**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»**

Студентка гр. 5382 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Черненко А.Б.

Студентка гр. 5382 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лисс Н.И.

Студентка гр. 5382 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Коппель Т.С.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Герасимова Т.В.

Санкт-Петербург

2017

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студентка Черненко А.Б. группы 5382 | | |
| Студентка Лисс Н.И. группы 5382 | | |
| Студентка Коппель Т.С. группы 5382  Тема практики: Визуализация алгоритмов на графах на Java | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.  Алгоритм: Беллмана — Форда | | |
| Сроки прохождения практики: 21.06.2017 – 04.07.2017 | | |
| Дата сдачи отчета: 30.06.2017 | | |
| Дата защиты отчета: 30.06.2017 | | |
|  | | |
| Студентка |  | Черненко А.Б. |
| Студентка |  | Лисс Н.И. |
| Студентка |  | Коппель Т.С. |
| Руководитель |  | Герасимова Т.В. |

**АННОТАЦИЯ**

Целью данной работы является разработка визуализатора алгоритма Беллмана — Форда на языке Java с использованием графического интерфейса. Данный алгоритм производит поиск кратчайшего пути во взвешенном графе. В мини-проекте ведется работа с графами. В результате выполнения работы программы, пользователь получает граф, на котором отображен кратчайший путь от одной вершины к другой.

**annotation**

The purpose of this work is to develop a visualizer of the Bellman-Ford algorithm in the Java language using a graphical interface. This algorithm searches for the shortest path in a weighted graph. The mini-project is working with the graphs. Like the result of the program, user have a graph that displays the shortest path from one vertex to another.

**Содержание**

[Отчет 1](#_Toc486917567)

[по учебной практике 1](#_Toc486917568)

[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 10](#_Toc486917578)

[5.1. План разработки 10](#_Toc486917579)

[5.2. Распределение ролей в бригаде 10](#_Toc486917580)

[6. ТЕСТИРОВАНИЕ ПРОГРАММЫ 11](#_Toc486917581)

[6.1. Тестирование работы программы при вводе из файла in.txt: 11](#_Toc486917582)

[6.2. Тестирование работы программы с помощью случайной генерации. 16](#_Toc486917583)

[7. ЗАКЛЮЧЕНИЕ 22](#_Toc486917584)

[8. ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД 23](#_Toc486917585)

# 1. ИСХОДНОЕ ЗАДАНИЕ

В данной практической работе требуется написать программу на языке JAVA, выполняющую поиск Кратчайших путей в графе с помощью алгоритма Беллмана — Форда для неотрицательных графов 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. Случай наличия отрицательного цикла будет рассмотрен ниже в отдельном разделе.

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 ∞.

# 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 «Считать граф из файла», либо сгенерировать, при нажатии «Сгенерировать граф».

## 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 «Начать работу алгоритма».

Затем к созданному графу применяется алгоритм Форда-Беллмана. Пользователь сможет просмотреть работу алгоритма в текстовом поле и запустить визуализацию работы алгоритма, нажав «Визуализация работы алгоритма». Далее пользователю предлагается ввести конечную вершину графа с помощью текстового поля и кнопки «Путь».

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

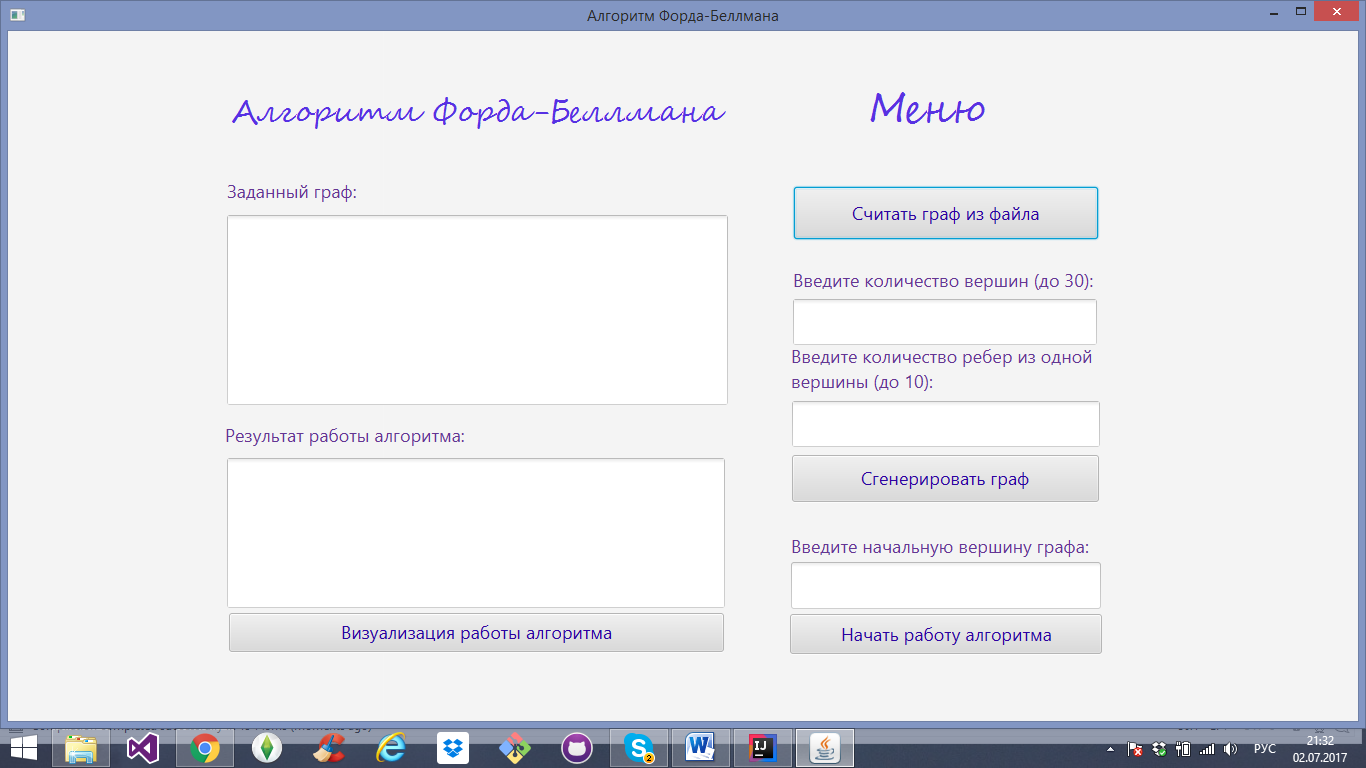
Выходными данными является визуализированный алгоритм: построенный кратчайший путь между начальной и конечной вершинами.

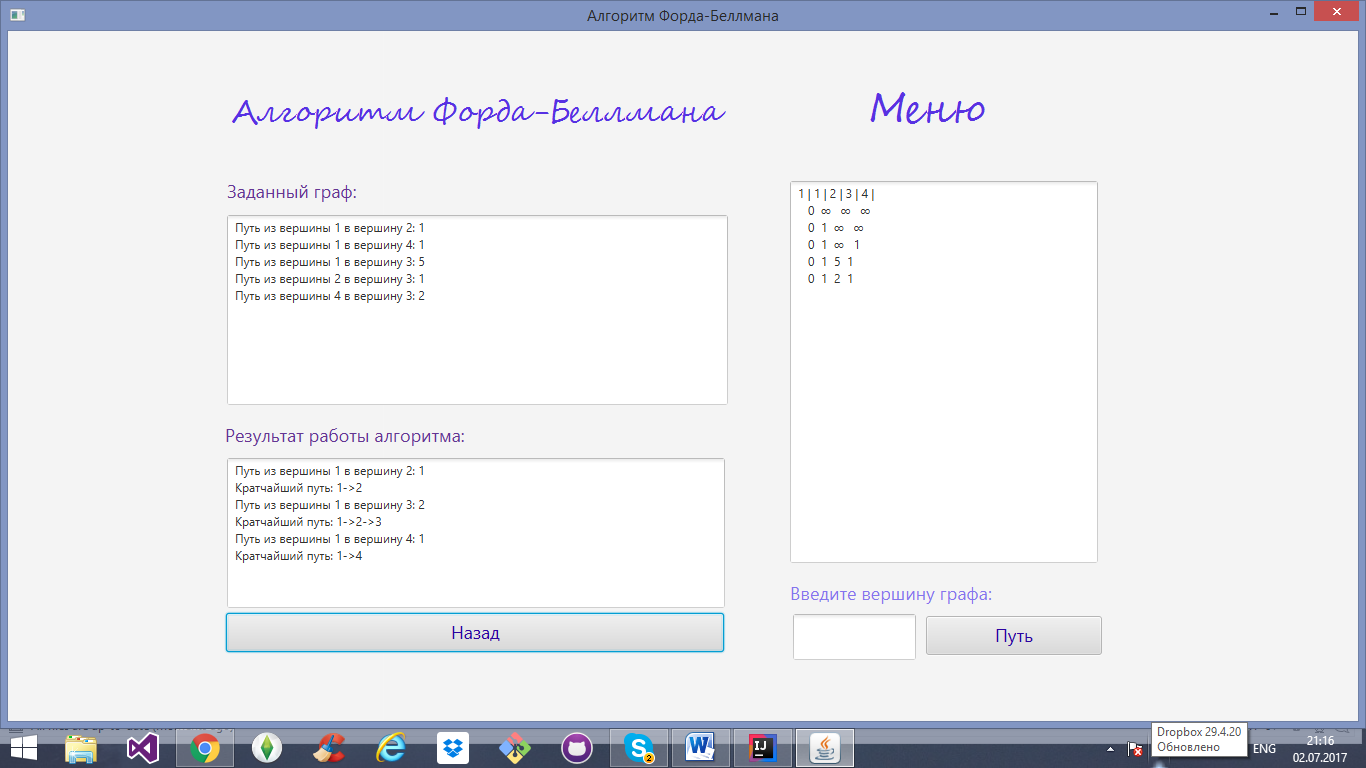
Кратчайший путь в итоговом графе выделяется красным цветом.

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.

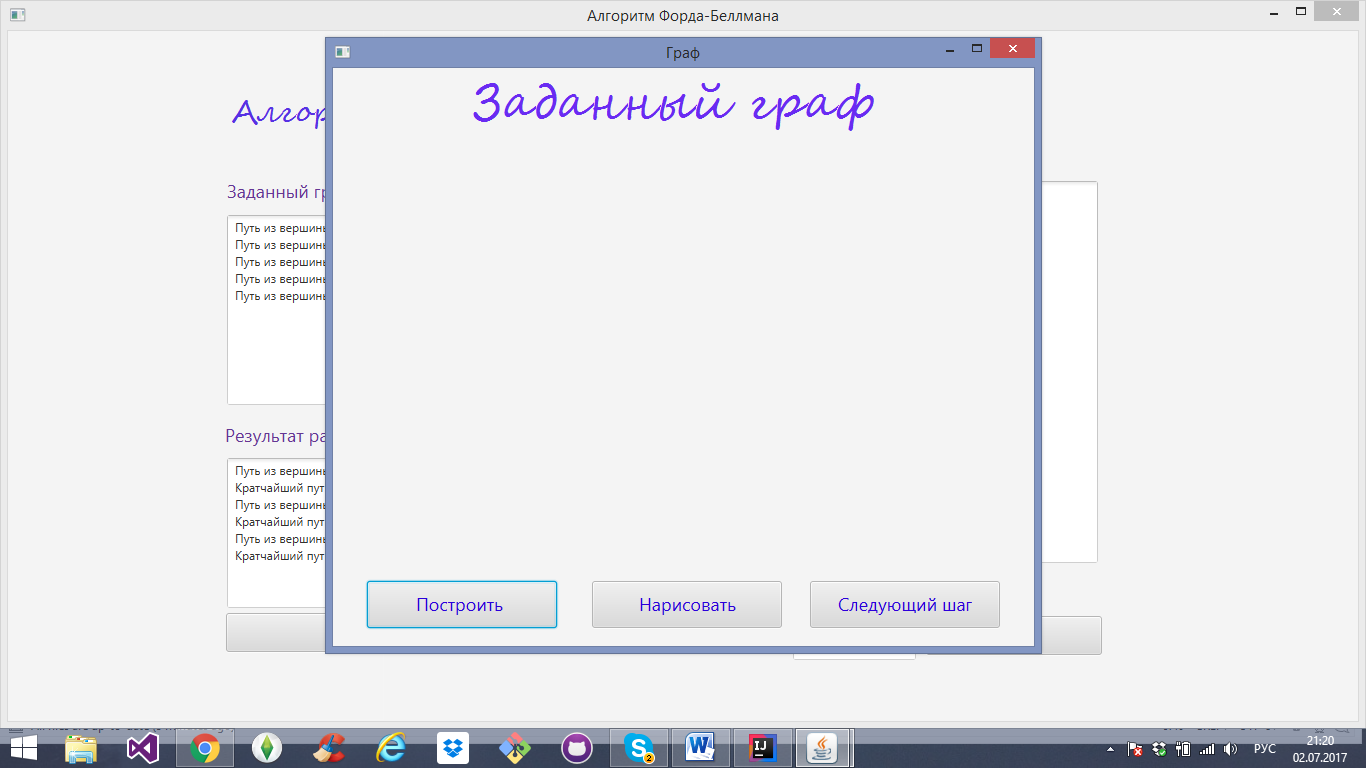
Программа реализована на языке Java с использованием Фреймворка JavaFX.

Далее представлен план Меню:   
Рис.1.

   
Рис.2.



Далее представлен план дочернего окна.

Рис.3.  


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 рандомного 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 рандомные 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) 23.06.2017 – написание спецификации программы, согласование написанной спецификации с преподавателем.  
2) 26.06.2017 - демонстрация пользовательского интерфейса. Реализация алгоритма Беллмана – Форда.   
3) 28.06.2017 - сдача демоверсии программы. Рабочая программа с некоторыми недочетами.  
4) 30.06.2017 - сдача готовой версии. Программа должна корректно работать на всех заданных тестах.

## 5.2. Распределение ролей в бригаде

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

|  |  |
| --- | --- |
| Лисс Н.И. | Визуализация на JavaFX, графический интерфейс. |
| Черненко А.Б. | Реализация алгоритма Форда-Беллмана. |
| Коппель Т.С. | Реализация ребер и вершин. Создание пользовательского и случайно сгенерированного графа. |

**Использованные структуры данных:**

Граф представлен списком двух вершин и весом ребра между ними.

