphAlg.appendText(ways.elementAt(k) + "\t");  
 else P.graphAlg.appendText(ways.elementAt(k) + "\t");  
 }  
 }  
 P.graphAlg.appendText("\n");  
  
 }  
 }  
 }  
  
 }  
 }  
  
  
  
 */\*\*  
 \*   
 \* The function of derivation of shortest paths from a given vertex in a graph.  
 \*  
 \** ***@param*** *P controller, in which the results will be displayed.  
 \*/* public void outputWays(Controller P) {  
 if (negativeCircle == 0) {  
 P.result.clear();  
 Vector<Integer> path = new Vector<>();  
 for (int j = 0; j < n; j++) {  
 if (j != (v)) {  
 if (ways.elementAt(j) == Integer.*MAX\_VALUE*) {  
 P.result.appendText("Way from the top " + (v + 1) + " to the top " + (j + 1) + ": NO\n");  
 } else {  
 path.clear();  
 for (int cur = j; cur != -1; cur = road.elementAt(cur))  
 path.add(cur);  
 P.result.appendText("Way from the top " + (v + 1) + " to the top " + (j + 1) + ": " + ways.elementAt(j) + "\nShortest way: ");  
 for (int i = path.size() - 1; i >= 1; i--) {  
 int l = (path.elementAt(i) + 1);  
 P.result.appendText(l + "->");  
 }  
 P.result.appendText((path.elementAt(0) + 1) + "\n");  
 path.clear();  
 }  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* The function of assigning coordinates to the nodes of the graph.  
 \*/* private void XY() {  
 double fi = 360 / n;  
 for (int i = 0; i < n; i++) {  
 WaysPoint q = new WaysPoint();  
 int x = 130;  
 int vertexX = 120;  
 q.x = x + (int) (long) (vertexX \* Math.*cos*(i \* fi \* Math.*PI* / 180));  
 int vertexY = 105;  
 int y = 105;  
 q.y = y + (int) (long) (vertexY \* Math.*sin*(i \* fi \* Math.*PI* / 180));  
 q.name = i + 1;  
 visual.add(q);  
 }  
 }  
  
}

***4) Файл GraphInterface.java.***

package sample;  
  
*/\*\*  
 \* The interface for implementing the graph.  
 \*/*public interface GraphInterface {  
 void inputFile();  
  
 void inputGeneration();  
  
 void searchAlgorithm(Controller P);  
  
 void outputWays(Controller P);  
}

***5) Файл GraphController.java.***

***6) Файл sample.fxml.***

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.Cursor?>  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.control.Label?>  
<?import javafx.scene.control.SplitPane?>  
<?import javafx.scene.control.TextArea?>  
<?import javafx.scene.control.TextField?>  
<?import javafx.scene.layout.AnchorPane?>  
<?import javafx.scene.layout.ColumnConstraints?>  
<?import javafx.scene.layout.GridPane?>  
<?import javafx.scene.layout.RowConstraints?>  
<?import javafx.scene.text.Font?>  
  
