**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

Проект

**по практике «JAVA»**

Тема: Мосты графа. Визуализация алгоритма.

|  |  |  |
| --- | --- | --- |
| Студенты гр. 4383 | Рымарь М.И. Коптюг А.Д.  Алешин С.В. |  |
| Преподаватель | Шолохова О.М. |  |

Санкт-Петербург

2016

Содержание

[ **ИСХОДНАЯ ПОСТАНОВКА ЗАДАЧИ** 3](#_Toc455103016)

[ **АНАЛИЗ И ПРИМЕР РЕШЕНИЯ ЗАДАЧИ** 3](#_Toc455103017)

[ **ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ** 4](#_Toc455103018)

[ Исходные данные: 4](#_Toc455103019)

[ Ограничения на исходные данные: 5](#_Toc455103020)

[ Результирующие выходные данные: 5](#_Toc455103021)

[ **СПЕЦИФИКАЦИЯ ПРОГРАММЫ** 5](#_Toc455103022)

[ Исключительные ситуации в программе: 5](#_Toc455103023)

[ Выходные данные 5](#_Toc455103024)

[ Сценарий диалога 5](#_Toc455103025)

[ **РАЗРАБОТКА СТРУКТУР ДАННЫХ И АЛГОРИТМОВ** 6](#_Toc455103026)

[ **ТЕСТИРОВАНИЕ ПРОГРАММЫ** 9](#_Toc455103027)

[ **КОД ПРОГРАММЫ** 13](#_Toc455103028)

[ **ВЫВОД** 35](#_Toc455103029)

# **ИСХОДНАЯ ПОСТАНОВКА ЗАДАЧИ**

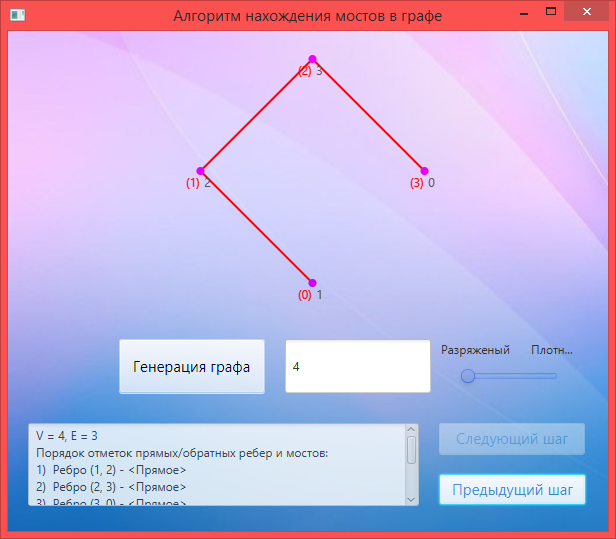
Мост в связном неориентированном графе - это ребро, удаление которого делает граф несвязным. Реализовать алгоритм нахождения мостов графа (модификацией поиска в глубину из задачи 2: ребро *{v,w}* графа будет мостом тогда и только тогда, когда *{v,w}* является ребром дерева и *Low(w)>=NumVert(w)).* Провести экспериментальный анализ его эффективности на графах разной структуры. Продумать и предложить механизм генерации необходимых для эксперимента тестовых данных.

# **АНАЛИЗ И ПРИМЕР РЕШЕНИЯ ЗАДАЧИ**

1. Существует несколько способов реализовать граф. Один из них – матрица смежности. Также существует и такая структура как список смежности. В данной работе был использован список смежности на базе вектора для реализации представления графа. Этот выбор обоснован тем, что в работе будут генерироваться разреженные графы, а при таких графах в матрицах инцидентности появляется много лишних нулей, которые лишь занимают память
2. Для работы требуется генерировать неориентированные связные графы. Генерация была реализована в два этапа. Первым этапом n вершин соединяются n-1 ребром, чтобы граф стал связным. Таким образом строится остовное дерево графа. Вторым этапом добавляются остальные ребра, если это необходимо при текущей генерации
3. После генерации графа требуется приступить к подготовке необходимых данных для выполнения алгоритма по поиску графов. Сначала алгоритмом ПВГ по графу строится дерево ПВГ, а также по пути отмечаются все обратные ребра, если такие имеются, и нумеруются вершины по порядку ПВГ-обхода.
4. Для каждой вершины vV определим множество P(v). P(v) = {v} U {w : (w = предок(v)) & ( xV: ((x = v) (x=потомок(v))) & (<x,w>B))},где B - множество обратных рёбер (хорд). Иными словами, P (v) состоит из v и всех предков вершины v, которых можно достичь из v, пройдя сначала несколько (возможно, ни одной) дуг дерева T, а затем одно обратное ребро и остановившись. Таким образом после построения дерева DFS и нахождения всех обратных ребер, будет вычислена функция P(v) для каждой вершины дерева.
5. После вычисления P(v), для каждой вершины будет найдена Low(v) = min { numVert(x) ⎢ x ∈ P (v) }. То есть эта функция, которая определяет наименьший номер вершины, который был получен функцией P (v) для этой вершины.
6. Далее необходимо для каждого ребра {v,w} дерева проверить соотношение Low(w)>=NumVert(w)). И если это условие выполнено, то это ребро и есть мост.
7. После выполнения двух ПВГ-алгоритмов данные о включенных прямых, обратных ребер либо мостов записываются в вектор для удобства дальнейшего использования в графическом интерфейсе пользователя.

**Пример выполнения.**

* Программа реализована с использованием графического интерфейса
* Пользователь выбирает тип графа: разреженный или плотный
* Генерируется граф
* Пользователь может пошагово выполнить алгоритм добавления прямых и обратных ребер в граф, а также мостов. Есть возможность переходить к предыдущему шагу
* На выходе получается такое окно:



# **ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ**

* Исходные данные:

Вводимое значение пользователем количества вершин в поле формы в графическом интерфейсе.

Выбранная пользователем плотность ребер в графе.

# Ограничения на исходные данные:

Данные должны являться целочисленными и >1&& <=10. Второе условие было выбрано для того, чтобы предоставить более комфортную демонстрацию выполнения алгоритма.

# Результирующие выходные данные:

Результирующими выходными данными являются нарисованный граф в окне интерфейса, который отображает в себе все прямые, обратные ребра, а также мосты.

Также выходными данными являются сообщения, выводимые в форму диагностических сообщений.

# **СПЕЦИФИКАЦИЯ ПРОГРАММЫ**

# Исключительные ситуации в программе:

Пользователь хочет создать граф из 0 или 1 вершины. Такие графы не представляют интереса в данной работе и не являются целью этой работы, потому что в таких графах нет ребер, значит его нельзя проверить на наличие мостов.

Пользователь вводит в поле ввода количеств вершин значение другого типа, отличного от целочисленного. Будет выведено сообщение об ошибке.

Пользователь хочет сделать следующий или предыдущий шаг алгоритма, хотя граф еще не был сгенерирован, либо шаги закончились. Программа не позволит сделать такие действия путем отключения действия кнопки в данных ситуациях.

Пользователь хочет сгенерировать граф, хотя поле ввода количества вершин – пустое. В данном случае будет подано на экран сообщение об ошибке.

* Выходные данные включают в себя:

Нарисованная объект – граф, тестовые сообщение в поле графического интерфейса.

# Сценарий диалога

Перед пользователем появится окно графического интерфейса, где пользователь сможет написать количество вершин, выбрать плотность графа, выбрать следующий или предыдущий шаг алгоритма, а также сгенерировать граф. При правильно поочередности предыдущих действий программа нарисует в определенном поле граф, продемонстрирует алгоритм и выведет некоторую информацию о графе в поле диагностических сообщений. При ошибочных действиях будут сгенерированы исключения в программе.