*Класс для хранения ребер графа:*

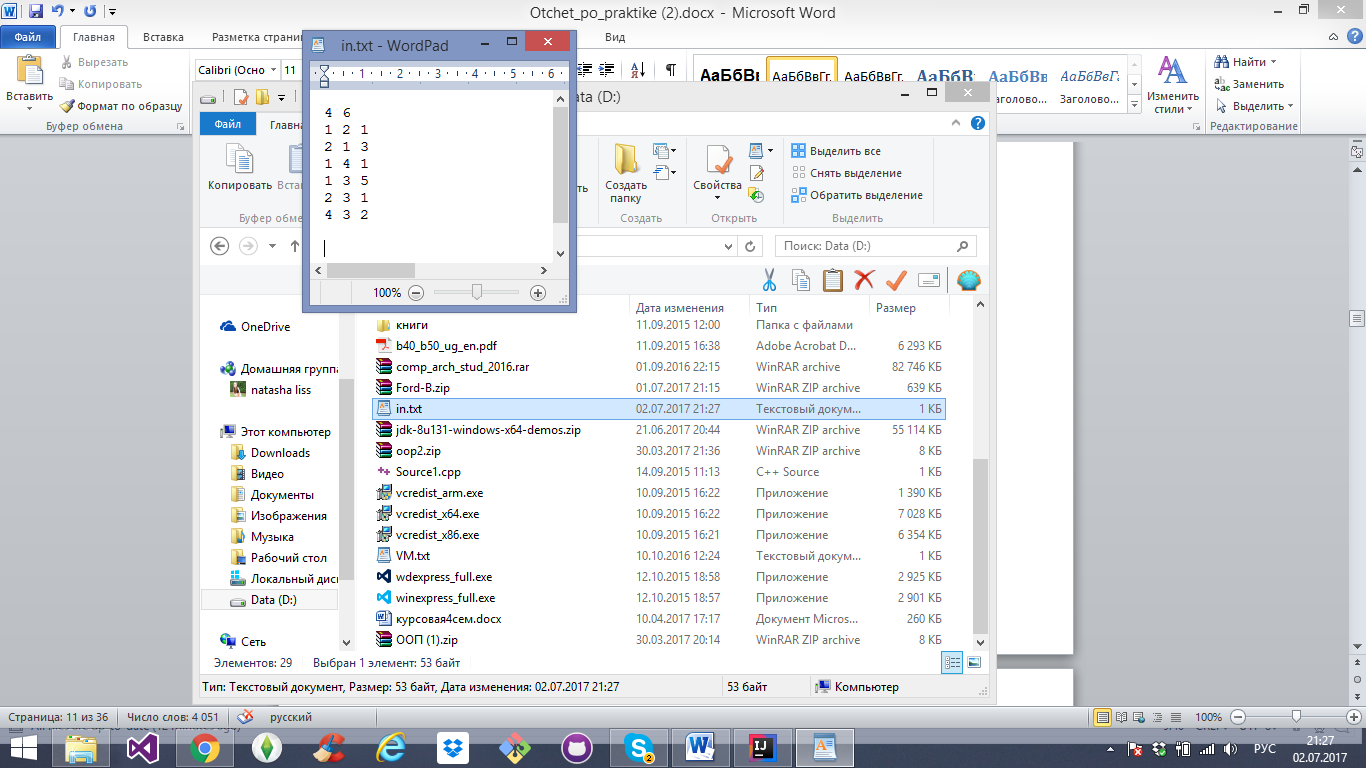
**public class** ElementGraphWay {  
 **int from**; //начальная вершина  
 **int to**; //конечная вершина  
 **int l**; //вес ребра  
}

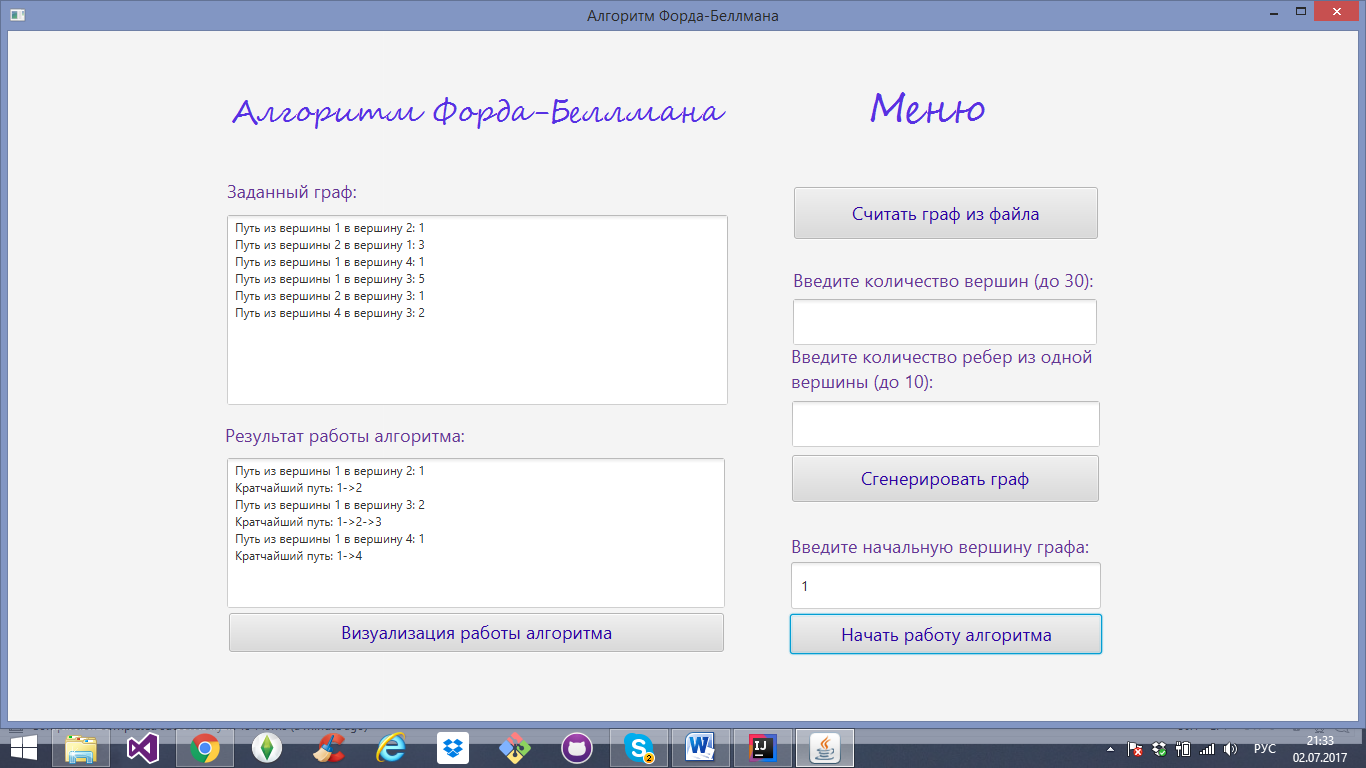
*Класс для хранения координат узлов графа:***public class** WaysPoint {  
 **int x**; //х-координата вершины  
 **int y**; //у-координата вершины  
 **int Name**;//имя вершины  
}

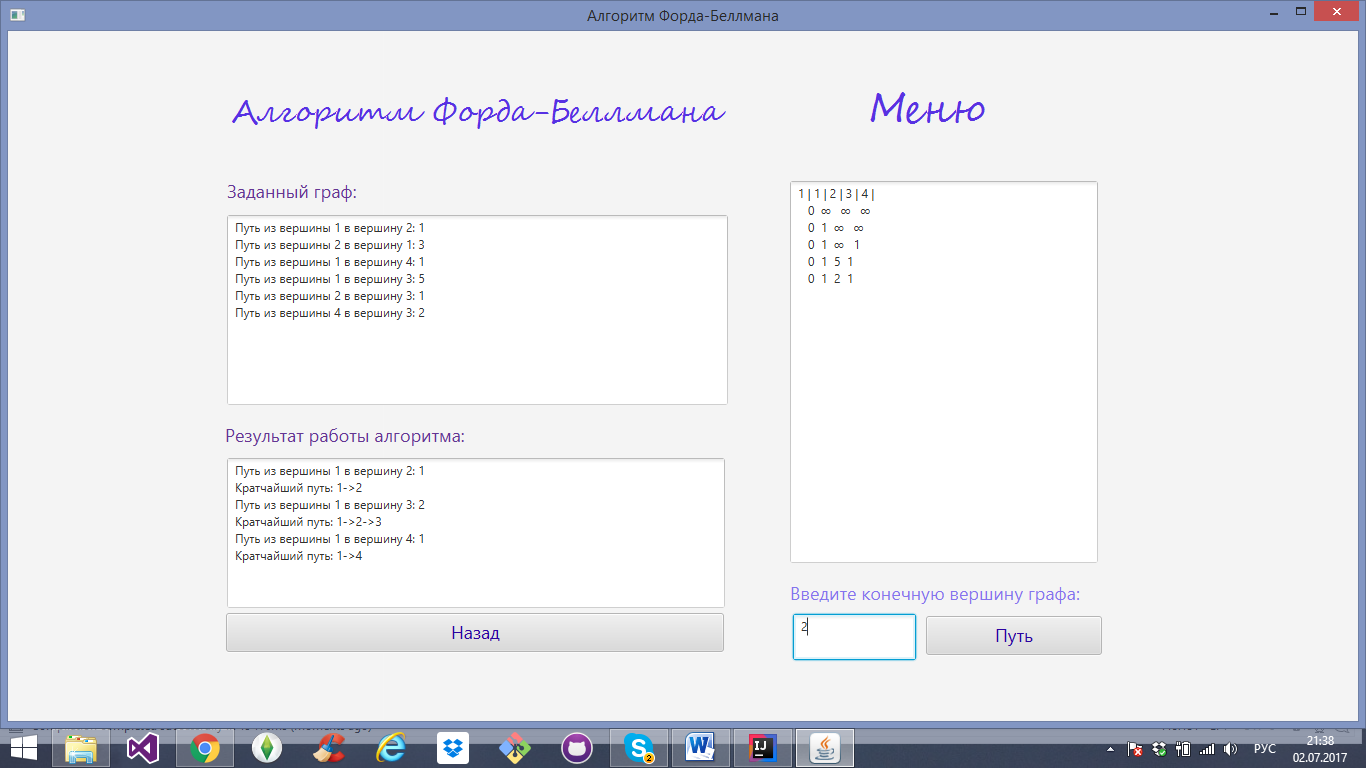
# 6. ТЕСТИРОВАНИЕ ПРОГРАММЫ

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

После того, как открылось главное окно с меню, пользователь наживает на кнопку «Считать граф из файла»



Далее пользователь вводит начальную вершину, например, 1. Пользователю в текстовом поле виден результат работы алгоритма: 

Далее, после нажатия кнопки «Визуализация работы алгоритма», пользователя просят ввести конечную вершину, т.е. вершину, в которую нужно будет найти кратчайший путь, например, 2: 

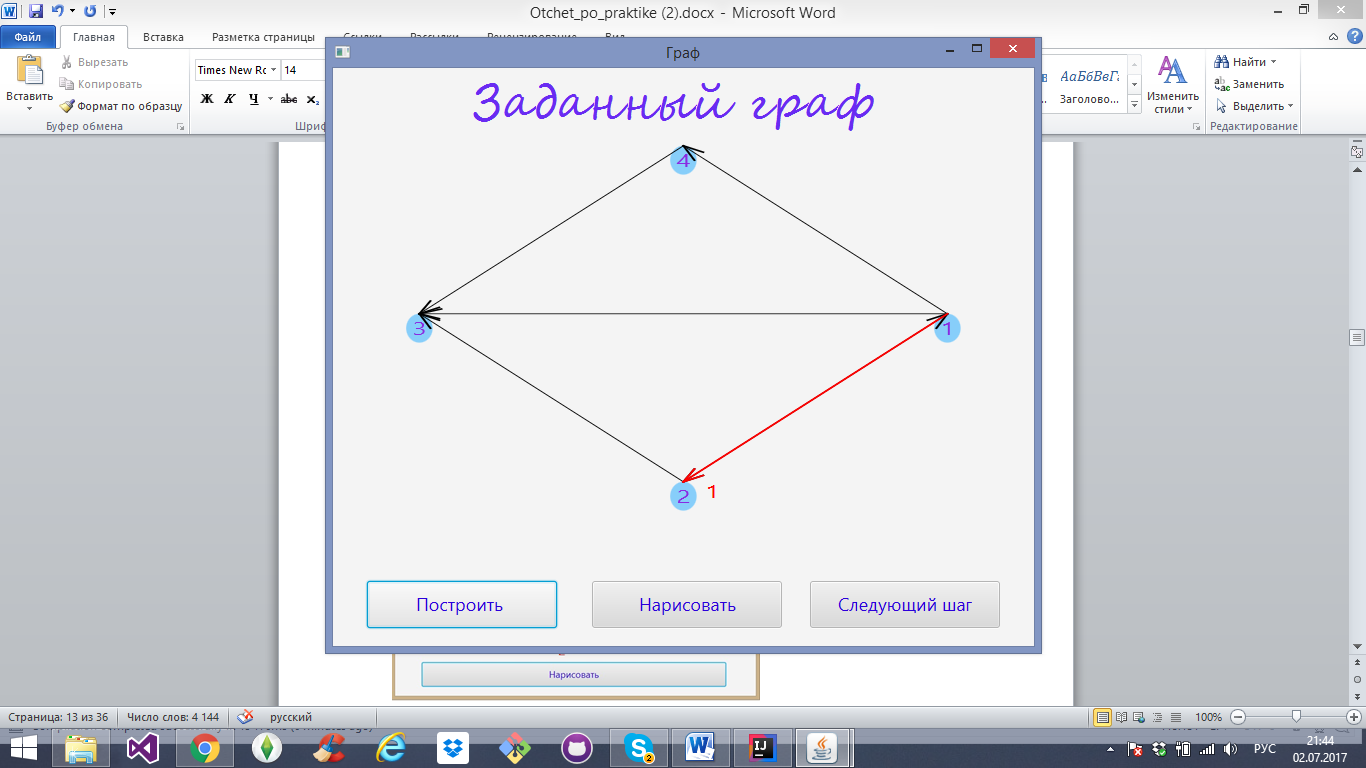
После нажатия кнопки «Путь», на экране появится дочернее окно с тремя кнопками: «Построить», «Нарисовать» и «Следующий шаг».

Если пользователь хочет вывести на экран конечный результат, ему нужно нажать кнопку «Построить».