<?import javafx.scene.effect.ImageInput?>  
<?import javafx.scene.image.Image?>  
<GridPane alignment="center" hgap="10" vgap="10" xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1" fx:controller="sample.Controller">  
 <children>  
 <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="678.0" prefWidth="965.0" GridPane.columnIndex="1" GridPane.rowIndex="1" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1">  
 <children>  
 <SplitPane layoutX="-205.0" layoutY="-59.0" prefHeight="737.0" prefWidth="383.0" style="-fx-border-color: whitesmoke;" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="-205.0" AnchorPane.rightAnchor="787.0" AnchorPane.topAnchor="-59.0">  
 <effect>  
 <ImageInput>  
 <source>  
 <Image url="file:/C:/Users/admin/IdeaProjects/ford-bellman/background.jpg" />  
 </source>  
 </ImageInput>  
 </effect>  
 </SplitPane>  
 <Button fx:id="fileGraph" layoutX="588.0" layoutY="145.0" mnemonicParsing="false" onAction="#fileGeneration" prefHeight="52.0" prefWidth="304.0" text="Read from the file" textFill="#2d03a1">  
 <font>  
 <Font size="18.0" />  
 </font>  
 <cursor>  
 <Cursor fx:constant="HAND" />  
 </cursor>  
 </Button>  
 <Label fx:id="text6" layoutX="585.0" layoutY="470.0" prefHeight="69.0" prefWidth="308.0" text="Enter the vertex of the graph:" textFill="#653794">  
 <font>  
 <Font size="18.0" />  
 </font>  
 </Label>  
 <Button fx:id="generateGraph" layoutX="586.0" layoutY="413.0" mnemonicParsing="false" onAction="#generateGraph" prefHeight="47.0" prefWidth="307.0" text="Generate graph" textFill="#2d03a1">  
 <font>  
 <Font size="18.0" />  
 </font>  
 <cursor>  
 <Cursor fx:constant="HAND" />  
 </cursor>  
 </Button>  
 <TextField fx:id="textField1" layoutX="585.0" layoutY="520.0" prefHeight="47.0" prefWidth="310.0">  
 <cursor>  
 <Cursor fx:constant="TEXT" />  
 </cursor>  
 <font>  
 <Font size="14.0" />  
 </font>  
 </TextField>  
 <Button fx:id="begAlg" layoutX="584.0" layoutY="572.0" mnemonicParsing="false" onAction="#workBegin" prefHeight="40.0" prefWidth="312.0" text="Start the algorithm" textFill="#2d03a1">  
 <font>  
 <Font size="18.0" />  
 </font>  
 <cursor>  
 <Cursor fx:constant="HAND" />  
 </cursor>  
 </Button>  
 <Label fx:id="labelFB" layoutX="14.0" layoutY="15.0" prefHeight="105.0" prefWidth="515.0" text="Ford-Bellman algorithm" textFill="#653794">  
 <font>  
 <Font name="ACaslonPro-Regular" size="40.0" />  
 </font>  
 </Label>  
 <Label fx:id="menu" layoutX="663.0" layoutY="32.0" prefHeight="72.0" prefWidth="230.0" text="Menu" textFill="#653794">  
 <font>  
 <Font name="ACaslonPro-Regular" size="40.0" />  
 </font>  
 </Label>  
 <Label fx:id="text3" layoutX="587.0" layoutY="221.0" prefHeight="35.0" prefWidth="305.0" text="Enter the number of vertices (less 30):" textFill="#653794">  
 <font>  
 <Font size="18.0" />  
 </font>  
 </Label>  
 <TextArea fx:id="amountVertex" layoutX="587.0" layoutY="257.0" prefHeight="40.0" prefWidth="304.0">  
 <font>  
 <Font size="14.0" />  
 </font></TextArea>  
 <Label fx:id="text4" layoutX="585.0" layoutY="297.0" prefHeight="35.0" prefWidth="311.0" text="Enter the number of edges" textFill="#653794">  
 <font>  
 <Font size="18.0" />  
 </font>  
 </Label>  
 <Label fx:id="text5" layoutX="585.0" layoutY="324.0" prefHeight="30.0" prefWidth="312.0" text="from one vertex (less 10):" textFill="#653794">  
 <font>  
 <Font size="18.0" />  
 </font>  
 </Label>  
 <TextArea fx:id="amountEdges" layoutX="586.0" layoutY="359.0" prefHeight="46.0" prefWidth="308.0">  
 <font>  
 <Font size="14.0" />  
 </font></TextArea>  
 <TextArea fx:id="result" layoutX="21.0" layoutY="416.0" prefHeight="150.0" prefWidth="498.0" />  
 <Label fx:id="text2" layoutX="19.0" layoutY="358.0" prefHeight="71.0" prefWidth="249.0" text="The result of the algorithm:" textFill="#653794">  
 <font>  
 <Font size="18.0" />  
 </font>  
 </Label>  
 <Label fx:id="text1" layoutX="21.0" layoutY="93.0" prefHeight="112.0" prefWidth="130.0" text="Assigned graph:" textFill="#653794">  
 <font>  
 <Font size="18.0" />  
 </font>  
 </Label>  
 <TextArea fx:id="textGraph" layoutX="21.0" layoutY="173.0" prefHeight="190.0" prefWidth="501.0" AnchorPane.bottomAnchor="315.0" AnchorPane.leftAnchor="21.0" AnchorPane.rightAnchor="443.0" AnchorPane.topAnchor="173.0" />  
 <Button fx:id="next" layoutX="23.0" layoutY="571.0" mnemonicParsing="false" onAction="#nextMenu" prefHeight="39.0" prefWidth="495.0" text="Visualization of the algorithm's work" textFill="#2d03a1">  
 <font>  
 <Font size="18.0" />  
 </font>  
 </Button>  
 <Button fx:id="back" layoutX="20.0" layoutY="571.0" mnemonicParsing="false" onAction="#backMenu" prefHeight="39.0" prefWidth="498.0" text="Back" textFill="#2d03a1" visible="false">  
 <font>  
 <Font size="18.0" />  
 </font>  
 </Button>  
 <Label fx:id="vertex" layoutX="584.0" layoutY="480.0" prefHeight="143.0" prefWidth="319.0" text="Enter the vertex of the graph:" textFill="#8674ee" visible="false">  
 <font>  
 <Font size="18.0" />  
 </font>  
 </Label>  
 <TextArea fx:id="graphV" layoutX="587.0" layoutY="572.0" prefHeight="46.0" prefWidth="123.0" visible="false" />  
 <Button fx:id="graphWay" layoutX="720.0" layoutY="574.0" mnemonicParsing="false" onAction="#getWay" prefHeight="30.0" prefWidth="176.0" text="Path" textFill="#2d03a1" visible="false">  
 <font>  
 <Font size="18.0" />  
 </font>  
 </Button>  
 <TextArea fx:id="graphAlg" layoutX="584.0" layoutY="139.0" prefHeight="382.0" prefWidth="308.0" visible="false" />  
 </children>  
 <cursor>  
 <Cursor fx:constant="HAND" />  
 </cursor>  
 </AnchorPane>  
 </children>  
 <columnConstraints>  
 <ColumnConstraints />  
 <ColumnConstraints />  
 </columnConstraints>  
 <rowConstraints>  
 <RowConstraints />  
 <RowConstraints />  
 </rowConstraints>  
</GridPane>