# **РАЗРАБОТКА СТРУКТУР ДАННЫХ И АЛГОРИТМОВ**

В данной работе используется такая структура данных, неориентированный граф без циклов. Граф хранится при помощи списка смежности, так как в данной работе имеют место быть разреженные графы, а для них не так удобны матрицы смежности.

* Граф генерируется в два этапа: первым этапом создается связная основа вершин (остовное дерево), вторым этапом достраиваются оставшиеся ребра
* Остовное дерево строится при помощи двух контейнеров, в которых хранятся просмотренные и непросмотренные вершины

Алгоритм поиска мостов применяется к посмотренному остовному ПВГ-дереву графа.

Поэтому необходимо построить это дерево. В программе присутствует такая функция.

* Функция заполняет список смежности остовного дерева, перенумеровывает вершины в порядке обхода, отмечает по пути обратные ребра

После построения ПВГ-дерева и нахождения всех обратных ребер, необходимо вычислить функцию P(v) для каждой древесной вершины.

Далее вычисляется Low(v) для вершин, после чего можно рассматривать все древесные ребра и проверять их на выполнение условия *Low(w)>=NumVert(w)), где NumVert(w) –* номер вершины при ПВГ-обходе.

Для визуализации и реализации алгоритма был использован язык программирования JAVA и платформа для создания GUI JAVAFX.

**Общее описание основных функций**

|  |  |  |
| --- | --- | --- |
| **Название функции** | **Параметры** | **Назначение** |
| int graph\_initialisation(vector <vector<int>> graph, int V, density d); | Граф, количество вершин, плотность | Генерирует граф с заданным количеством вершин и плотностью |
| int graph\_V(); | - | Вывод сообщений для ввода графа |
| void graph\_show(vector <vector<int>> graph); | Граф | Подает список смежности на экран |
| void graph\_do\_carcass(vector <vector<int>> graph, vector<vector<int>> b, int V); | Граф, количество вершин, список, имеющий информацию о том, в какую вершину можно построить ребро из любой вершины | Строит остовное дерево с данным количеством вершин |
| void DFS\_bridge(vector<vector<int>> graph, int V, int E, count\_ gr\_counter, ostream file, bool value); | Граф, количество вершин, количество ребер, счетчик времени и операций,выходной файл, булевская переменная для вывода на экран сообщений | Главная функция, выполняющая весь алгоритм нахождения мостов. Включает в себя несколько подфункций. Вычисляет Low(v), проверяет условия наличие моста. |
| bool back\_check(vector<vector<int>> DFS, int vertex1, int vertex2, bool check, int step\_, vector<int> watched, count\_ gr\_counter); | ПВГ-дерево графа, две вершины, булевская переменная проверки, шаг рекурсии, контейнер просмотренных вершин, счетчик времени и базовых операций | Проверяет, есть ли обратное ребро из второй вершины в первую |
| oid DFS\_pre(vector<vector<int>> &graph, vector<int> NumVert, vector<vector<int>> DFS, vector<vector<int>> P, vector<vector<int>> Back, vector<int> unsearched, vector<int> searched, int iV, int Number, count\_ &gr\_counter); | Граф, список перенумерованных вершин, ПВГ-дерево, функции P(v), список обратных ребер, списки просмотренных и непросмотренных вершин, текущая вершина, номер для нумерования вершин ПВГ-дерева, счетчик времени и базовых операций | Строит ПВГ-дерево, перенумеровывает вершины,находит обратные ребра |
| void show\_all(vector<int> NumVert, vector<vector<int>> DFS, vector<vector<int>> P, vector<vector<int>> Back); | Список перенумерованных вершин, ПВГ-дерево, списко функций P(v), список обратных ребер | Подает информацию о данных графа на экран |
| bool wooden(vector<vector<int>> DFS, int iV, int KREK, count\_ gr\_counter); | ПВГ-дерево, текущая вершина, вторая вершина, счетчик времени и базовых операций | Проверяет, есть ли данное ребро в древесных ребрах |
| bool rev(vector<vector<int>> Back, int iV, int KREK, count\_ gr\_counter); | Список обратных ребер, текущая вершина, вторая вершина, счетчик времени и базовых операций | Проверяет, есть ли данное ребро в обратных ребрах |
| void find\_bridges(vector<int> NumVert, vector<vector<int>> DFS, vector<vector<int>> P, vector<vector<int>> Back, int iV, int FOR\_RECORD, vector<int> watch, int count, count\_ gr\_counter); | Граф, список перенумерованных вершин, ПВГ-дерево, список функций P(v), список обратных ребер, текущая вершина, поступившая вершина, список просмотренных вершин, шаг рекурсии, счетчик времени и базовых операций | Вычисляет функцию P(v) для поступившей вершины |
| public void Generation\_kek() | - | Генерация графа в окне графического интерфейса, его отрисовка, установка начальных значений для кнопок управления графического интерфейса |
| public void paint\_GRAPH ( graph G, Circle R) | Граф, круг | Отрисовка графа на экране |
| public void hideRibs (int counter, Algorithm alg) | Счетчик отрисованных ребер, объект класса алгоритм | Скрывает последнее отрисованное ребро на экране |
| public void showRibs(int counter , Algorithm alg) | Счетчик отрисованных ребер, объект класса алгоритм | Рисует следующее ребро которое было добавлено алгоитмом |
| public void incCounter() { | - | Инкрементирование счетчика отрисованных ребер, отрисовка ребра |
| public void decCounter() | - | Декрементирование счетчика, скрытие последнего отрисованного ребра |
| public void error(String s) | Аварийное сообщение | Подача сообщение исключения на экран |

**Модули программы** :

Программа разбита на несколько модулей:

Graph.java - содержит объявление и определение функций по генерации и выводу информации по графу на экран.

Main.java- основной модуль

Algorithm.java – содержит объявление и определение функций для реализации главного алгоритма, структуру счетчика.

Controller.java – содержит функции по обработке действий с формами в окне графического интерфейса