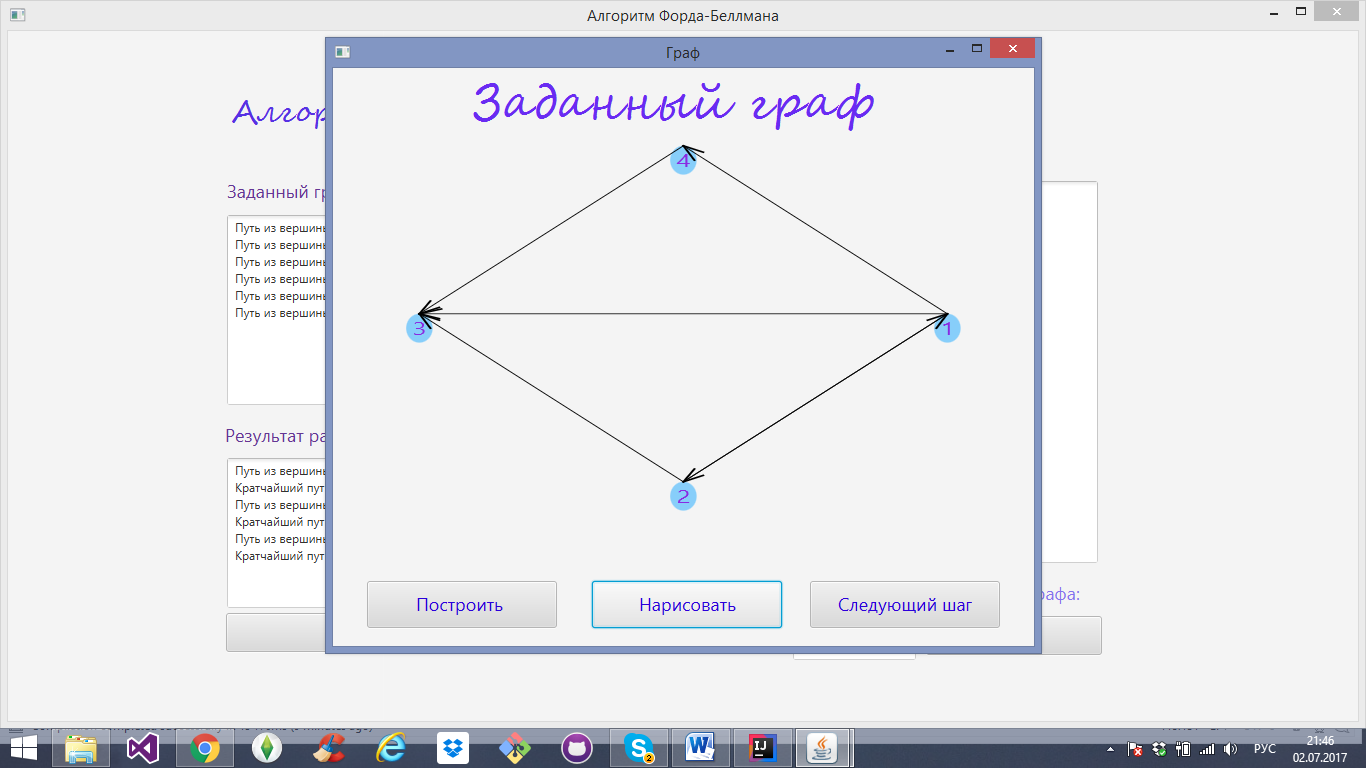
Если пользователь хочет вывести на экран пошаговое выполнение алгоритма, то ему нужно нажать сначала кнопку «Нарисовать», а затем «Следующий шаг», пока алгоритм не закончит свою работу.

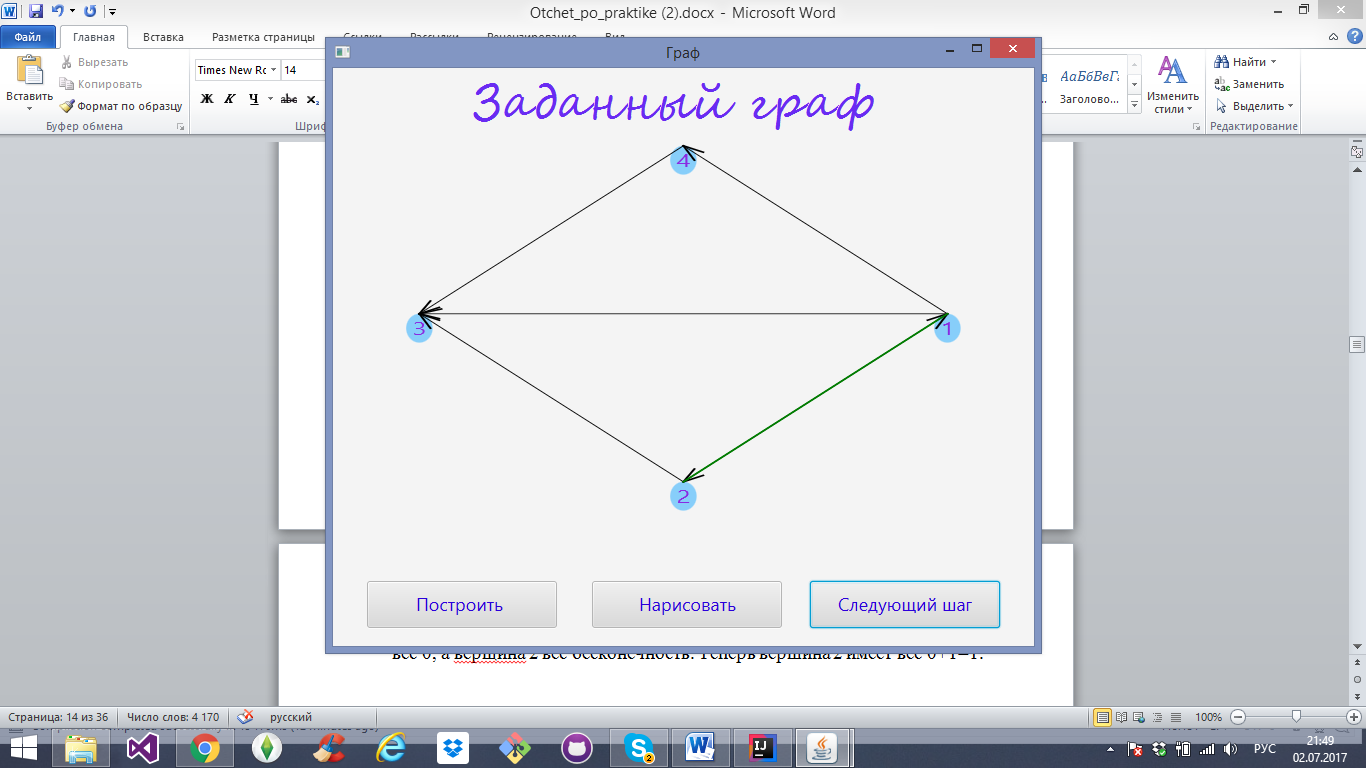
1)Вывод конечного результата.

Кратчайший путь из вершины 1 до вершины 2 выделен красным цветом:

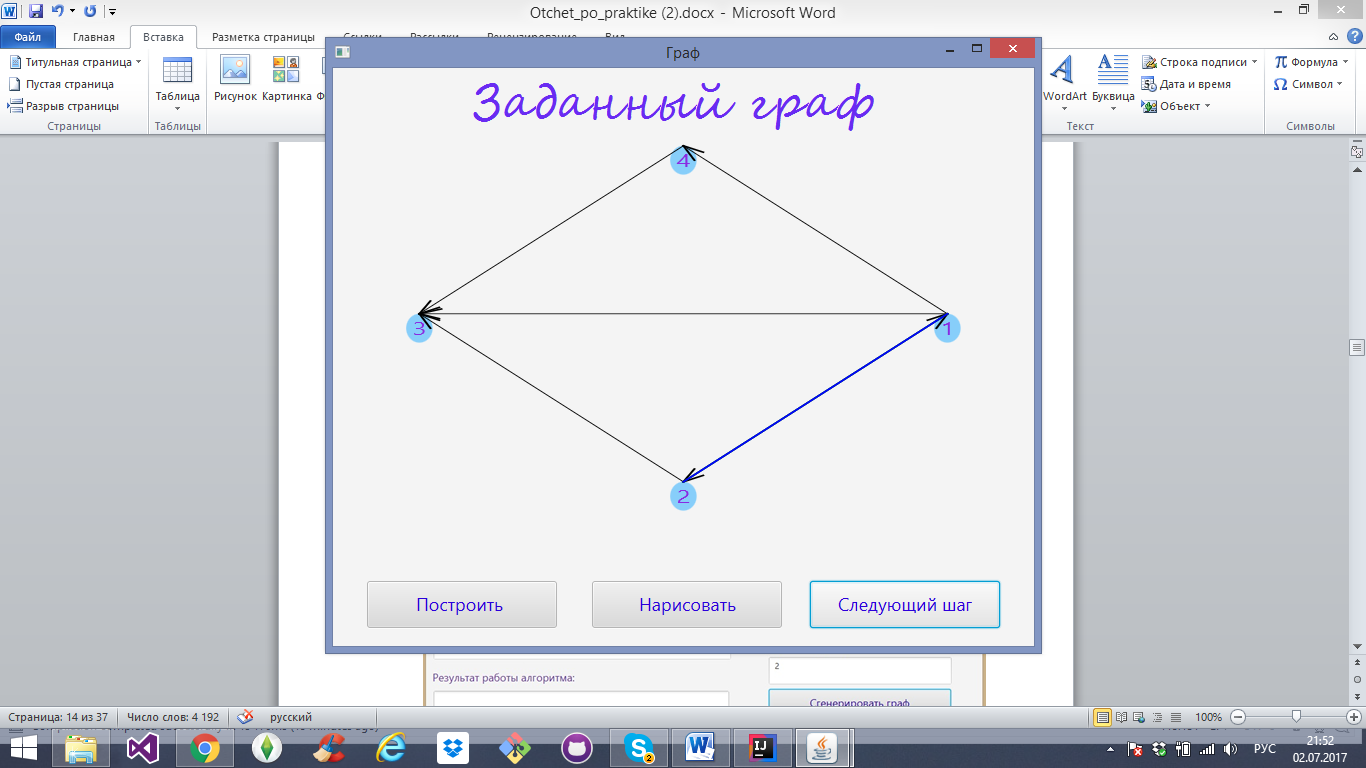


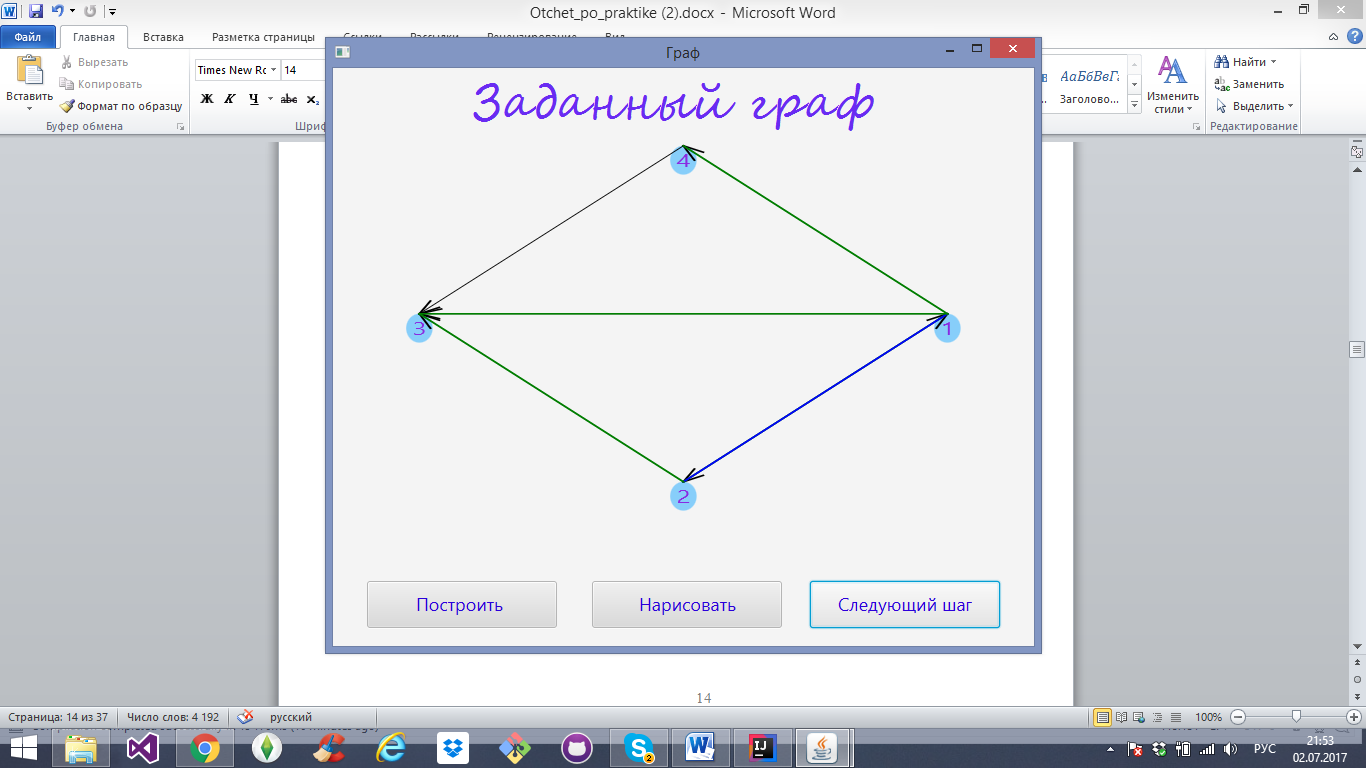
2)Пошаговое выполнение.