***7) Файл GUI\_prototype.fxml.***

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.control.Label?>  
<?import javafx.scene.layout.AnchorPane?>  
<?import javafx.scene.layout.ColumnConstraints?>  
<?import javafx.scene.layout.GridPane?>  
<?import javafx.scene.layout.Pane?>  
<?import javafx.scene.layout.RowConstraints?>  
<?import javafx.scene.text.Font?>  
  
<?import javafx.scene.effect.ImageInput?>  
<?import javafx.scene.image.Image?>  
<?import javafx.scene.control.SplitPane?>  
<GridPane alignment="center" hgap="10" vgap="10" xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1" fx:controller="sample.GraphController">  
 <columnConstraints>  
 <ColumnConstraints />  
 <ColumnConstraints />  
 <ColumnConstraints />  
 <ColumnConstraints />  
 </columnConstraints>  
 <rowConstraints>  
 <RowConstraints />  
 </rowConstraints>  
 <children>  
 <AnchorPane prefHeight="578.0" prefWidth="671.0" GridPane.columnIndex="1">  
 <children>  
 <SplitPane layoutX="-205.0" layoutY="-59.0" prefHeight="737.0" prefWidth="383.0" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="-205.0" AnchorPane.rightAnchor="787.0" AnchorPane.topAnchor="-59.0">  
 <effect>  
 <ImageInput>  
 <source>  
 <Image url="file:/C:/Users/admin/IdeaProjects/ford-bellman/background.jpg" />  
 </source>  
 </ImageInput>  
 </effect>  
 </SplitPane>  
 <Label layoutX="190.0" layoutY="-97.0" prefHeight="256.0" prefWidth="411.0" text="The given graph" textFill="#653794" AnchorPane.bottomAnchor="419.0" AnchorPane.leftAnchor="190.0" AnchorPane.rightAnchor="120.0" AnchorPane.topAnchor="-97.0">  
 <font>  
 <Font name="ACaslonPro-Regular" size="47.0" />  
 </font>  
 </Label  
 <Button fx:id="buildBut" layoutX="24.0" layoutY="513.0" mnemonicParsing="false" onAction="#graphButton" prefHeight="47.0" prefWidth="142.5" text="Build" textFill="#2c00da" AnchorPane.bottomAnchor="18.0" AnchorPane.leftAnchor="24.0" AnchorPane.rightAnchor="457.0" AnchorPane.topAnchor="513.0">  
 <font>  
 <Font size="18.0" />  
 </font></Button>  
 <Pane fx:id="pane1" layoutX="208.0" layoutY="169.0" prefHeight="100.0" prefWidth="256.0" scaleX="2.2" scaleY="1.6" snapToPixel="false" AnchorPane.bottomAnchor="153.0" AnchorPane.leftAnchor="208.0" AnchorPane.rightAnchor="207.0" AnchorPane.topAnchor="169.0" />  
 <Button fx:id="prevButton" layoutX="467.0" layoutY="513.0" mnemonicParsing="false" onAction="#prevStep" prefHeight="47.0" prefWidth="142.5" text="Previous step" textFill="#2c00da" AnchorPane.bottomAnchor="18.0" AnchorPane.leftAnchor="233.0" AnchorPane.rightAnchor="223.0" AnchorPane.topAnchor="513.0">  
 <font>  
 <Font size="18.0" />  
 </font>  
 </Button>  
 <Button fx:id="nextButton" layoutX="467.0" layoutY="513.0" mnemonicParsing="false" onAction="#nextStep" prefHeight="47.0" prefWidth="142.5" text="Next step" textFill="#2c00da" AnchorPane.bottomAnchor="18.0" AnchorPane.leftAnchor="467.0" AnchorPane.rightAnchor="14.0" AnchorPane.topAnchor="513.0">  
 <font>  
 <Font size="18.0" />  
 </font>  
 </Button>  
 <Button fx:id="nextButton1" layoutX="234.0" layoutY="450.0" mnemonicParsing="false" onAction="#graphPaint" prefHeight="47.0" prefWidth="213.0" text="Draw" textFill="#2c00da">  
 <font>  
 <Font size="18.0" />  
 </font>  
 </Button>  
 </children>  
 </AnchorPane>  
 </children>  
</GridPane>