Sample\_1.xml – модуль, содержащие все свойства главного окна и всех форм на нем

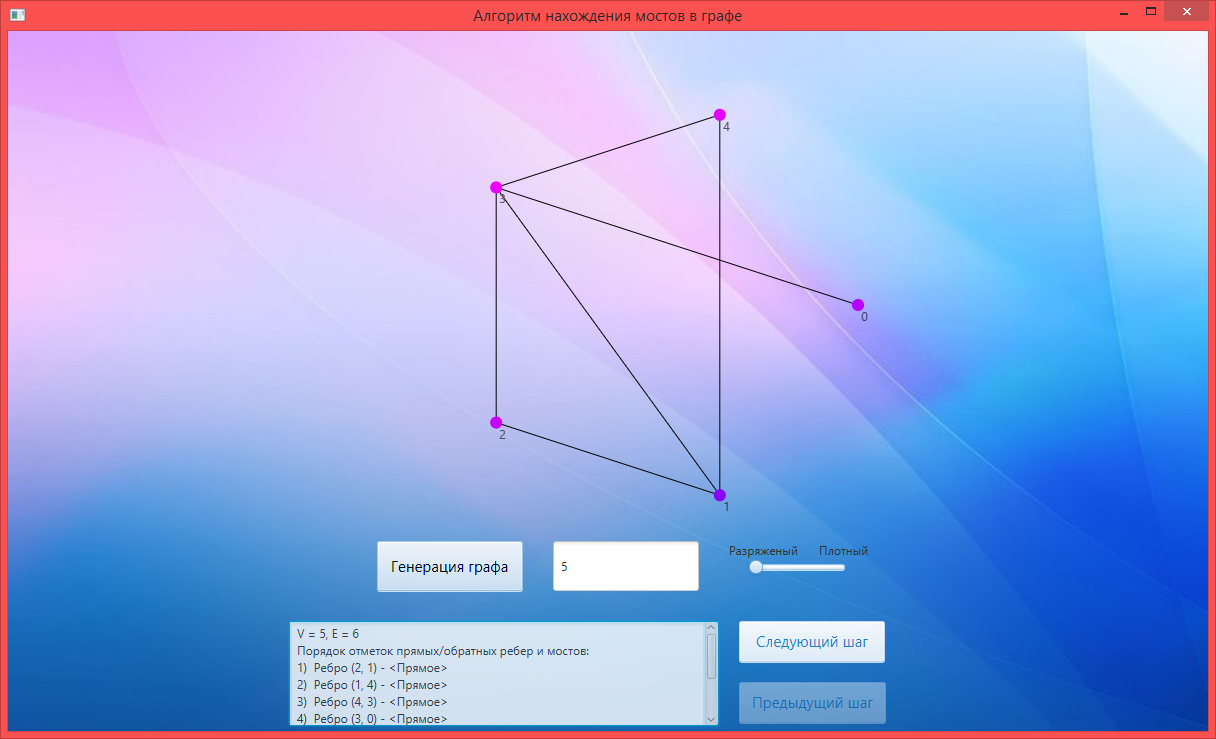
Fon.png – фон окна графического интерфейса

# **ТЕСТИРОВАНИЕ ПРОГРАММЫ**

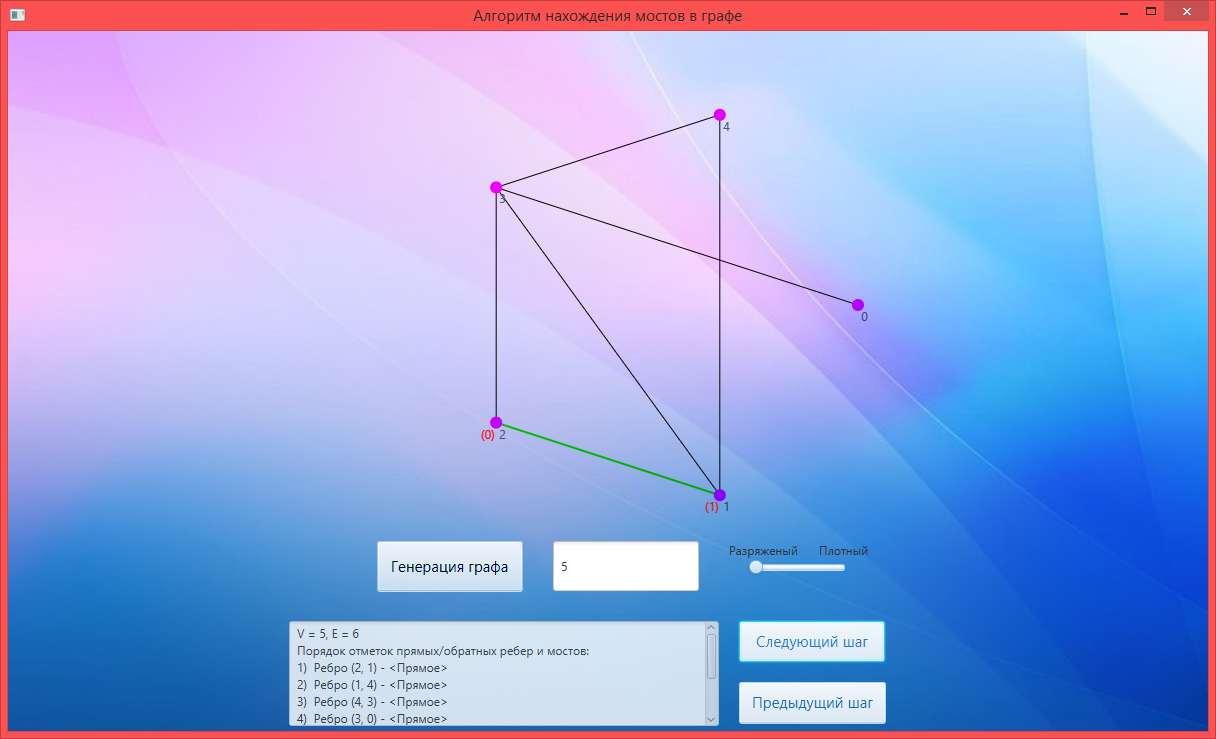
1. Проведем тестирование для |V| = 5, |E| = 6

Пошаговая работа алгоритма:

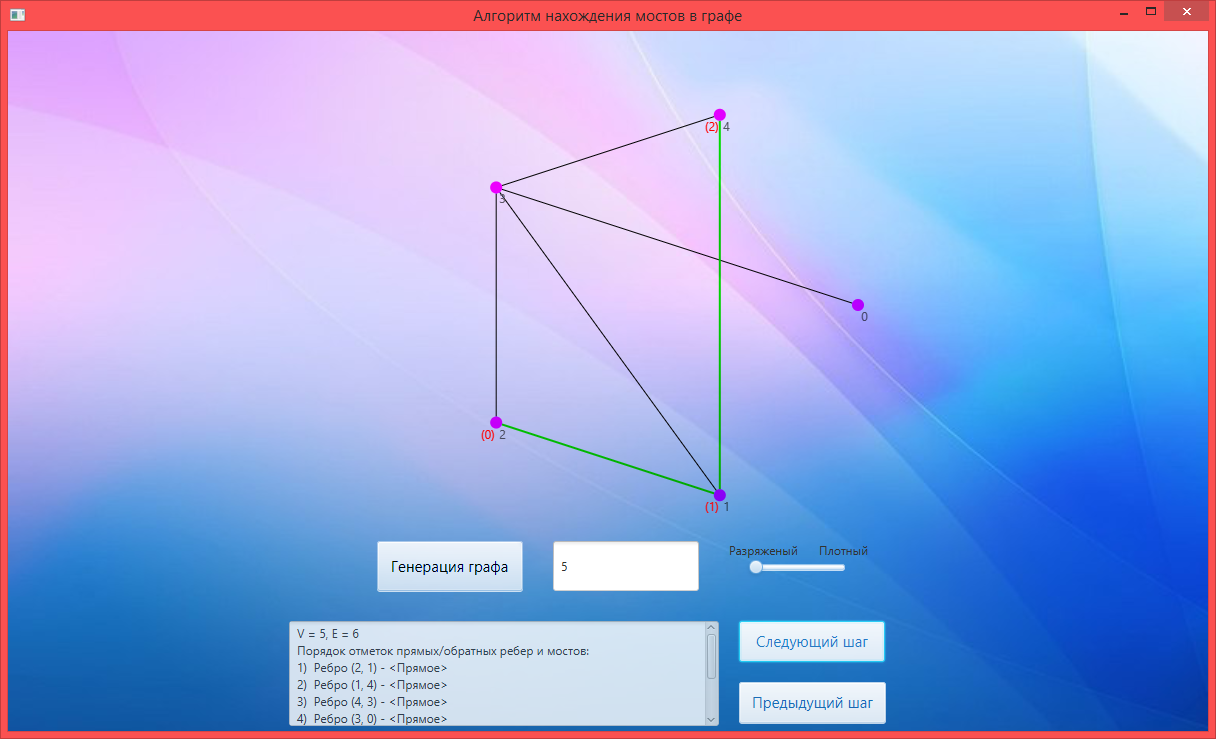
Зеленый цвет – прямое ребро. Желтый цвет – обратное. Красный цвет – мост.