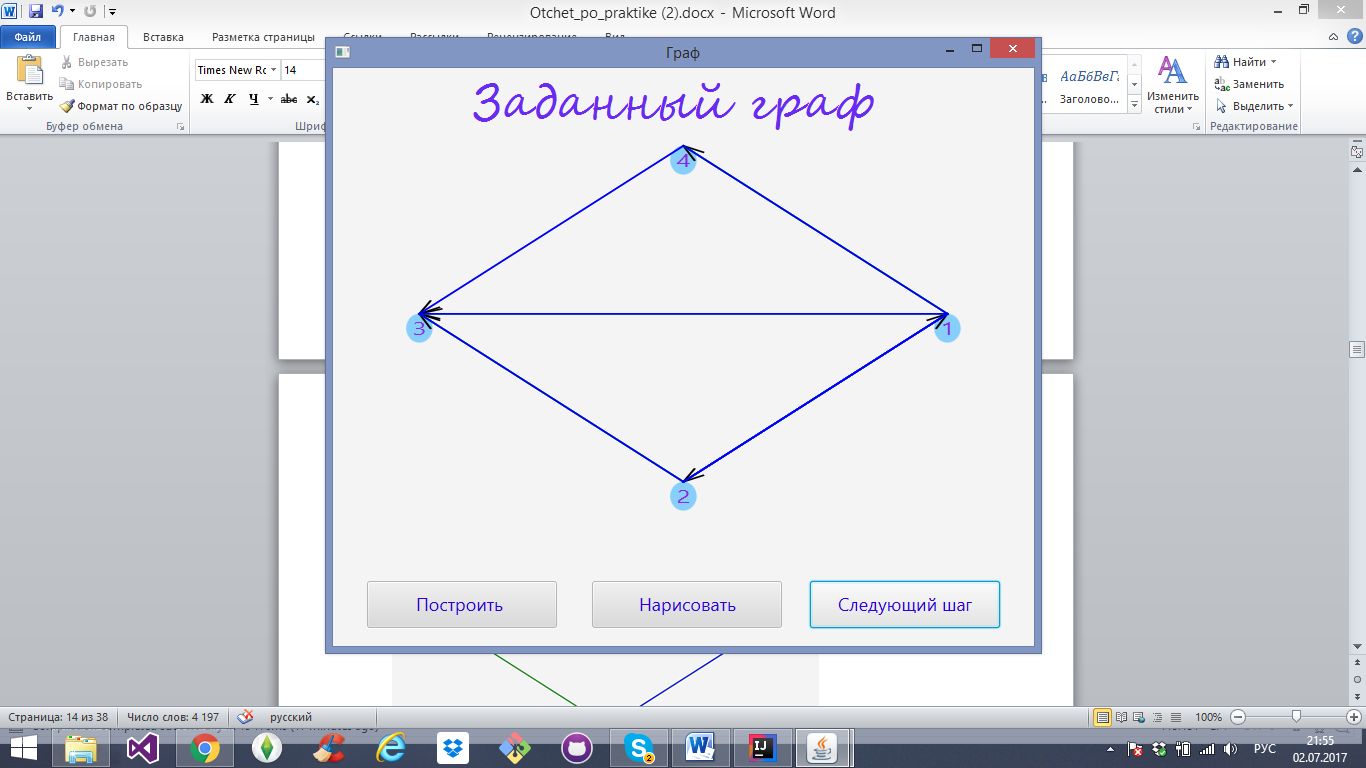


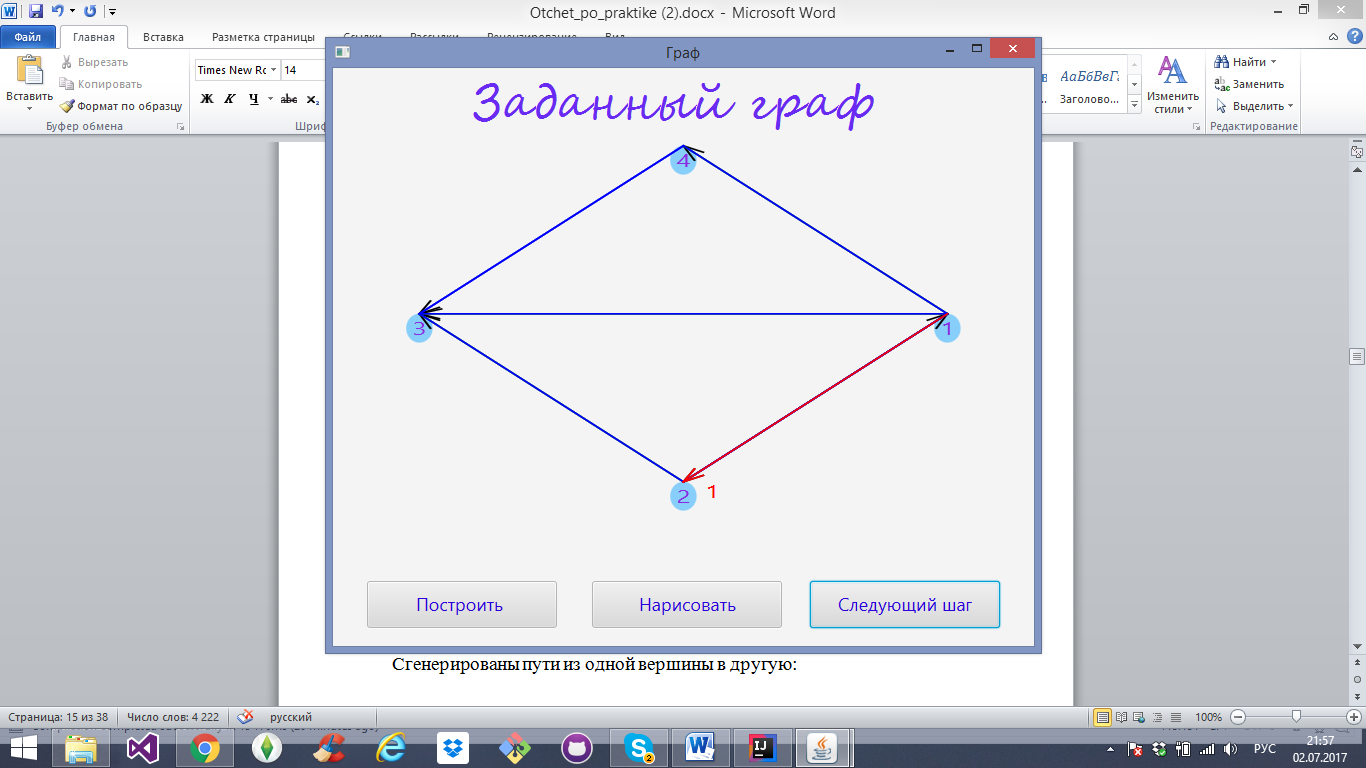
Первое ребро (1-2) обработано и произошла релаксация, тк вершина 1 имеет вес 0, а вершина 2 вес ∞ (0<∞). Теперь вершина 2 имеет вес 0+1=1. 

Второе ребро (2-1) обработано и релаксации не произошло, тк вершина 2 имеет вес 1, а вершина 1 имеет вес 0 (1>0).



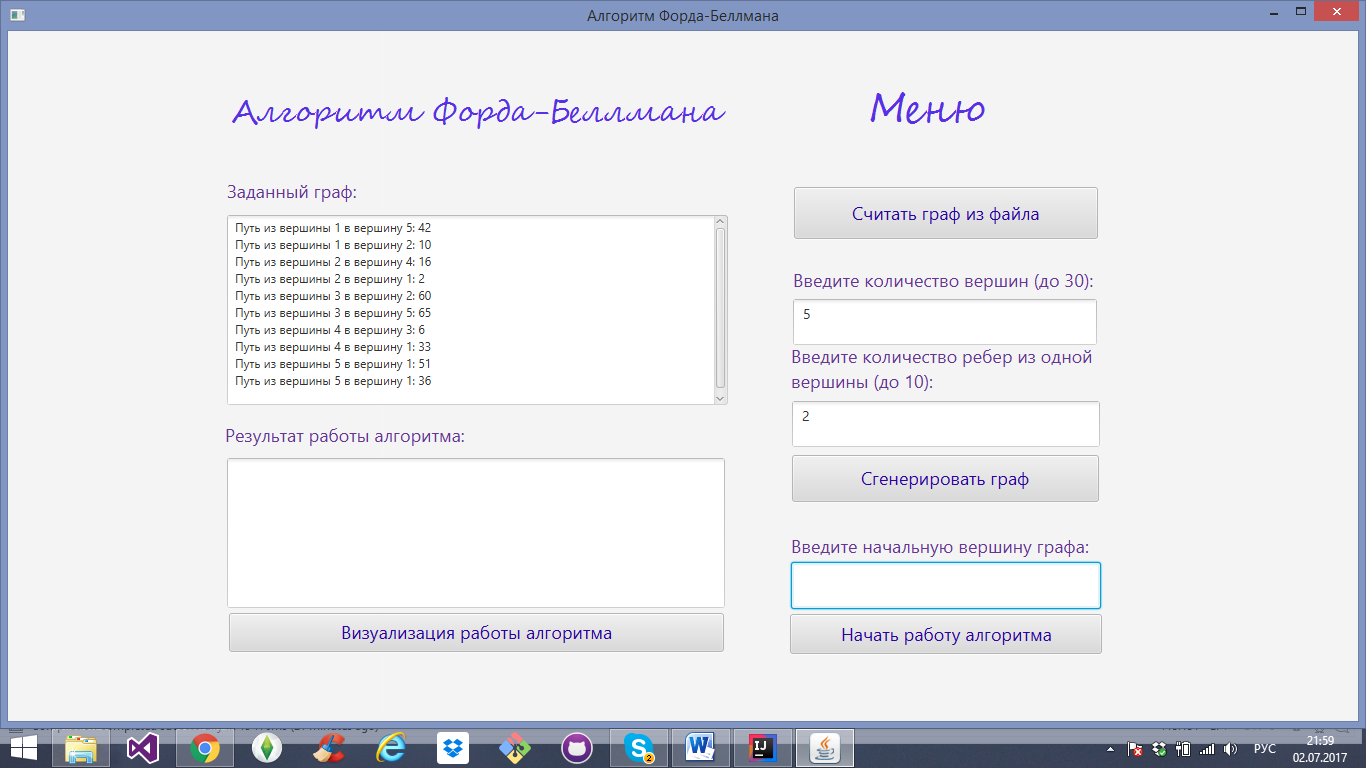
Следующие три шага – произведена релаксация. 

Следующие 6 шагов релаксации не произведено. 

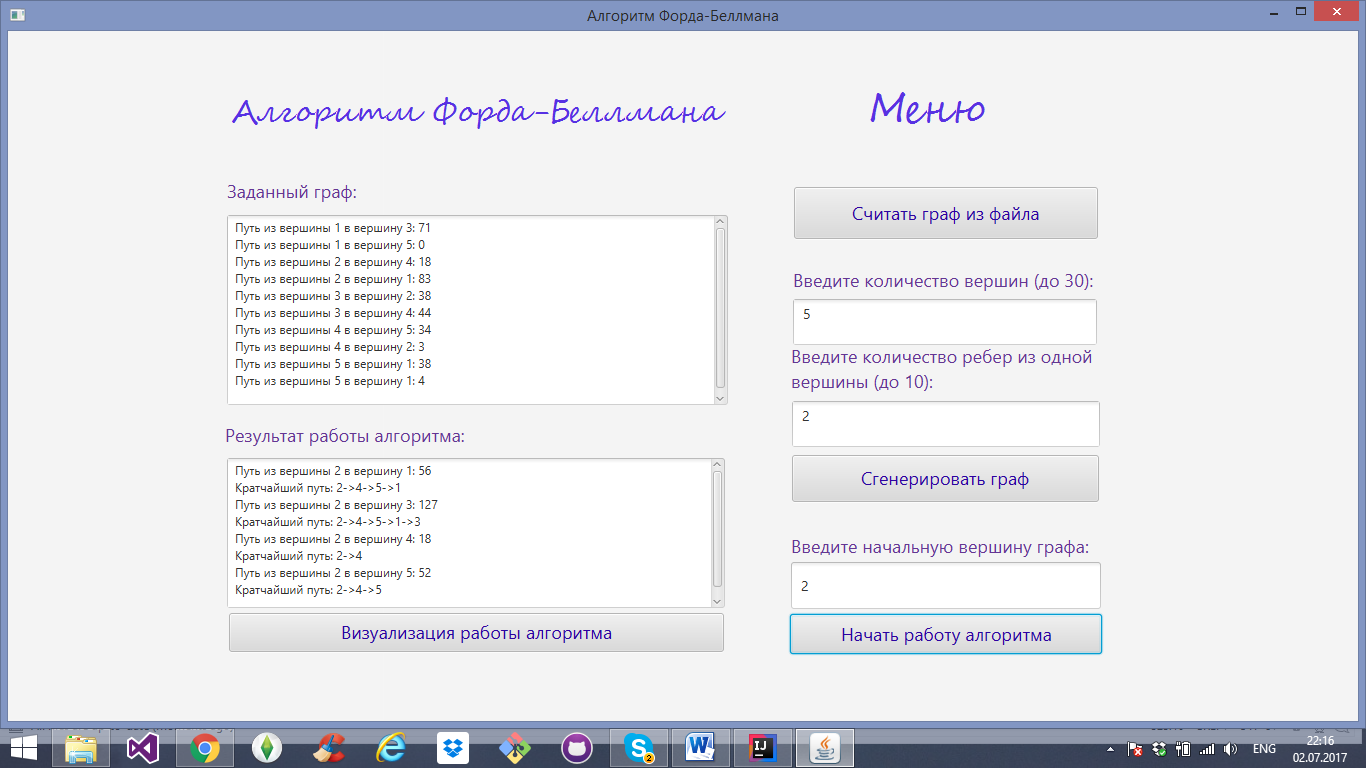
Самой последней релаксацией в вершину 2 была первая релаксация из вершины 1. Значит, кратчайший путь (1-2) в весом 1. 

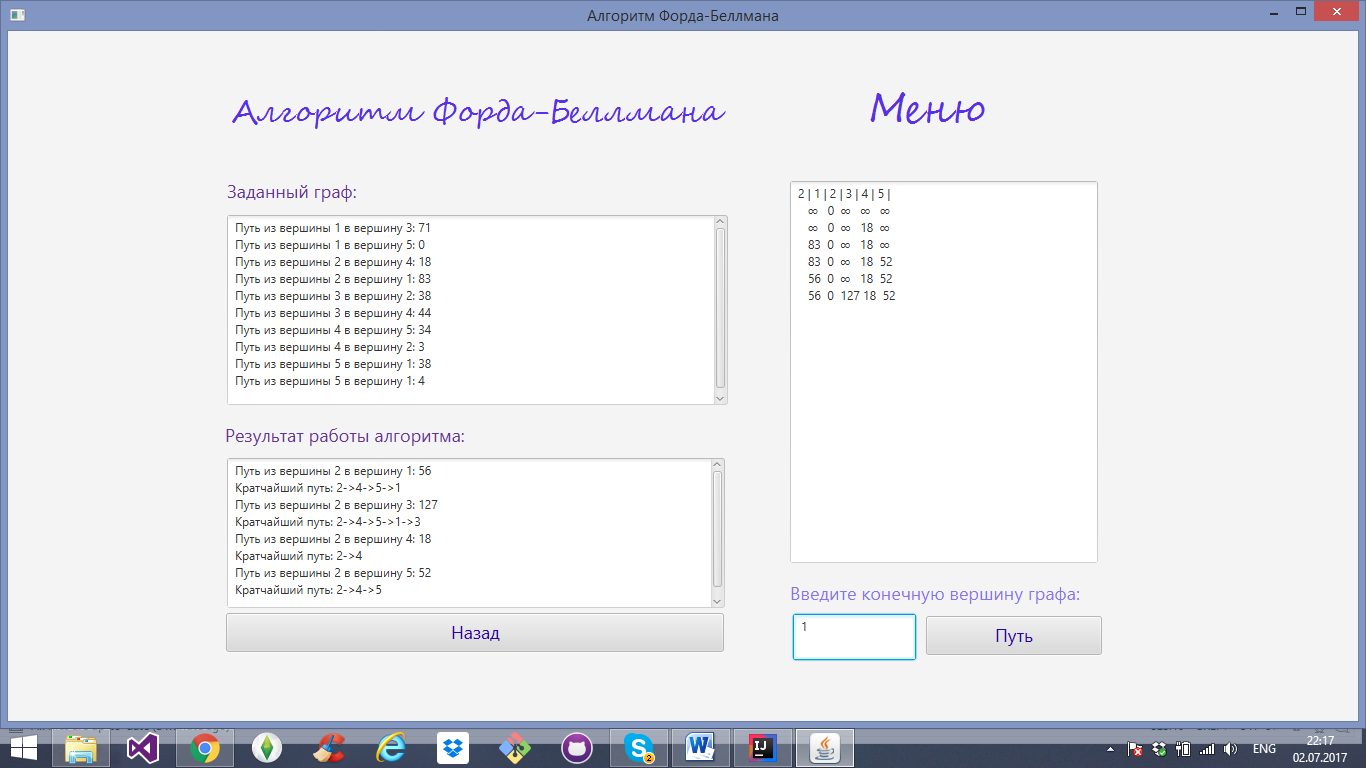
## 6.2. Тестирование работы программы с помощью случайной генерации.