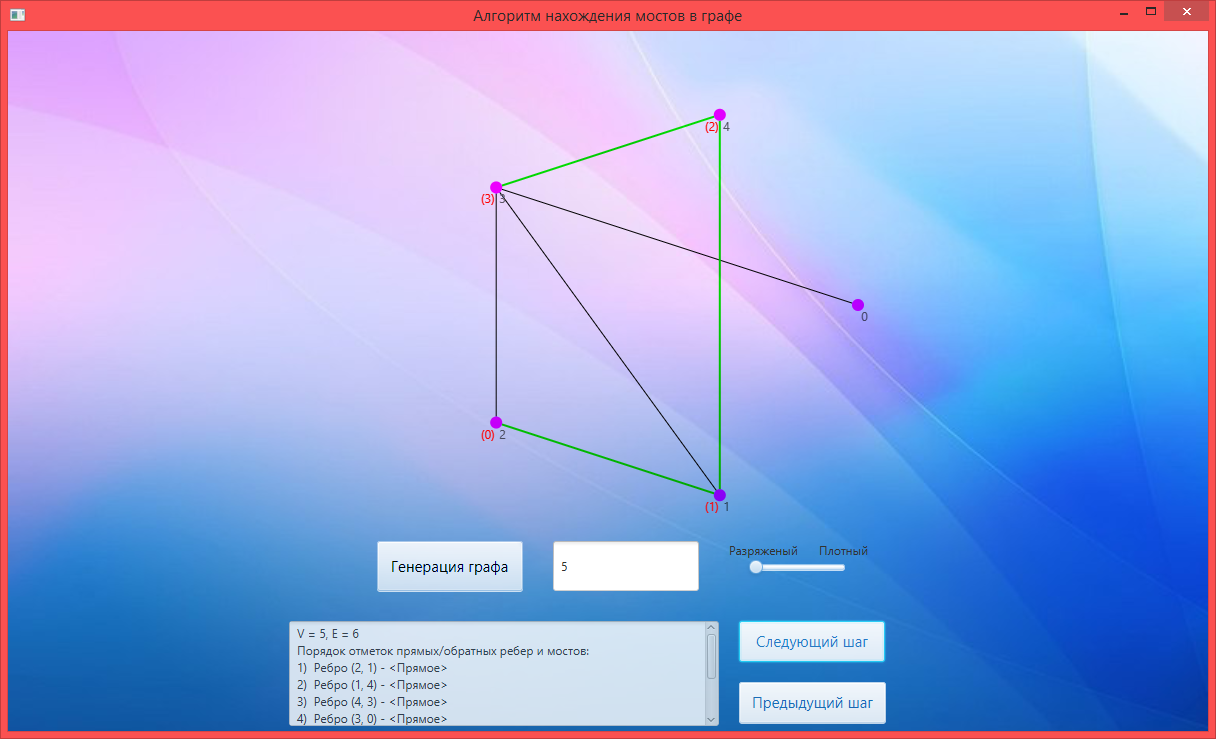
1) Ребро (2, 1) - <Прямое>



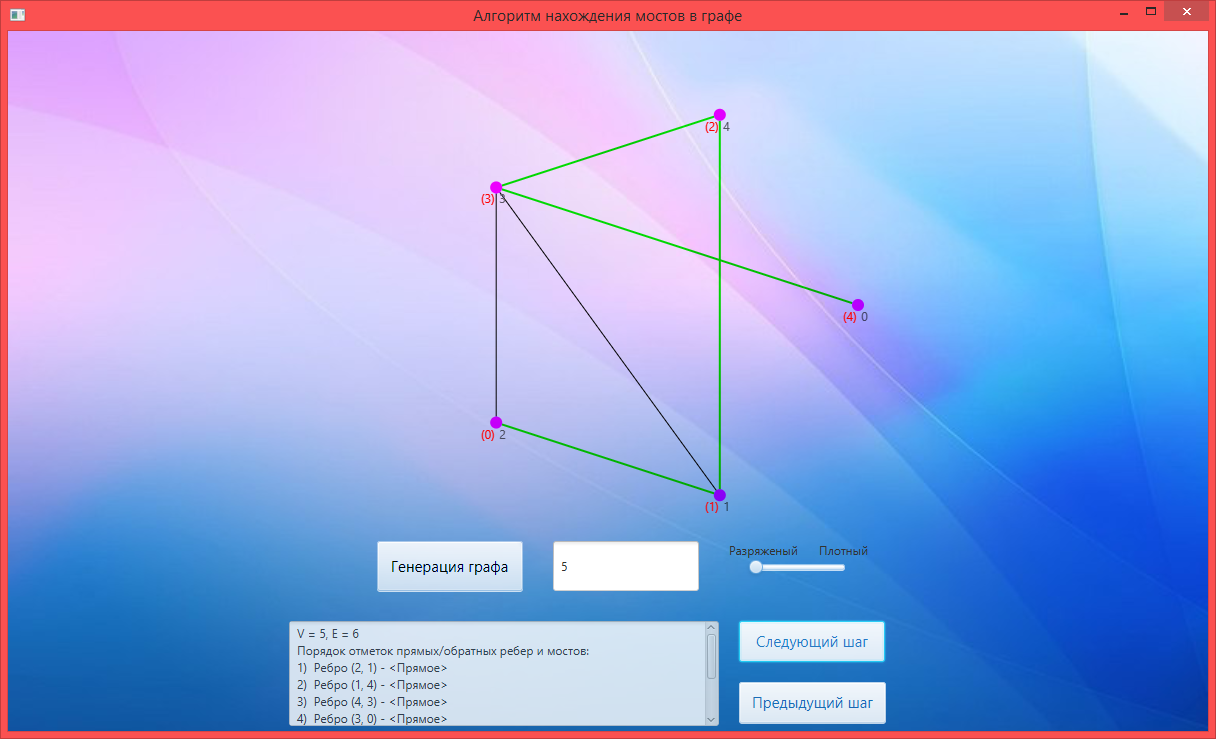
2)Ребро (1, 4) - <Прямое>



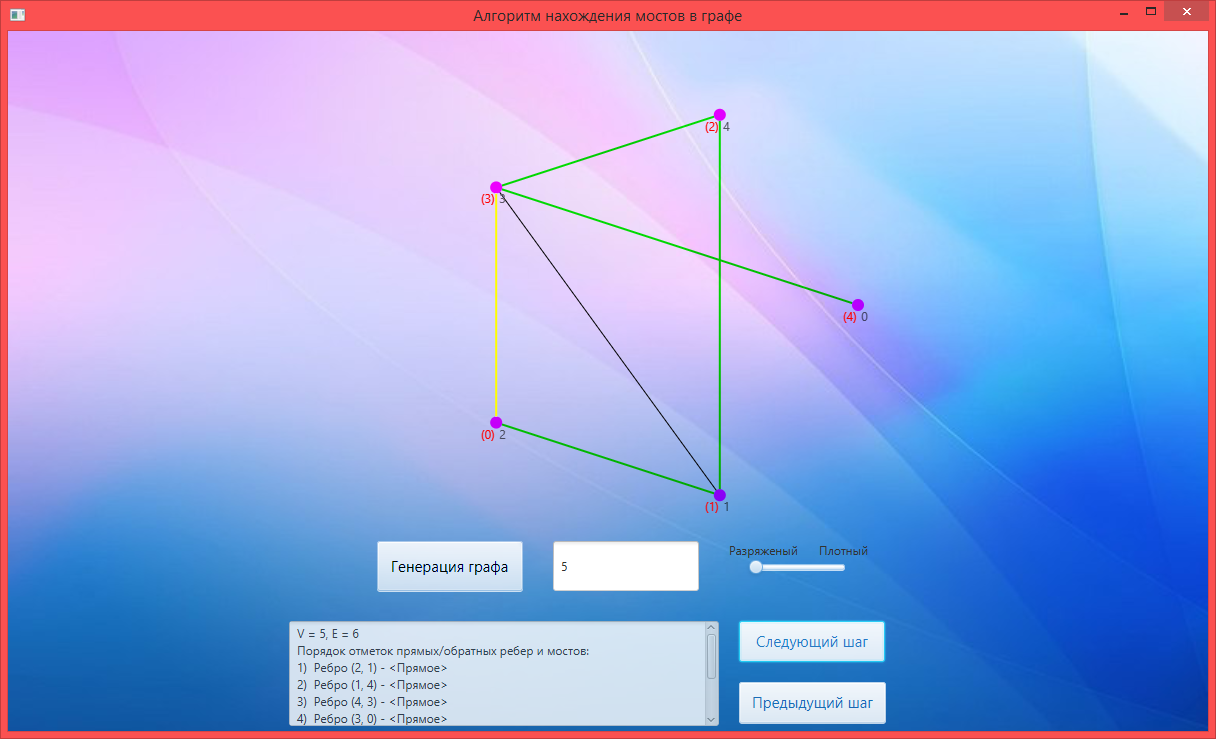
3) Ребро (4, 3) - <Прямое>



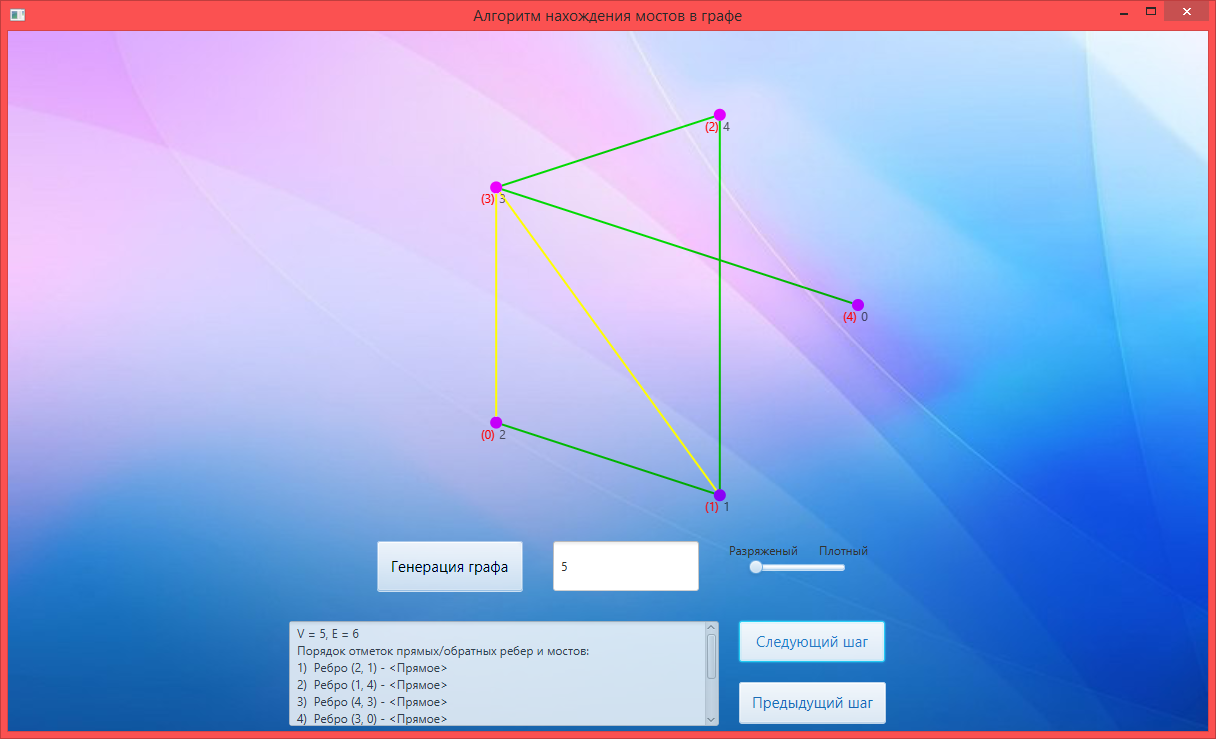
4) Ребро (3, 0) - <Прямое>



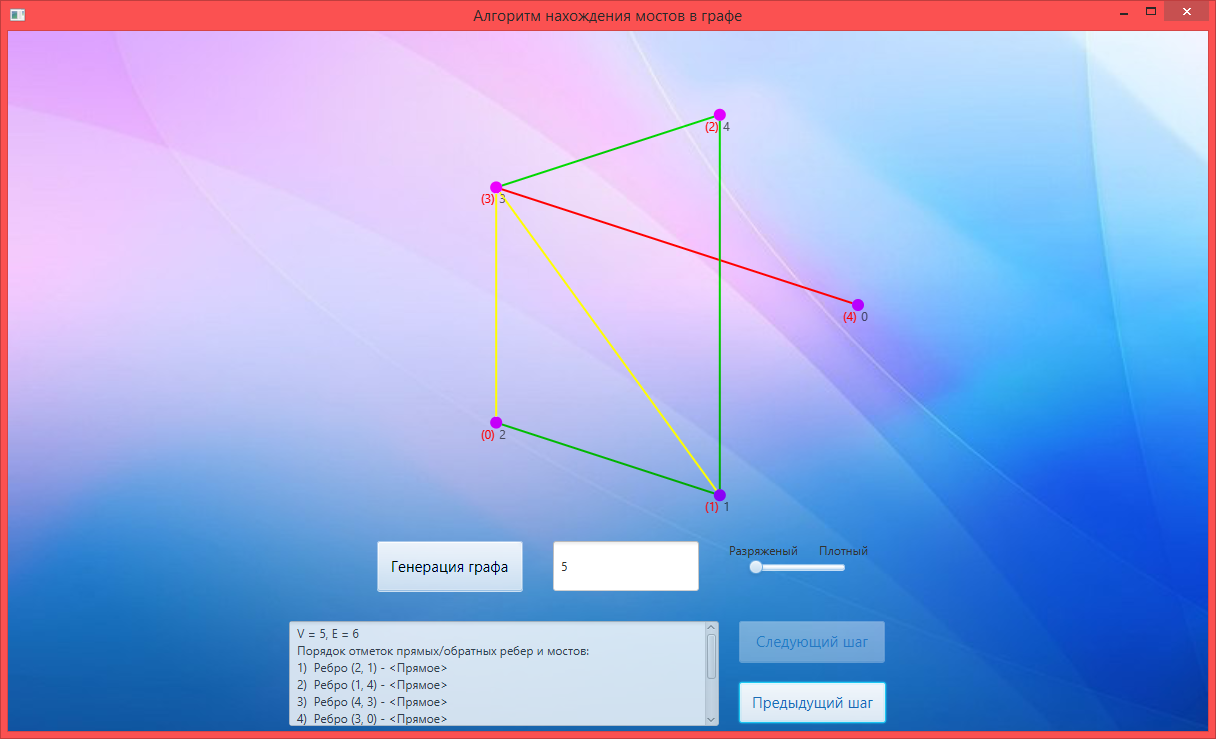
5) Ребро (3, 2) - <Обратное>



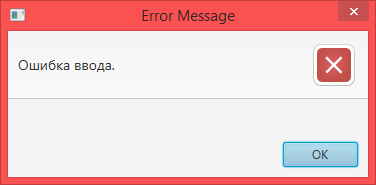
6) Ребро (3, 1) - <Обратное>



7) Ребро (3, 0) - <Мост>



Пример вывода исключения:



Вывод сообщения при попытке генерации без входных данных.