Сгенерированы пути из одной вершины в другую:

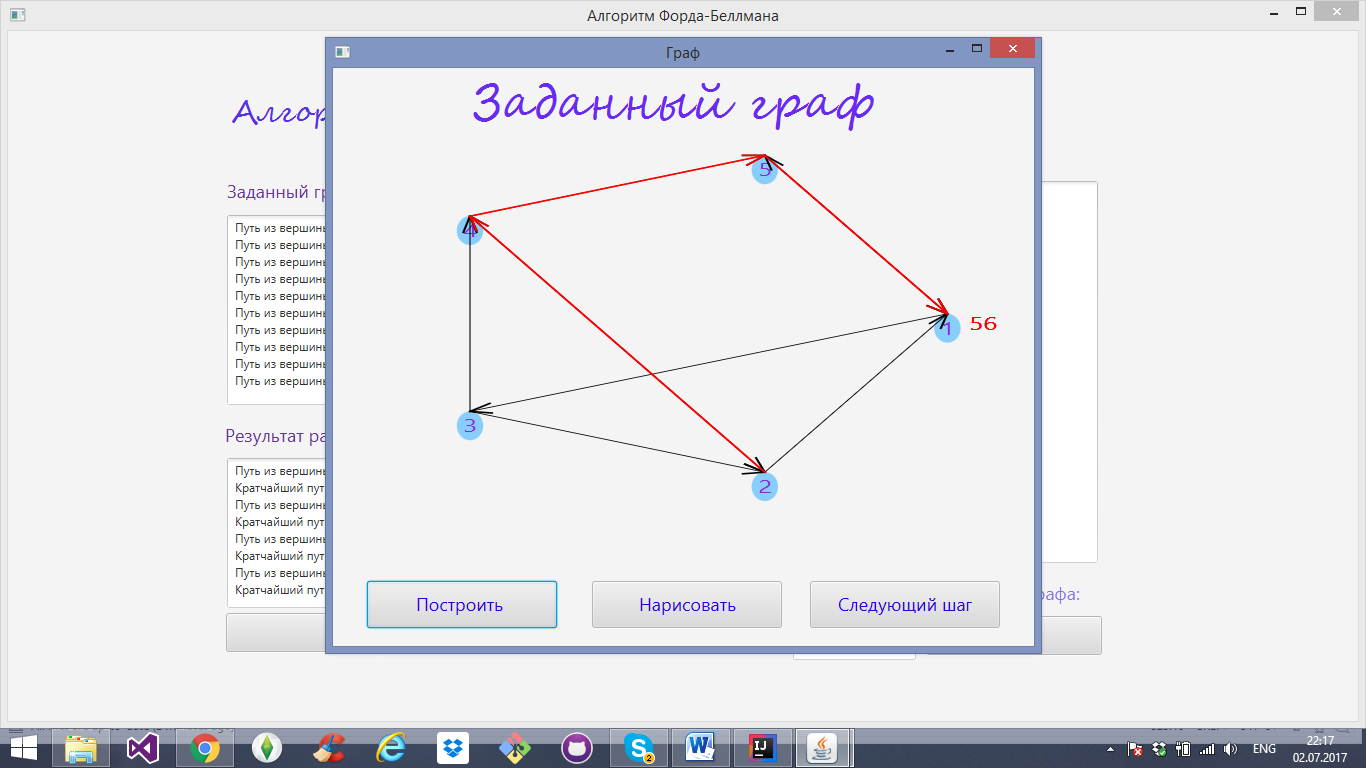


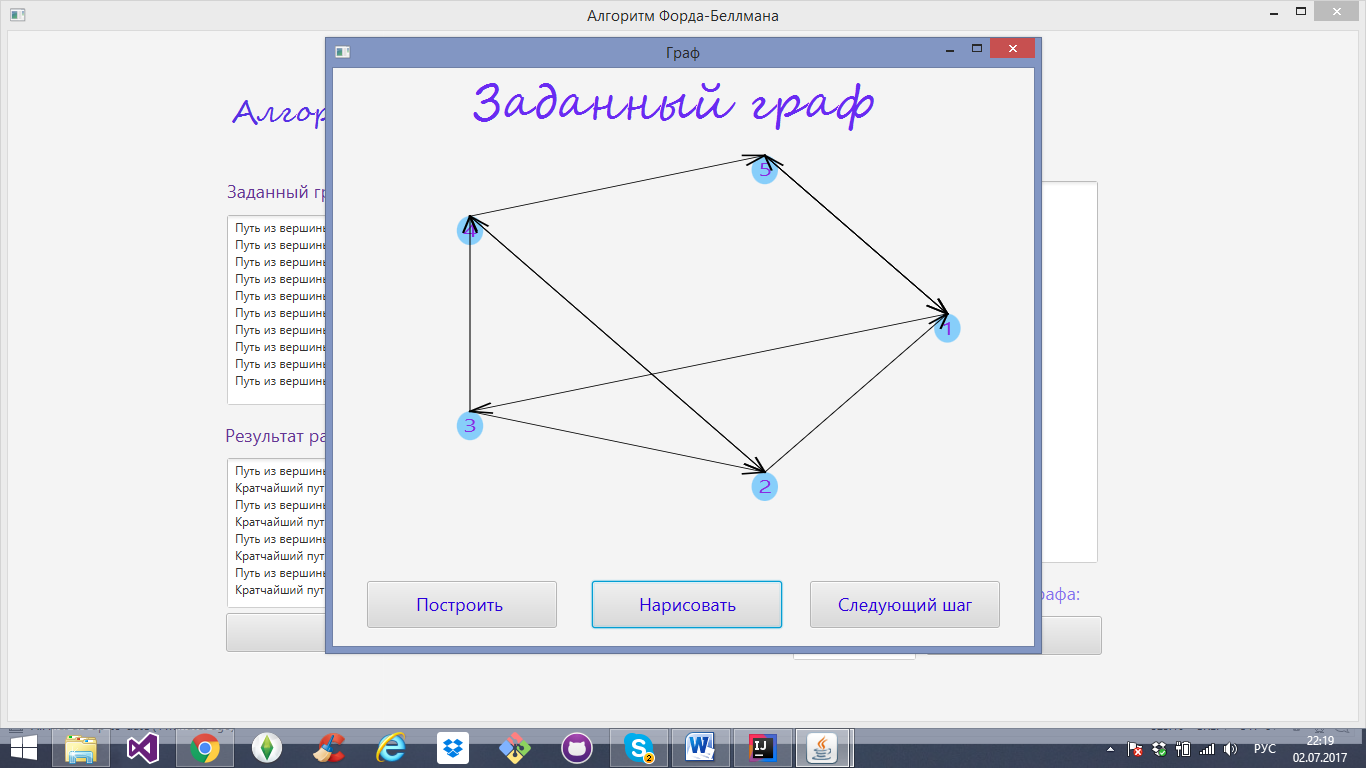
Пользователя просят ввести начальную вершину, например, 2. После нажатия кнопки «Начать работу алгоритма» и «Визуализация работы алгоритма» пользователь может увидеть результаты работы алгоритма. Затем пользователя просят ввести конечную вершину, например, 1.

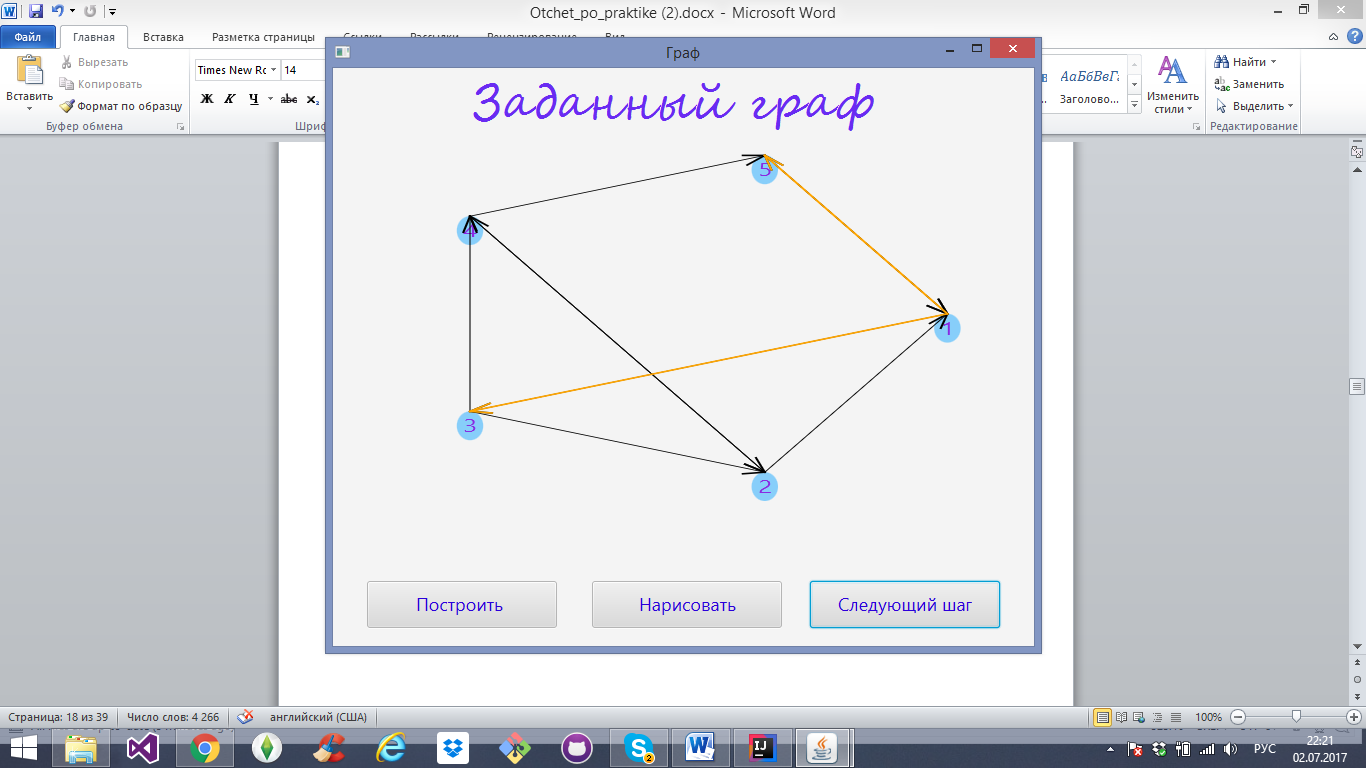




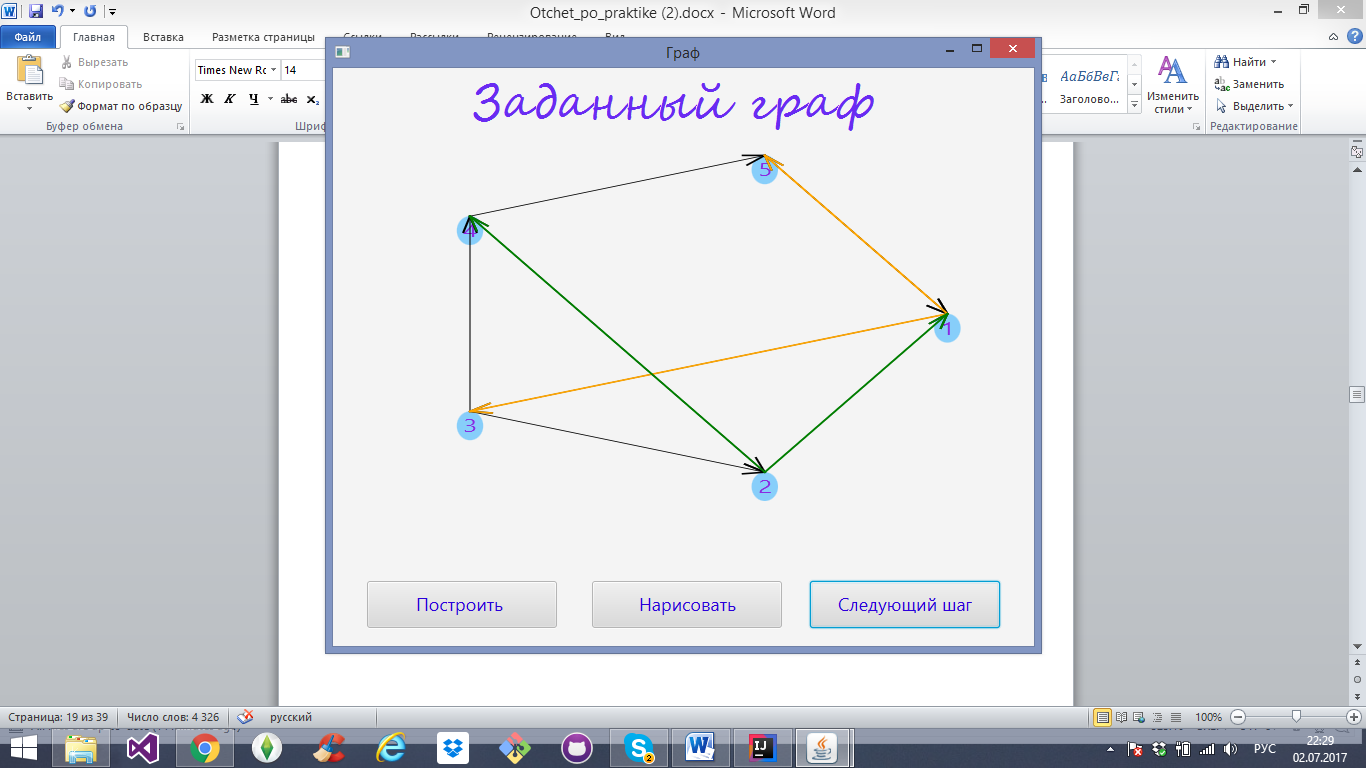
1)Вывод конечного результата.  
Кратчайший путь из вершины 2 до вершины 1 выделен красным цветом. А также рядом с вершиной подписана минимальная стоимость пути до неё.