# **КОД ПРОГРАММЫ**

graph.java

|  |
| --- |
| package sample;  import java.util.Scanner;  import java.util.Vector;  /\*\*  \* Created by Максим on 24.06.2016.  \*/  public class graph {  Vector<Vector<Integer>> gr\_; // список смежности графа  public enum density { rare, normal };  graph(int number) {  V = number;  gr\_ = new Vector<Vector<Integer>>();  for(int i = 0; i < V; i++) //добавлено  {  //Vector<Integer> In = new Vector<Integer>();  gr\_.add(new Vector<Integer>());  }  }  public static int V;  public int E;  /\*\*  \*  \* @param b  \* @param V - количество вершин  \*/  public void do\_carcass(Vector<Vector<Integer>> b, int V){  Vector<Integer> searched = new Vector<Integer>(); // просмотренные вершины  Vector<Integer> unsearched= new Vector<Integer>(); // непросмотренные вершины  for (int k = 0; k < V; k++)  {  unsearched.add(k);  }  int RAND\_FOR\_CARCASS = 0;  int first = 0;  while (unsearched.size() != 0)  {  if (searched.size() == 0)  {  first = 0;  }  else  {  first = (int)(Math.random()\* searched.size());  first = searched.get(first);  }  boolean flag1 = true;  boolean flag2 = true;  for (int l = 0; l < searched.size(); l++)  {  if (searched.get(l) == first)  {  flag1 = false;  }  }  if (flag1)  {  searched.add(first);  }  boolean flag33 = false;  boolean flag4 = false;  int INDEX1 = 0;  int INDEX2 = 0;  for (int u = 0; u < unsearched.size(); u++)  {  if (unsearched.get(u) == first)  {  flag33 = true;  INDEX1 = unsearched.get(u);  }  }  if (flag33)  {  unsearched.removeElement(INDEX1);  }  RAND\_FOR\_CARCASS = (int)(Math.random()\*unsearched.size());  RAND\_FOR\_CARCASS = unsearched.get(RAND\_FOR\_CARCASS); // посмотреть  gr\_.get(first).add(RAND\_FOR\_CARCASS);// = rand() % (V - 1 + 1) + 1;  gr\_.get(RAND\_FOR\_CARCASS).add(first);  //вставка  for (int l = 0; l < searched.size(); l++)  {  if (searched.get(l) == RAND\_FOR\_CARCASS)  {  flag2 = false;  }  }  if (flag2)  {  searched.add(RAND\_FOR\_CARCASS);  }  //удаление  for (int u = 0; u < unsearched.size(); u++)  {  if (unsearched.get(u) == RAND\_FOR\_CARCASS)  {  flag4 = true;  INDEX2 = unsearched.get(u);  }  }  if (flag4)  {  unsearched.removeElement(INDEX2);  }  b.get(first).removeElement(RAND\_FOR\_CARCASS);  b.get(RAND\_FOR\_CARCASS).removeElement(first);  }  }  /\*\*  \*  \* @param V  \* @param d  \* @return  \*/  int initialisation(int V, density d)  {  int E = 0;  if (V == 2)  {  E = 1;  }  else  {  if (d == density.rare) // разреженный граф  {  E = (int)(Math.random()\*(V\*(V - 1) / 2 - V + 1) + V - 1); // что-то типа от V-1 до V\*(V - 1) / 2  while (E > (V\*(V - 1) / 2 + (V - 1)) / 2)  {  E = (int)(Math.random()\*(V\*(V - 1) / 2 - V + 1 ) + V - 1) ;  }  }  else  {  E = (int)(Math.random()\*(V\*(V - 1) / 2 - V + 1) + V - 1); // что-то типа от V-1 до V\*(V - 1) / 2  while (E <= (V\*(V - 1) / 2 + (V - 1)) / 2)  {  E = (int)(Math.random()\*(V\*(V - 1) / 2 - V + 1 + 1) + V - 1); // добавил единичку  }  }  }  int rand\_vertex = 0;  Vector<Vector<Integer>> b = new Vector<Vector<Integer>> (V);  for (int ink = 0; ink < V; ink++){  b.add(new Vector<Integer>());  }  for (int ind = 0; ind < V; ind++)  {  for (int dni = 0; dni < V ; dni++)  {  if (ind != dni)  {  b.get(ind).add(dni);  }  }  }  do\_carcass(b, V);  int balance\_v = E - V + 1;  int pnt = 0;  while (balance\_v != 0)  {  pnt = (int)(Math.random()\* V); // выбираем случайную вершину  if (b.get(pnt).size() != 0)  {  rand\_vertex = (int)(Math.random()\* (b.get(pnt).size()));  rand\_vertex = b.get(pnt).get(rand\_vertex);  gr\_.get(pnt).add(rand\_vertex);  b.get(pnt).removeElement(rand\_vertex);  gr\_.get(rand\_vertex).add(pnt);  b.get(rand\_vertex).removeElement(pnt);  balance\_v--;  }  }  return E;  }  /\*\*  \* Тут должна выводится диаграмма графа в окошке  \*/  public int enter(){  int V;  System.out.println("Введите количество вершин в графе:");  Scanner sc = new Scanner(System.in);  V = sc.nextInt();  while (V <= 1) // исправил на 1  {  System.out.println("Повторите ввод:");  V = sc.nextInt();  }  return V;  }  public void show(Vector<Vector<Integer>> K){  System.out.println("Список смежности графа:");  for (int i = 0; i < K.size(); i++)  {  System.out.print(i + "-я вершина: ");//i << "-я вершина: ";  for (int j = 0; j < K.get(i).size(); j++)  {  System.out.print(K.get(i).get(j));  // cout << graph[i][j] << ' ';  }  System.out.println('\n');  }  }  } |

Algorithm.java

|  |
| --- |
| package sample;  import java.util.Collections;  import java.util.Vector;  /\*\*  \* Created by Максим on 26.06.2016.  \*/  public class Algorithm {  Vector<Integer> revNum = new Vector<>();  //Vector<Integer> NumVert = new Vector<>();  Vector<Integer> Low;// = new Vector<>(G.V);  Vector<Vector<Integer>> P ;//= new Vector<>(G.V);  Vector<Vector<Integer>> DFS ;//= new Vector<>(G.V);  Vector<Vector<Integer>> Back ;//= new Vector<>(G.V); // обратные ребра  private static int Number = 0;  public Vector<Vector<Integer>> Order = new Vector<>(/\*2\*G.E\*/); // массив, в котором указывается порядок включения ребер в Обратные/Прямые ребра  public int Numeration; // для нумерации ребер в предыдущем массиве  private void incNumeration() { Numeration++;}  /\*\*  \*  \* @param G  \*/  public void DFS\_bridge(graph G)  {  Vector<Integer> NumVert = new Vector<>(G.V);  Vector<Integer> Low = new Vector<>(G.V);  Vector<Vector<Integer>> P = new Vector<>(G.V);  Vector<Vector<Integer>> DFS = new Vector<>(G.V);  Vector<Vector<Integer>> Back = new Vector<>(G.V); // обратные ребра  // Vector<Vector<Integer>> Order = new Vector<>(2\*G.E); // порядок включения ребер !!!!!!!!!!!!!!!!!  Vector<Integer> searched = new Vector<>(G.V); // просмотренные вершины  Vector<Integer> unsearched = new Vector<>(); // непросмотренные вершины  for (int k = 0; k < G.V; k++)  {  unsearched.add(k); //посмотреть  searched.add(k,-1);  Back.add(new Vector<>());  DFS.add(new Vector<>());  P.add(new Vector<>());  NumVert.add(-1);  Low.add(-1);  }  System.out.println("\*\*\*Размерность основного массива Order = " + 2\*G.E + "\*\*\*");  for (int k = 0; k < 2\*G.E; k++)  {  /\*  первый и второй элемент - вершины ребра, \  3-ий элемент - идентификатор:  '0' - мост;  '1' - прямое ребро;  '2' - обратное ребро;  \*/  Order.add(new Vector<>());  }  Numeration = 0;  //построение DFS-остова  int iV;//текущая вершина  // int Number = 0; // номер для перенумерования вершин в дереве  int first\_vertex = (int)(Math.random()\* G.V);  System.out.println("Корень = " + first\_vertex);  //NumVert[first\_vertex] = Number; // перенумеровали в первую вершину  NumVert.setElementAt(Number,first\_vertex);  P.get(first\_vertex).add(first\_vertex);//[first\_vertex].push\_back(first\_vertex); // вычисляем P(v) для первой вершины  iV = first\_vertex;  unsearched.setElementAt(-1,iV);  searched.setElementAt(iV,iV);  DFS\_pre(G, unsearched, searched, iV, Number,NumVert,DFS,Back,P, Order);  Vector<Integer> watch = new Vector<>(G.V);  for (int i = 0; i < G.V; i++) {  watch.add(i,-1);  }  Vector<Integer> unwatch = new Vector<>();  for (int k = 0; k < DFS.size(); k++)  {  unwatch.add(k);//push\_back(k);  }  iV = first\_vertex;  System.out.println(" ");  //Vector<Integer> revNum = new Vector<>(G.V);  for (int i = 0; i < G.V; i++)  {  revNum.add(0); // например ноль  }  for (int in = 0; in < G.V; in++)  {  revNum.setElementAt(in,NumVert.get(in));  }  int safety = 0;  int FOR\_RECORD = 0;  int count = 0;  for (int pk = 0; pk < G.V; pk++)  {  safety = pk;  pk = revNum.get(pk);  if (pk != first\_vertex) // чтобы первую не задействовать  {  find\_bridges(pk, pk, watch, count,DFS,Back,P);  }  watch.setElementAt(pk,pk);  pk = safety;  }  for (int u = 0; u < DFS.size(); u++)  {  for (int y = 0; y < P.get(u).size(); y++)  {  P.get(u).setElementAt(NumVert.get(P.get(u).get(y)),y);  }  }  for (int r = 0; r < DFS.size(); r++)  {  Low.setElementAt(Collections.min(P.get(r)),r);  }  System.out.println("МОСТЫ:");  for (int fin = 0; fin < G.V; fin++)  {  for (int infin = 0; infin < DFS.get(fin).size(); infin++)  {  if(Low.get(DFS.get(fin).get(infin)) >= NumVert.get(DFS.get(fin).get(infin))) //проверить, например на equals  {  System.out.println(fin + " - " + DFS.get(fin).get(infin));  FOR\_RECORD++;  // Запоминаем порядок включения мостов  Order.get(Numeration).add(0, fin);  Order.get(Numeration).add(1, DFS.get(fin).get(infin));  Order.get(Numeration).add(2, 3); // тип ребра: '2' == обратное  incNumeration();  }  }  }  show\_all(G,NumVert,DFS,Back,P);  System.out.println("Количество мостов - " + FOR\_RECORD);  System.out.println("\n Размер вектора Order = " + Order.size());  Order.setSize(G.E + FOR\_RECORD);  System.out.println(" А теперь = " + Order.size());  Order.setSize(Numeration);  System.out.println(" А вот прям теперь = " + Order.size());  // Проверка получившегося массива  System.out.println("\n Порядок добавления ребер:");  for (int i = 0; i < Order.size(); i++) // WARNING Иногда цикл выжодит за область существующих элементов --> ERROR  {  System.out.println("\n(" + Order.get(i).get(0) + ", " +  Order.get(i).get(1) + ", " + Order.get(i).get(2) + ") ");  }  }  /\*\*  \*  \* @param G  \* @param unsearched  \* @param searched  \* @param iV  \* @param Number  \* @param NumVert  \* @param DFS  \* @param Back  \* @param P  \* @param Order  \* @return  \*/  public int DFS\_pre(graph G, Vector<Integer> unsearched, Vector<Integer> searched, int iV, int Number,  Vector<Integer> NumVert,Vector<Vector<Integer>> DFS,Vector<Vector<Integer>> Back,  Vector<Vector<Integer>> P, Vector<Vector<Integer>> Order)  {  for (int j = 0; j < G.gr\_.get(iV).size(); j++)  {  if(searched.get(G.gr\_.get(iV).get(j)) == G.gr\_.get(iV).get(j)) //проверить на equals  {  boolean kek = false;  int step\_ = 0;  Vector<Integer> watched = new Vector<>(G.gr\_.size());  for (int i = 0; i < G.V; i++) {  watched.add(i,-1);  }  if (!wooden(DFS, iV, G.gr\_.get(iV).get(j)) && !(rev(Back, iV, G.gr\_.get(iV).get(j)))) // нужна проверка, есть ли это ребро уже в обратных или древесых  {  if (back\_check(DFS, G.gr\_.get(iV).get(j), iV, kek, step\_, watched)) // поменял местами вершины в функции  {  Back.get(iV).add(G.gr\_.get(iV).get(j));  // Запоминаем порядок включения обратных ребер  Order.get(Numeration).add(iV);  Order.get(Numeration).add(G.gr\_.get(iV).get(j));  Order.get(Numeration).add(2); // тип ребра: '2' == обратное  incNumeration();  }  }  }  else  {  DFS.get(iV).add(G.gr\_.get(iV).get(j));  // Запоминаем порядок включения прямых ребер  //System.out.println("\*\*\*Размерность = (" + Order.capacity() + ", " + Order.get(0).capacity() + ")\*\*\*");  Order.get(Numeration).add(0, iV);  Order.get(Numeration).add(1, G.gr\_.get(iV).get(j));  Order.get(Numeration).add(2, 1); // тип ребра: '1' == прямое  incNumeration();  //System.out.print("\n Последний элемент:" + );  Number++;  //int T = Number;  NumVert.setElementAt( Number,G.gr\_.get(iV).get(j));  // NumVert.setElementAt(Number+1,G.gr\_.get(iV).get(j));  P.get(G.gr\_.get(iV).get(j)).add(G.gr\_.get(iV).get(j));  unsearched.setElementAt(-1,G.gr\_.get(iV).get(j));  searched.setElementAt(G.gr\_.get(iV).get(j),G.gr\_.get(iV).get(j));  Number = DFS\_pre(G, unsearched, searched, G.gr\_.get(iV).get(j), Number,NumVert,DFS,Back,P, Order); // рекурсия  }  }  return Number;  }  /\*\*  \*  \* @param DFS  \* @param vertex1  \* @param vertex2  \* @param check  \* @param step\_  \* @param watched  \* @return  \*/  boolean back\_check(Vector<Vector<Integer>> DFS, int vertex1, int vertex2, boolean check, int step\_, Vector<Integer> watched)  {  for (int j = 0; j < DFS.get(vertex1).size(); j++)  {  if ((check) || (DFS.get(vertex1).get(j) == vertex2)) // если уже найдено или нашлось сейчас//опять equals  {  if (check) // если обратное  {  return check;  }  else  {  if (step\_ == 0) // если прямой предок  {  check = false;  return check;  }  else // если непрямой предок  {  check = true;  return check; // проверить  }  }  }  else  {  if (watched.get(j).equals(j))  {  watched.setElementAt(j,j);  back\_check(DFS, j, vertex2, check, ++step\_, watched); // проверяем дальше  }  //иначе продолжаем цикл  }  }  return true;  }  /\*\*  \*  \* @param G  \* @param NumVert  \* @param DFS  \* @param Back  \* @param P  \*/  public void show\_all(graph G, Vector<Integer> NumVert, Vector<Vector<Integer>> DFS,Vector<Vector<Integer>> Back,Vector<Vector<Integer>> P)  {  System.out.println("Остовное дерево:");  G.show(DFS);  System.out.println("Обратные ребра:");  G.show(Back);  System.out.println("P(V) для вершин:");  G.show(P);  System.out.println("Перенумерация в DFS-обходе:");  for (int j = 0; j < NumVert.size(); j++)  {  System.out.print(j + "-я вершина: ");  System.out.println(NumVert.get(j) + " ");//опять махинации с пробелом!!  }  System.out.println(" ");  }  /\*\*  \*  \* @param DFS  \* @param iV  \* @param KREK  \* @return  \*/  boolean wooden(Vector<Vector<Integer>> DFS, int iV, int KREK) // пока проверка на древесность в одну сторону  {  //cout << "ДРЕВЕСНОСТЬ" << endl;  for (int zer = 0; zer < DFS.get(KREK).size(); zer++)  {  if (DFS.get(KREK).get(zer) == iV)  {  //cout << "Да, древесная" << endl;  return true;  }  }  //cout << "Нет, не древесная" << endl;  return false;  }  /\*\*  \*  \* @param Back  \* @param iV  \* @param KREK  \* @return  \*/  boolean rev(Vector<Vector<Integer>> Back, int iV, int KREK)  {  //cout << "ОБРАТНОСТЬ" << endl;  for (int zer = 0; zer < Back.get(KREK).size(); zer++)  {  if (Back.get(KREK).get(zer) == iV) // equals проверить  {  //cout << "Да, обратная" << endl;  return true;  }  }  //cout << "Нет, не обратная" << endl;  return false;  }  /\*\*  \*  \* @param iV  \* @param FOR\_RECORD  \* @param watch  \* @param count  \* @param DFS  \* @param Back  \* @param P  \*/  void find\_bridges( int iV, int FOR\_RECORD, Vector<Integer> watch, int count, Vector<Vector<Integer>> DFS,Vector<Vector<Integer>> Back,Vector<Vector<Integer>> P)  {  for (int j = 0; j < DFS.get(iV).size(); j++)  {  //cout << "Тек вершина " << iV << ", тек номер j " << j << endl;  if(DFS.get(DFS.get(iV).get(j)).size() != 0)  {  find\_bridges( DFS.get(iV).get(j), FOR\_RECORD, watch, count,DFS,Back,P); // проверить  }  for (int z = 0; z < Back.get(DFS.get(iV).get(j)).size(); z++)  {  // если ведущее обратное ребро попало а просмотренную вершину, вершина в p(v)  if(watch.get(Back.get(DFS.get(iV).get(j)).get(z)) == Back.get(DFS.get(iV).get(j)).get(z)) //equals все дела  {  if (!(P.get(FOR\_RECORD).contains(Back.get(DFS.get(iV).get(j)).get(z))))  {  //запихать вершину в которую можно попасть таким способом  P.get(FOR\_RECORD).add(Back.get(DFS.get(iV).get(j)).get(z));  Collections.sort(P.get(FOR\_RECORD));  }  }  }  }  if (count == 0)  {  for (int z = 0; z < Back.get(FOR\_RECORD).size(); z++)  {  if (watch.get(Back.get(FOR\_RECORD).get(z)) == Back.get(FOR\_RECORD).get(z)) //equals  {  if(!(P.get(FOR\_RECORD).contains(Back.get(FOR\_RECORD).get(z))))  {  P.get(FOR\_RECORD).add(Back.get(FOR\_RECORD).get(z));  }  }  }  }  }  } |