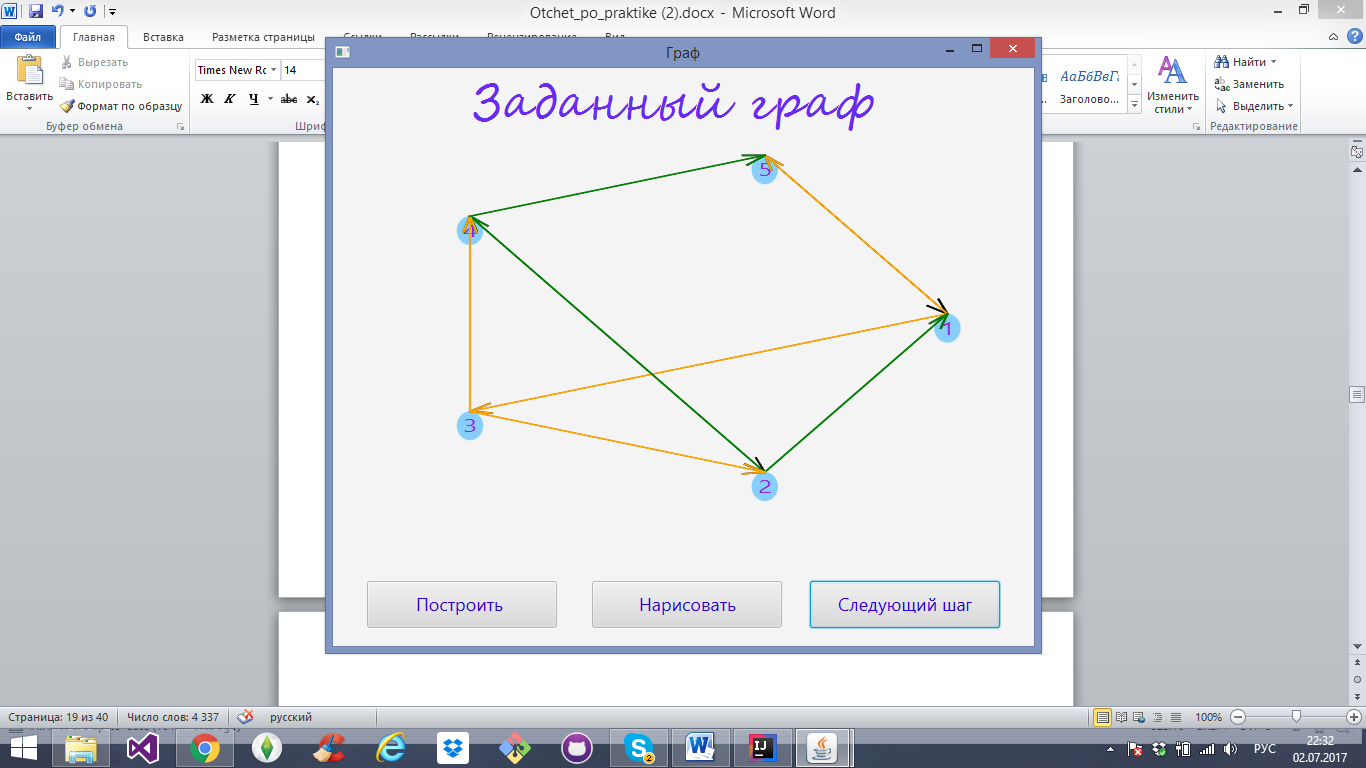


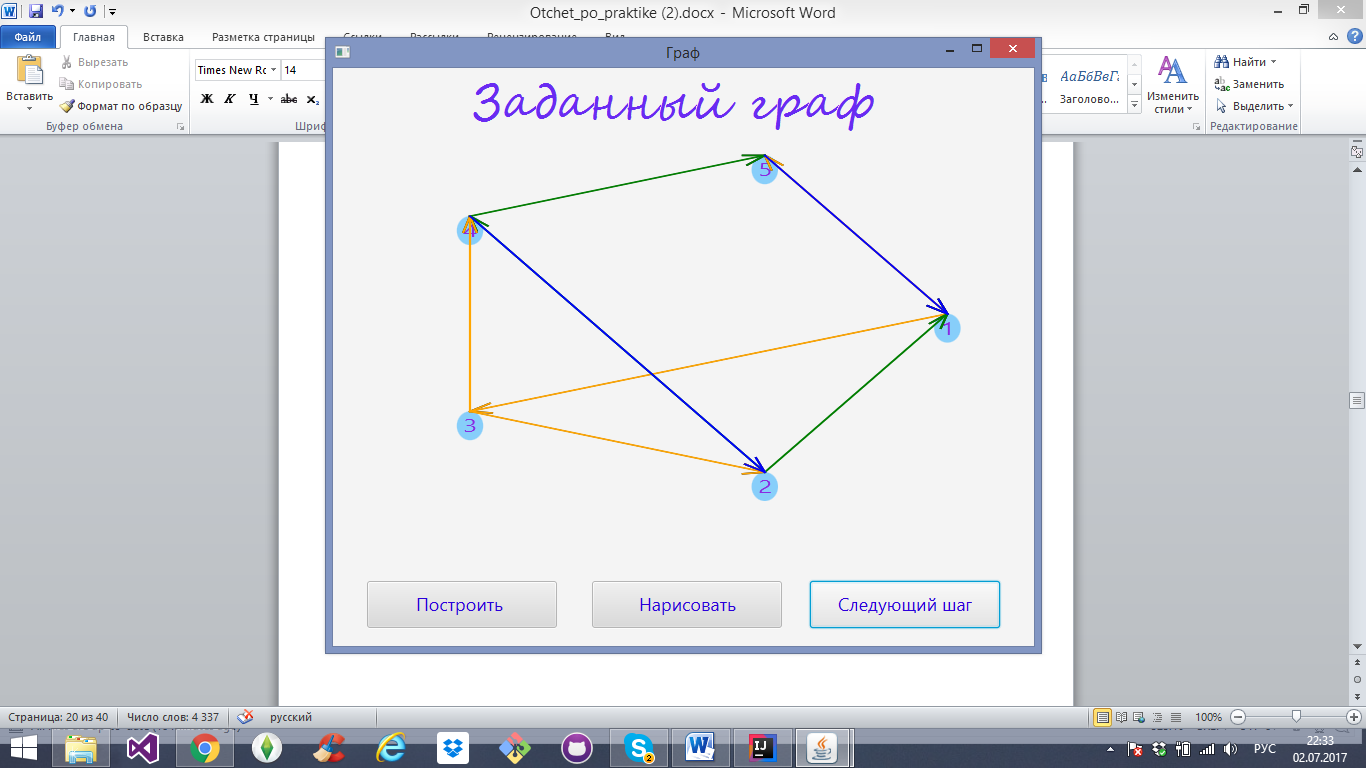
2)Пошаговое выполнение.  
При нажатии кнопки «Нарисовать», на экран выводится граф. 

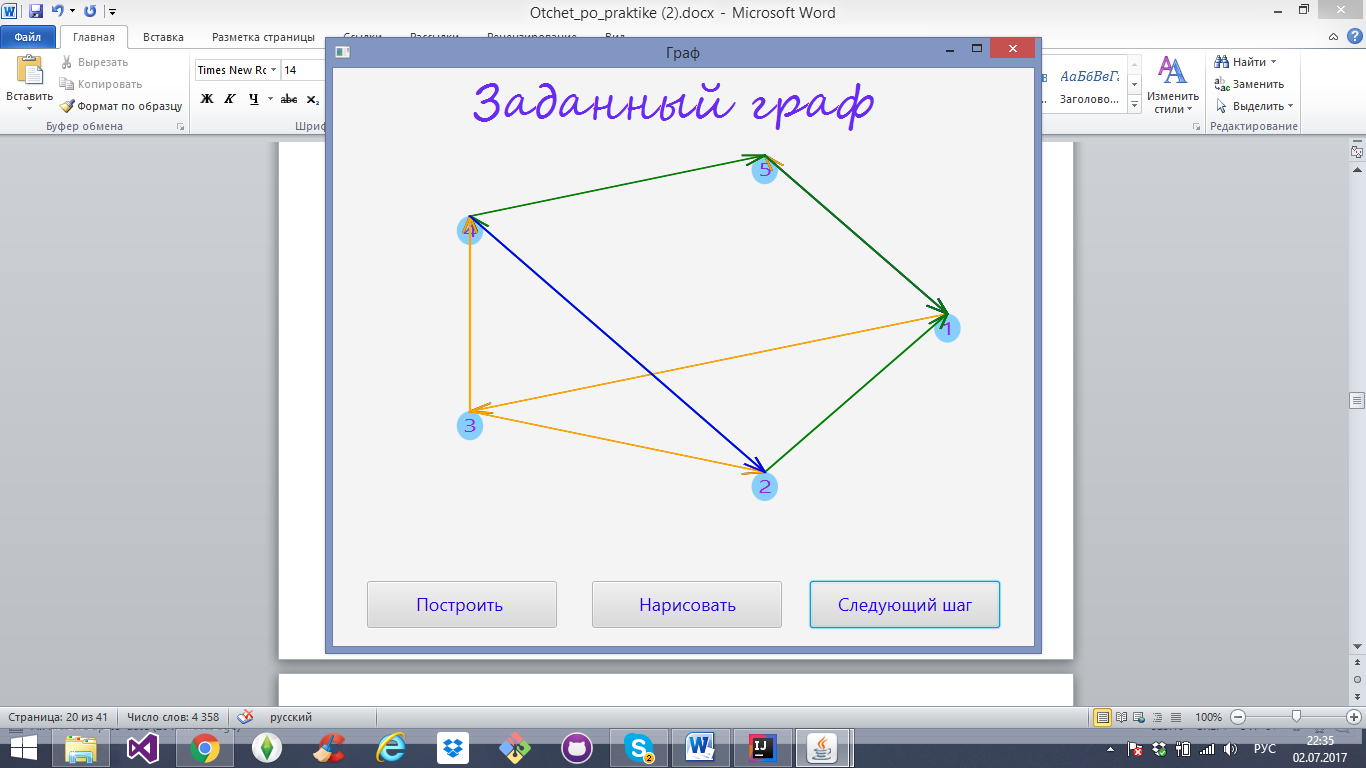
Первые два шага – обход рёбер, до которых пока не найдено пути от вершины 2.

На следующих двух шагах происходит релаксация из вершины 2. Это ребра (2-4) и (2-1). Вес этих рёбер равен соответственно 18 и 83. Т.е. если найдётся путь из 2 вершины до 1 через вершину 4, такой что он будет весить меньше, чем 83, то он будет кратчайшим.

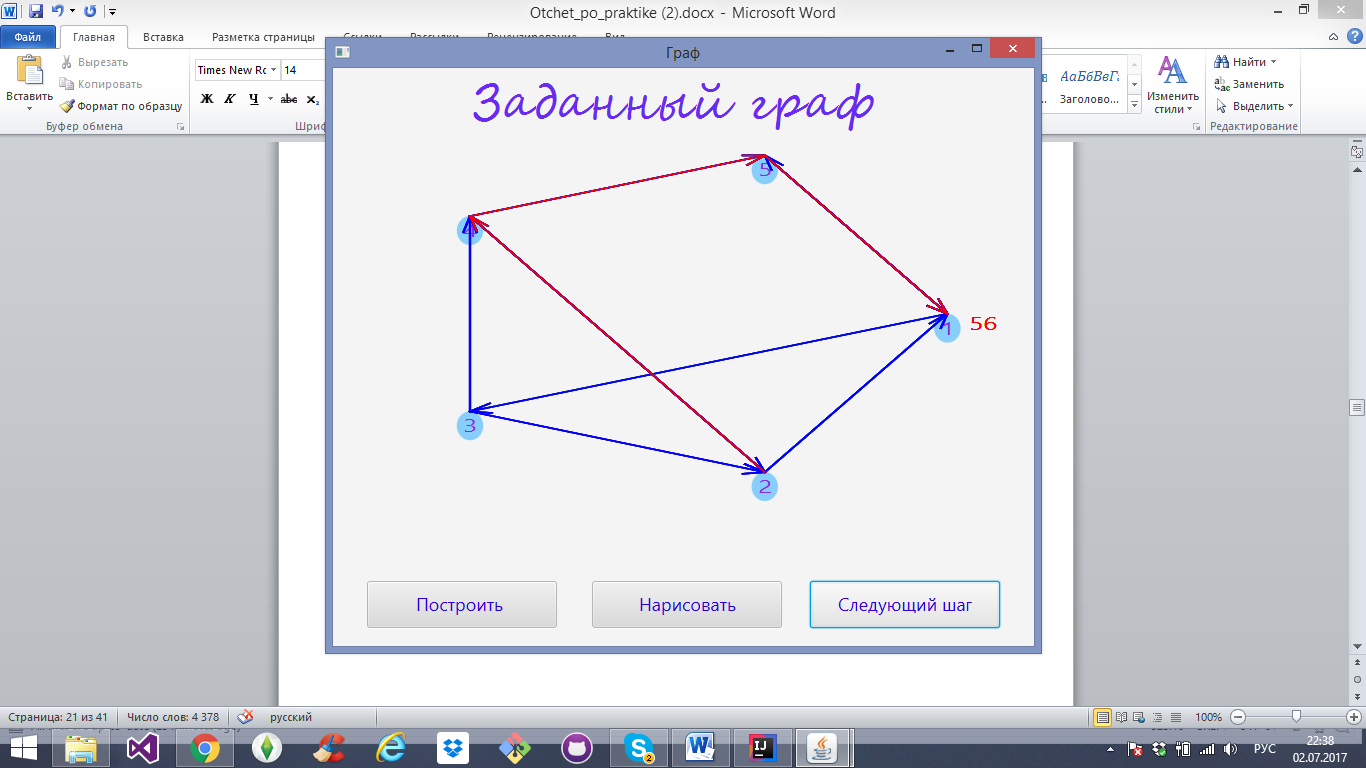


Через три шага мы видим, что произошла релаксация на ребре (4-5). 

На ребре (5-1) c весом 38 релаксации не произошло. 

Зато на следующем шаге на ребре (4-5) с весом 4 произошла релаксация. Т.е. последняя релаксация произошла на этом ребре, значит путь по всем релаксированным до него ребрам является кратчайшим. 

Это действительно так.



# 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

**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(**"Алгоритм Форда-Беллмана"**);  
 primaryStage.setScene(**new** Scene(root, 1350, 690));  
 primaryStage.show();  
 }  
  
  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
}

**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.scene.paint.Color;  
**import** javafx.scene.shape.Ellipse;  
**import** javafx.scene.shape.Line;  
**import** javafx.stage.Stage;  
**import** javafx.scene.layout.\*;  
**import** java.io.IOException;  
**import** java.util.Vector;  
  
  
*/\*\*  
 \* Класс для обработки событий элементов главного окна.  
 \*/***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 **labelFB**;  
 @FXML  
 **private** Label **text1**;  
 @FXML  
 **private** Label **text2**;  
 @FXML  
 **private** Label **text3**;  
 @FXML  
 **private** Label **text4**;  
 @FXML  
 **private** Label **text5**;  
 @FXML  
 **private** Label **text6**;  
 @FXML  
 **private** Label **menu**;  
 @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**;  
  
 */\*\*  
 \* Функция обработки события "Генерация случайного графа".  
 \* Проверяет правильное заполнение полей "Количество вершин" и "Количество ребер".  
 \* Обрабатывает нажатие клавиши "Сгенерировать граф".  
 \*/* @FXML  
 **public void** generateGraph() {  
 **if** (**amountVertex**.getText() == **null** || **amountVertex**.getText().length() == 0) {  
 error(**"Введите количество вершин"**);  
 **int** x = Integer.*parseInt*(**amountVertex**.getText());  
 } **else** {  
 **if** (**amountEdges**.getText() == **null** || **amountEdges**.getText().length() == 0) {  
 error(**"Введите количество ребер"**);  
 **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(**"Граф может быть не отображен."**);  
 } **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();*//Функция генерирования рандомного графа.* **for** (**int** i = 0; i < *P*.**list**.size(); i++) {  
 **textGraph**.appendText(**"Путь из вершины "** + (*P*.**list**.elementAt(i).**from** + 1) + **" в вершину "** + (*P*.**list**.elementAt(i).**to** + 1) + **": "** + *P*.**list**.elementAt(i).**l** + **"\n"**);  
 }  
 Stage stageWindow = **new** Stage();  
 */\*try {  
 FXMLDocumentController(stageWindow);  
 //graphWindow.arT(4);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }\*/* }  
 } **catch** (NumberFormatException ex) {  
 ex.printStackTrace();  
 error(**"Введено некорректное значение в одно из полей! Пожалуста, вводите только цифры."**);  
 }  
 }  
 }  
  
  
 */\*\*  
 \* Функция для вывода ошибок при заполнении полей.  
 \*  
 \** ***@param s*** *строка, которая сообщает, какая ошибка допущена при заполнении полей.  
 \*/* **public void** error(String s) {  
 **alert** = **new** Alert(Alert.AlertType.***ERROR***);  
 **alert**.setTitle(**"Некорректный ввод"**);  
 **alert**.setHeaderText(s);  
 **alert**.showAndWait();  
 }  
  
 */\*\*  
 \* Функция запуска алгоритма поиска гратчайших путей графа.  
 \*/* **public void** beginAlgorithm() {  
 *k* = 1;  
 *P*.searchAlgorithm(**this**);  
 *P*.outputWays(**this**);  
 }  
  
 */\*\*  
 \* Функция обработки нажатия клавиши "Начать работу алгоритма".  
 \*/* @FXML  
 **public void** workBegin() {  
 **if** (*P*.**n** == 0) {  
 error(**"Граф не сгенерирован"**);  
 } **else** {  
 **if** (**textField1**.getText() == **null** || **textField1**.getText().length() == 0) {  
 error(**"Введите вершину"**);  
 *//int m = Integer.parseInt(amountEdges.getText());  
 //P.v = m;* } **else** {  
 **try** {  
 **int** m = Integer.*parseInt*(**textField1**.getText());  
 *P*.**v** = m;  
 **if** (*P*.**v** > *P*.**n** || *P*.**v** <= 0)*//if (m > P.n || m <= 0)* error(**"Вершина задана неверно"**);  
 **else** beginAlgorithm();  
 } **catch** (NumberFormatException ex) {  
 ex.printStackTrace();  
 error(**"Введено некорректное значение начальной вершины! Пожалуста, вводите только цифры."**);  
 }  
 }  
 }  
 }  
  
  
 */\*\*  
 \* Функция обработки нажатия клавиши "Считать граф из файла".  
 \*  
 \** ***@throws*** *IOException  
 \*/* @FXML  
 **public void** fileGeneration () {  
 *k* = 0;  
 *P*.**list**.clear();  
 *P*.**ways**.clear();  
 *P*.**road**.clear();  
 *P*.**V**=-1;  
 **textGraph**.clear();  
 *P*.inputFile();*//Функция считывания графа из файла.* **for** (**int** i = 0; i < *P*.**list**.size(); i++) {  
 **textGraph**.appendText(**"Путь из вершины "** + (*P*.**list**.elementAt(i).**from** + 1) + **" в вершину "** + (*P*.**list**.elementAt(i).**to** + 1) + **": "** + *P*.**list**.elementAt(i).**l** + **"\n"**);  
 }  
 Stage stageWindow = **new** Stage();  
 */\*try {  
 FXMLDocumentController(stageWindow);  
 //graphWindow.arT(4);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }\*/* }  
  
 */\*\*  
 \* Функция вызова дочернего окна для отрисовки графа.  
 \*/* **public void** FXMLDocumentController(Stage stageWindow) **throws** IOException {  
 Parent root = FXMLLoader.*load*(getClass().getResource(**"GUI\_prototype.fxml"**));  
 stageWindow.setTitle(**"Граф"**);  
 Scene scene = **new** Scene(root);  
 stageWindow.setScene(scene);  
 stageWindow.show();  
 }  
  
 */\*\*  
 \* Функция отображения компонентов главной панели при нажатии клавиши "Визуализация работы алгоритма".  
 \*/* @FXML  
 **public void** nextMenu() {  
 **if** (*k* == 0)  
 error(**"Алгоритм не начал свою работу"**);  
 **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**;  
 }  
 }  
  
 */\*\*  
 \* Функция возврата в пердыдущее меню при визуализаци алгоритма.  
 \*/* @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**;  
 }  
  
 */\*\*  
 \* Функция вывода кратчайшего пути до заданной вершины.  
 \*/* @FXML  
 **public void** getWay() {  
 **if** (**graphV**.getText() == **null** || **graphV**.getText().length() == 0) {  
 error(**"Введите количество вершин"**);  
 *//int x = Integer.parseInt(graphV.getText());* } **else** {  
  
 **try** {  
 **int** x = Integer.*parseInt*(**graphV**.getText());  
 x--;  
 *P*.**V** = x;  
 **if** (x + 1 > *P*.**n** || x + 1 < 0 || x == (*P*.**v**)) {*//?* error(**"Граф может быть не отображен."**);  
 } **else** {  
 Stage stageWindow = **new** Stage();  
 **try** {  
 FXMLDocumentController(stageWindow);  
 *//graphWindow.arT(4);* } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 } **catch** (NumberFormatException ex) {  
 ex.printStackTrace();  
 error(**"Введено некорректное значение конечной вершины! Пожалуста, вводите только цифры."**);  
 }  
 }  
 }  
}

**Файл Graph.java**

**package** sample;  
  
*/\*\*  
 \* Created by helladmin on 26.06.2017.  
 \*/* **import** java.io.File;  
 **import** java.io.FileNotFoundException;  
 **import** java.util.Random;  
 **import** java.util.Scanner;  
 **import** java.util.Vector;  
  
*/\*\*  
 \* Класс для хранения графа и отображения действий над ним.  
 \*/***public class** Graph **implements** GraphInterface {  
 */\*\*  
 \* Класс для хранения ребер графа.  
 \*/* **public class** ElementGraphWay {*//переименовать не через нижнее подчеркивание* **int from**;  
 **int to**;  
 **int l**;*//weight* }  
  
 */\*\*  
 \* Класс для хранения координат узлов графа.  
 \*/* **public class** WaysPoint {  
 **int x**;  
 **int y**;  
 **int name**;*// с маленькой* }  
  
 **final int inf** = 1000000000;  
 **int x** = 130;  
 **int y** = 105;  
 **int vertexX** = 120;  
 **int vertexY** = 105;  
 **final** Random **random** = **new** Random();  
 Vector<ElementGraphWay> **list** = **new** Vector<ElementGraphWay>();  
 Vector<Integer> **ways** = **new** Vector<Integer>();  
 Vector<Integer> **road** = **new** Vector<Integer>();  
 Vector<WaysPoint> **vicual** = **new** Vector<WaysPoint>();  
 **int n**;*//количество узлов* **int m** = 0;*//количество ветвей* **int v**;*// узел из которого нужно считать пути;* **int V** = -1;  
 **int negativeCircle** = 0;  
  
 */\*\*  
 \* Функция считывания графа из файла.  
 \*/* **public void** inputFile() {  
 **negativeCircle** = 0;  
 **list**.clear();  
 **vicual**.clear();  
 Scanner sc;  
 **try** {  
 sc = **new** Scanner(**new** File(**"D://in.txt"**));*// создаём объект класса Scanner* **try** {  
 **if** (sc.hasNextInt()) { *// возвращает истинну если с потока ввода можно считать целое число* **n** = sc.nextInt();*// считывает целое число с потока ввода и сохраняем в переменную* } **else** System.***out***.println(**"В файле недостаточно данных"**);  
 **if** (sc.hasNextInt()) {  
 **m** = sc.nextInt();  
 } **else** System.***out***.println(**"В файле недостаточно данных"**);  
 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(**"Файл пуст!"**);  
 }  
 } **catch** (FileNotFoundException ex) {  
 System.***out***.println(**"Файл не существует!"**);  
 }  
  
 }  
  
 */\*\*  
 \* Функция генерирования рандомного графа.  
 \*/* **public void** inputGeneration() {  
 **negativeCircle** = 0;  
 **list**.clear();  
 **vicual**.clear();  
 **for** (**int** i = 0; i < **n**; i++) {  
 **for** (**int** j = 0; j < **m**; j++) {  
 ElementGraphWay Q = **new** ElementGraphWay();  
 Q.**from** = i;  
 **int** q;*// путь куда* **do** {  
 q = **random**.nextInt(**n**) + 1;  
 --q;  
 } **while** (q == i);  
 Q.**to** = q;  
 **int** a = **random**.nextInt(100); *//вес* Q.**l** = a;  
 **list**.add(Q);  
 }  
 }  
 **m** = **n** \* **m**;  
 XY();  
 }  
  
 */\*\*  
 \* Функция поиска кратчайших путей из заданной вершины в графе.  
 \*  
 \** ***@param P*** *контроллер, в который будут выведены результаты.  
 \*/* **public void** searchAlgorithm(Controller P) {  
 **for** (**int** i = 0; i < **n**; i++) {*//int n;//количество узлов* **ways**.add(**inf**);  
 **road**.add(-1);  
 }  
 **v**--;*// узел из которого нужно считать пути;* **ways**.set(**v**, 0);  
 P.**graphAlg**.clear();  
 P.**graphAlg**.appendText((**v** + 1) + **" |"**);  
 **for** (**int** k = 1; k <= **n**; k++) {  
 P.**graphAlg**.appendText(**" "** + k + **" |"**);  
 *//if (k > 9) {  
 //P.graphAlg.appendText(" " + k + " |");  
 //} else P.graphAlg.appendText(" " + k + " |");* }  
 P.**graphAlg**.appendText(**"\n"**);  
 P.**graphAlg**.appendText(**" "**);  
 **for** (**int** k = 0; k < **n**; k++) {  
 **if** (**ways**.elementAt(k) == **inf**)  
 P.**graphAlg**.appendText(**" ∞ "**);  
 **else** {  
 **if** (**ways**.elementAt(k) > 9)  
 P.**graphAlg**.appendText(**" "** + **ways**.elementAt(k) + **" "**);  
 **else** P.**graphAlg**.appendText(**" "** + **ways**.elementAt(k) + **" "**);  
 }  
 }  
 P.**graphAlg**.appendText(**"\n"**);  
 **for** (**int** i = 1; i <= **n**; ++i) {*//number of vertexes* **for** (**int** j = 0; j < **m**; ++j) {*//int m = 0;//количество ветвей* **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(**"Граф имеет отрицательные циклы"**);  
 P.**result**.appendText(**"Граф имеет отрицательные циклы"**);  
 **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(**" "**);  
 **for** (**int** k = 0; k < **n**; k++) {  
 **if** (**ways**.elementAt(k) == **inf**)  
 P.**graphAlg**.appendText(**" ∞ "**);  
 **else** {  
 **if** (**ways**.elementAt(k) > 99)  
 P.**graphAlg**.appendText(**" "** + **ways**.elementAt(k) + **""**);  
 **else** P.**graphAlg**.appendText(**" "** + **ways**.elementAt(k) + **" "**);  
 }  
 }  
 P.**graphAlg**.appendText(**"\n"**);  
  
 }  
 }  
 }  
  
 }  
 }  
  
  
  
 */\*\*  
 \* Функция вывода кратчайших путей из заданной вершины в графе.  
 \*  
 \** ***@param P*** *контроллер, в который будут выведены результаты.  
 \*/* **public void** outputWays(Controller P) {  
 **if** (**negativeCircle** == 0) {  
 P.**result**.clear();  
 Vector<Integer> path = **new** Vector<Integer>();  
 **for** (**int** j = 0; j < **n**; j++) {  
 **if** (j != (**v**)) {  
 **if** (**ways**.elementAt(j) == 1000000000) {  
 P.**result**.appendText(**"Путь из вершины "** + (**v** + 1) + **" в вершину "** + (j + 1) + **": NO\n"**);  
 } **else** {  
 path.clear();  
 **for** (**int** cur = j; cur != -1; cur = **road**.elementAt(cur))  
 path.add(cur);  
 P.**result**.appendText(**"Путь из вершины "** + (**v** + 1) + **" в вершину "** + (j + 1) + **": "** + **ways**.elementAt(j) + **"\nКратчайший путь: "**);  
 **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();  
 }  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* Функция присвоения координат узлам графа.  
 \*/* **public void** XY() {  
 **double** fi = 360 / **n**;  
 **for** (**int** i = 0; i < **n**; i++) {  
 WaysPoint q = **new** WaysPoint();  
 q.**x** = **x** + (**int**) (**long**) (**vertexX** \* Math.*cos*(i \* fi \* Math.***PI*** / 180));  
 q.**y** = **y** + (**int**) (**long**) (**vertexY** \* Math.*sin*(i \* fi \* Math.***PI*** / 180));  
 q.**name** = i + 1;  
 **vicual**.add(q);  
 }  
 }  
  
}

**Файл GraphController.java**

**package** sample;  
  
*/\*\*  
 \* Created by helladmin on 26.06.2017.  
 \*/* **import** javafx.event.ActionEvent;  
 **import** javafx.fxml.FXML;  
 **import** javafx.scene.Scene;  
 **import** javafx.scene.control.Alert;  
 **import** javafx.scene.control.Button;  
 **import** javafx.scene.control.Label;  
 **import** javafx.scene.layout.Pane;  
 **import** javafx.scene.paint.Color;  
 **import** javafx.scene.shape.Circle;  
 **import** javafx.scene.shape.Ellipse;  
 **import** javafx.scene.shape.Line;  
  
 **import** java.awt.geom.Line2D;  
 **import** java.io.IOException;  
 **import** java.util.ArrayList;  
 **import** java.util.Vector;  
  
*/\*\*  
 \* Класс для управления элементами дочерней панели.  
 \*/***public class** GraphController **extends** Controller {  
 **private** Alert **alert**;  
 **public int counter** = 0;  
 @FXML  
 **private** Pane **pane1**;  
 **static int** *Setting* = 1;  
 **public boolean fordInWork** = **false**;  
 **public boolean nextStep** = **false**;  
 **public int relaxCounter** = 0;  
 **public boolean last** = **false**;  
 **public int cycleCounter** = 0;  
 **public boolean cycle** = **false**;  
 **final int inf** = 1000000000;  
  
 @FXML  
 **public static** Button *nextButton*;  
  
 @FXML  
 **public static** Button *nextButton1*;  
  
 @FXML  
 **public static** Button *buildBut*;  
  
  
  
 */\*\*  
 \* Обработка нажатия клавиши "Построить".  
 \* Изображает заданный граф.  
 \*/* @FXML  
 **public void** graphButton() {  
 *//изображение вершин* **for** (**int** i = 0; i < *P*.**n**; i++) {  
 Ellipse C = **new** Ellipse(*P*.**vicual**.elementAt(i).**x**, *P*.**vicual**.elementAt(i).**y**, 6, 9);  
 C.setFill(Color.***LIGHTSKYBLUE***);  
 **pane1**.getChildren().add(C);  
 Label label = **new** Label(Integer.*toString*(*P*.**vicual**.elementAt(i).**name**));  
 label.setTextFill(Color.***BLUEVIOLET***);  
 label.setLayoutX(*P*.**vicual**.elementAt(i).**x** - 3);  
 label.setLayoutY(*P*.**vicual**.elementAt(i).**y** - 9);  
 **pane1**.getChildren().add(label);  
  
 }  
 *//изображение рёбер* **for** (**int** i = 0; i < *P*.**m**; i++) {  
 Line q = **new** Line(*P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**from**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**from**).**y** - 9, *P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**to**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**to**).**y** - 9);  
 q.setStrokeWidth(0.5);  
 q.setFill(Color.***LIGHTGREY***);  
 **pane1**.getChildren().add(q);  
  
 *//изображение стрелок* **this**.arrow(*P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**from**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**to**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**from**).**y**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**to**).**y**, Color.***BLACK***);  
 }  
 *//красит в красный кратчайший путь* **if** (*P*.**V** != -1) {  
 **if** (*P*.**ways**.elementAt(*P*.**V**) == 1000000000) {  
  
 Label label = **new** Label(**"Путь из вершины "** + Integer.*toString*(*P*.**v** + 1) + **" в вершину "** + Integer.*toString*(*P*.**V** + 1) + **": NO\n"**);  
 label.setTextFill(Color.***RED***);  
 label.setLayoutX(0);  
 label.setLayoutY(0);  
 **pane1**.getChildren().add(label);  
 } **else** {  
  
 Vector<Integer> path = **new** Vector<Integer>();  
 **for** (**int** cur = *P*.**V**; cur != -1; cur = *P*.**road**.elementAt(cur))  
 path.add(cur);  
  
 **for** (**int** i = path.size() - 1; i >= 1; i--) {  
  
 **int** l = (path.elementAt(i));  
 **int** k = (path.elementAt(i - 1));  
  
 Label label1 = **new** Label(Integer.*toString*(*P*.**ways**.elementAt(*P*.**V**)));  
 label1.setTextFill(Color.***RED***);  
 label1.setLayoutX((*P*.**vicual**.elementAt(*P*.**V**)).**x** + 10);  
 label1.setLayoutY((*P*.**vicual**.elementAt(*P*.**V**).**y** - 12));  
 **pane1**.getChildren().add(label1);  
  
 Line q = **new** Line(*P*.**vicual**.elementAt(l).**x**, *P*.**vicual**.elementAt(l).**y** - 9, *P*.**vicual**.elementAt(k).**x**, *P*.**vicual**.elementAt(k).**y** - 9);  
 q.setStroke(Color.***RED***);  
 q.setStrokeWidth(1);  
 **pane1**.getChildren().add(q);  
  
  
 **this**.arrow(*P*.**vicual**.elementAt(l).**x**, *P*.**vicual**.elementAt(k).**x**, *P*.**vicual**.elementAt(l).**y**, *P*.**vicual**.elementAt(k).**y**, Color.***RED***);  
 }  
 path.clear();  
 }  
 }  
  
 }  
  
 */\*\*  
 \* Функция отрисовки направления движения по ребру.  
 \*  
 \** ***@param x*** *координаты  
 \** ***@param x1*** *координаты  
 \** ***@param y*** *координаты  
 \** ***@param y1*** *координаты  
 \** ***@param color*** *цвет линии.  
 \*/* **void** arrow(**int** x, **int** x1, **int** y, **int** y1, Color color) {  
 **double** beta = Math.*atan2*((y) - (y1 - 9), x1 - x); *//{ArcTan2 ищет арктангенс от x/y что бы неопределенностей не  
 // возникало(например, деление на 0)}* **double** alfa = Math.***PI*** / 10;*// {угол между основной осью стрелки и рисочки в конце}* **int** r1 = 10; *//{длинна риски}* **int** x2 = (**int**) Math.*round*(x1 - r1 \* Math.*cos*(beta + alfa));  
 **int** y2 = (**int**) Math.*round*((y1 - 9) + r1 \* Math.*sin*(beta + alfa));  
*//g2d.drawLine(x1,y1,x2,y2);* **int** x3 = (**int**) Math.*round*(x1 - r1 \* Math.*cos*(beta - alfa));  
 **int** y3 = (**int**) Math.*round*((y1 - 9) + r1 \* Math.*sin*(beta - alfa));  
*//g2d.drawLine(x1,y1,x2,y2);* Line q1 = **new** Line(x1, y1 - 9, x2, y2);  
 Line q2 = **new** Line(x1, y1 - 9, x3, y3);  
 q1.setStroke(color);  
 **pane1**.getChildren().add(q1);  
 q2.setStroke(color);  
 **pane1**.getChildren().add(q2);  
 }  
  
 */\*\*  
 \* Функция отрисовки графа.  
 \* Обработака нажатия кнопки " Нарисовать".  
 \*/* @FXML  
 **public void** graphPaint() {  
 **fordInWork** = **true**;  
  
 *//изображение вершин* **for** (**int** i = 0; i < *P*.**n**; i++) {  
 Ellipse C = **new** Ellipse(*P*.**vicual**.elementAt(i).**x**, *P*.**vicual**.elementAt(i).**y**, 6, 9);  
 C.setFill(Color.***LIGHTSKYBLUE***);  
 **pane1**.getChildren().add(C);  
 Label label = **new** Label(Integer.*toString*(*P*.**vicual**.elementAt(i).**name**));  
 label.setTextFill(Color.***BLUEVIOLET***);  
 label.setLayoutX(*P*.**vicual**.elementAt(i).**x** - 3);  
 label.setLayoutY(*P*.**vicual**.elementAt(i).**y** - 9);  
 **pane1**.getChildren().add(label);  
 }  
 *//изображение рёбер* **for** (**int** i = 0; i < *P*.**m**; i++) {  
 Line q = **new** Line(*P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**from**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**from**).**y** - 9, *P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**to**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**to**).**y** - 9);  
 q.setStrokeWidth(0.5);  
 q.setFill(Color.***BLACK***);  
 **pane1**.getChildren().add(q);  
  
 *//изображение стрелок* **this**.arrow(*P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**from**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**to**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**from**).**y**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(i).**to**).**y**, Color.***BLACK***);  
 }  
  
 **for** (**int** i = 1; i < (*P*.**n**+1); i++) {*//int n;//количество узлов  
 P*.**ways**.remove(1);*// was i like index  
 P*.**ways**.add(**inf**);  
 *P*.**road**.add(-1);  
 }  
  
 *//P.v--;// узел из которого нужно считать пути;  
 //P.V--;//узел в который ищем путь  
 P*.**ways**.set(*P*.**v**, 0);  
 *P*.**ways**.set(*P*.**V**, **inf**);  
 *// nextButton.setDisable(false);* }  
  
 */\*\*  
 \* Пошаговый поиск кратчайших путей из заданной вершины в графе.  
 \* Обработка нажатия клавиши "следующий шаг".  
 \*/* @FXML  
 **public void** nextStep(ActionEvent actionEvent) {  
 **if** (**fordInWork**) {  
 **nextStep** = **true**;  
  
 cycleFord();  
 **if** (**last**) {  
 **if** (*P*.**ways**.elementAt(*P*.**V**) == **inf**) {  
 Label label6 = **new** Label(**"Путь из вершины "** + Integer.*toString*(*P*.**v** + 1) + **" в вершину "** + Integer.*toString*(*P*.**V** + 1) + **": NO\n"**);  
 label6.setTextFill(Color.***ORANGE***);  
 label6.setLayoutX(0);  
 label6.setLayoutY(0);  
 **pane1**.getChildren().add(label6);  
 **cycle** = **false**;  
 }  
 Vector<Integer> path = **new** Vector<Integer>();  
 **if** (*P*.**V** != -1) {  
 **for** (**int** cur = *P*.**V**; cur != -1; cur = *P*.**road**.elementAt(cur)) {  
 path.add(cur);  
 }  
  
 **for** (**int** j = path.size() - 1; j >= 1; j--) {  
  
 **int** l = (path.elementAt(j));  
 **int** k = (path.elementAt(j - 1));  
  
 Label label1 = **new** Label(Integer.*toString*(*P*.**ways**.elementAt(*P*.**V**)));  
 label1.setTextFill(Color.***RED***);  
 label1.setLayoutX((*P*.**vicual**.elementAt(*P*.**V**)).**x** + 10);  
 label1.setLayoutY((*P*.**vicual**.elementAt(*P*.**V**).**y** - 12));  
 **pane1**.getChildren().add(label1);  
  
  
 Line q = **new** Line(*P*.**vicual**.elementAt(l).**x**, *P*.**vicual**.elementAt(l).**y** - 9, *P*.**vicual**.elementAt(k).**x**, *P*.**vicual**.elementAt(k).**y** - 9);  
 q.setStroke(Color.***RED***);  
 q.setStrokeWidth(1);  
 **pane1**.getChildren().add(q);  
  
  
 **this**.arrow(*P*.**vicual**.elementAt(l).**x**, *P*.**vicual**.elementAt(k).**x**, *P*.**vicual**.elementAt(l).**y**, *P*.**vicual**.elementAt(k).**y**, Color.***RED***);  
 }  
 path.clear();  
 *//}* }  
 path.clear();  
 *P*.**ways**.clear();  
 *P*.**road**.clear();  
 **nextStep** = **false**;  
 **fordInWork** = **false**;  
 **last** = **false**;  
 **counter** = 0;  
  
 **cycleCounter** = 0;  
  
 }  
 }  
 }  
  
 **public void** cycleFord() {  
 **if** (**fordInWork** && !**cycle**) {  
 **if** (!**last**) {  
 **if** (**counter** < *P*.**m**)*//false!!!!!! error P.ways.size()* stepSearchAlgorithm();  
 **else** {  
 **relaxCounter** = 0;  
 **counter** = 0;  
 **cycleCounter**++;  
 **cycle** = **true**;  
 }  
 }  
 }  
 **if** (**fordInWork** && **cycle**) {  
 **last** = **true**;  
 **if** (**counter** < *P*.**m**) {  
 stepSearchAlgorithm();  
 **cycle** = **true**;  
 **last** = **false**;  
 } **else** {  
 **if** (**relaxCounter** != 0)  
 **last** = **false**;  
 **else  
 last** = **true**;  
 **relaxCounter** = 0;  
 **counter** = 0;  
 **cycleCounter**++;  
 **cycle** = **false**;  
 }  
 }  
 }  
  
 **public void** stepSearchAlgorithm() {  
  
 **if** (*P*.**ways**.elementAt(*P*.**list**.elementAt(**counter**).**from**) < **inf**) {*//j=counter* **if** ((*P*.**ways**.elementAt(*P*.**list**.elementAt(**counter**).**from**) + *P*.**list**.elementAt(**counter**).**l**) < *P*.**ways**.elementAt(*P*.**list**.elementAt(**counter**).**to**)) {*//Зеленый - произошло ослабление.  
 P*.**ways**.set(*P*.**list**.elementAt(**counter**).**to**, (*P*.**ways**.elementAt(*P*.**list**.elementAt(**counter**).**from**) + *P*.**list**.elementAt(**counter**).**l**));  
 *P*.**road**.set(*P*.**list**.elementAt(**counter**).**to**, *P*.**list**.elementAt(**counter**).**from**);  
  
 Line q3 = **new** Line(*P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**from**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**from**).**y** - 9, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**to**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**to**).**y** - 9);  
 q3.setStrokeWidth(1);  
 q3.setStroke(Color.***GREEN***);  
 **pane1**.getChildren().add(q3);  
 **this**.arrow(*P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**from**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**to**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**from**).**y**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**to**).**y**, Color.***GREEN***);  
 **last** = **false**;  
 **relaxCounter**++;  
 **cycle** = **false**;  
 } **else** {  
 Line q1 = **new** Line(*P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**from**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**from**).**y** - 9, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**to**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**to**).**y** - 9);  
 q1.setStrokeWidth(1);  
 q1.setStroke(Color.***BLUE***);  
 **pane1**.getChildren().add(q1);  
 **this**.arrow(*P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**from**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**to**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**from**).**y**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**to**).**y**, Color.***BLUE***);  
 **cycle** = **false**;  
 }  
 }  
 **else** {  
 Line q3 = **new** Line(*P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**from**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**from**).**y** - 9, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**to**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**to**).**y** - 9);  
 q3.setStrokeWidth(1);  
 q3.setStroke(Color.***ORANGE***);  
 **pane1**.getChildren().add(q3);  
 **this**.arrow(*P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**from**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**to**).**x**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**from**).**y**, *P*.**vicual**.elementAt(*P*.**list**.elementAt(**counter**).**to**).**y**, Color.***ORANGE***);  
 **cycle** = **false**;  
 }  
 **counter**++;  
 }  
 }

**Файл Graph\_Interface.java**

**package** sample;  
  
*/\*\*  
 \* Created by helladmin on 26.06.2017.  
 \*/  
/\*\*  
 \* Интерфейс для реализации графа.  
 \*/***public interface** GraphInterface {  
 **void** inputFile();  
  
 **void** inputGeneration();  
  
 **void** searchAlgorithm(Controller P);  
  
 **void** outputWays(Controller P);  
}

**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***?>*<**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"**>  
  
 </**SplitPane**>  
 <**Button fx:id="fileGraph" layoutX="588.0" layoutY="145.0" mnemonicParsing="false" onAction="#fileGeneration" prefHeight="52.0" prefWidth="304.0" text="Считать граф из файла" 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="Введите начальную вершину графа:" 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="Сгенерировать граф" 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="Начать работу алгоритма" 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=" Алгоритм Форда-Беллмана" textFill="#5731e4"**>  
 <**font**>  
 <**Font name="Segoe Script" size="30.0"** />  
 </**font**>  
 </**Label**>  
 <**Label fx:id="menu" layoutX="663.0" layoutY="27.0" prefHeight="72.0" prefWidth="230.0" text="Меню" textFill="#5832e3"**>  
 <**font**>  
 <**Font name="Segoe Script" size="40.0"** />  
 </**font**>  
 </**Label**>  
 <**Label fx:id="text3" layoutX="587.0" layoutY="221.0" prefHeight="35.0" prefWidth="305.0" text="Введите количество вершин (до 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="Введите количество ребер из одной" 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="вершины (до 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="Результат работы алгоритма:" 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="Заданный граф:" 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="Визуализация работы алгоритма" 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="Назад" 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="Введите конечную вершину графа:" 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="Путь" 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**>

**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***?>*<**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**>  
  
 <**Pane layoutX="-213.0" layoutY="-200.0" prefHeight="494.0" prefWidth="37.0"**>  
  
 </**Pane**>  
 <**Label layoutX="129.0" layoutY="-97.0" prefHeight="256.0" prefWidth="411.0" text="Заданный граф" textFill="#6a2bf2" AnchorPane.bottomAnchor="419.0" AnchorPane.leftAnchor="129.0" AnchorPane.rightAnchor="131.0" AnchorPane.topAnchor="-97.0"**>  
 <**font**>  
 <**Font name="Segoe Script" size="47.0"** />  
 </**font**>  
 </**Label**>  
 <**Button fx:id="buildBut" layoutX="24.0" layoutY="513.0" mnemonicParsing="false" onAction="#graphButton" prefHeight="47.0" prefWidth="190.0" text="Построить " 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="256.0" 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="nextButton" layoutX="467.0" layoutY="513.0" mnemonicParsing="false" onAction="#nextStep" prefHeight="47.0" prefWidth="190.0" text="Следующий шаг" 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="249.0" layoutY="513.0" mnemonicParsing="false" onAction="#graphPaint" prefHeight="47.0" prefWidth="190.0" text="Нарисовать" textFill="#2c00da"**>  
 <**font**>  
 <**Font size="18.0"** />  
 </**font**>  
 </**Button**>  
 </**children**>  
 </**AnchorPane**>  
 </**children**>  
</**GridPane**>