Controller.java

|  |
| --- |
| package sample;  /\*\*  \* Created by ДРЮС on 27.06.2016.  \*/  import javafx.fxml.FXML;  import javafx.geometry.Rectangle2D;  import javafx.scene.canvas.Canvas;  import javafx.scene.control.\*;  import javafx.scene.image.Image;  import javafx.scene.image.ImageView;  import javafx.scene.layout.GridPane;  import javafx.scene.layout.HBox;  import javafx.scene.layout.Pane;  import javafx.scene.layout.VBox;  import javafx.scene.paint.Color;  import javafx.scene.shape.Circle;  import javafx.scene.shape.Line;  import javafx.scene.shape.Rectangle;  import javafx.scene.Scene;  import java.security.MessageDigest;  import java.util.ArrayList;  import java.util.Collections;  import java.util.Vector;  //import java.  public class Controller {  @FXML  private TextField Amount;  @FXML  private TextArea Messeges;  @FXML  private Button Generation;  @FXML  private Circle Circle1;  @FXML  private GridPane Main\_stage;  @FXML  private Pane Pane1;  @FXML  private Button Next\_step;  @FXML  private Button Prev\_step;  @FXML  private Slider PIMPSlide;  private ArrayList<Circle> CircArr;// = new ArrayList<>();  private ArrayList<Label> LabArr;// = new ArrayList<>();  private ArrayList<Line> LinArr;// = new ArrayList<>();  private ArrayList<Label> LabDFS;  /\*\*  \* Граф  \*/  private graph G;  private Algorithm alg;  private int counter = -1;  private static int INDEX = 0;  public void Generation\_kek() {  try {  int x = Integer.parseInt(Amount.getText());  if (x == 228)  {  EGG("Один четыре восемь восемь");  }  else if (x == 1488)  {  EGG("Два два восемь");  }  else if (x == 2016)  {  EGG("Графа нет! Но вы держитесь тут, всего вам доброго и хорошего настроения!");  }  else if (x > 10 || x < 2) {  error("Выход за пределы дозволенного.");  } else {  G = new graph(x);  // Инициализация графа  if(PIMPSlide.getValue() == 0) G.E = G.initialisation(x, graph.density.rare);  else G.E = G.initialisation(x, graph.density.normal);  paint\_GRAPH(G,Circle1);  G.show(G.gr\_);  Generation.setDisable(true);  // Отработка алгоритма  alg = new Algorithm();  alg.DFS\_bridge(G);  // Выдача плана прохождения алгоритма нахождения мостов  Messeges.clear();  Messeges.appendText("V = " + G.V + ", E = " + G.E+ "\n" );  // Messeges.appendText("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  Messeges.appendText("Порядок отметок прямых/обратных ребер и мостов:\n");  for (int i = 0; i < alg.Order.size(); i++)  {Messeges.appendText((i+1) + ") Ребро (" + alg.Order.get(i).get(0) + ", " + alg.Order.get(i).get(1) + ") - ");  switch (alg.Order.get(i).get(2))  {  case (1):  Messeges.appendText("<Прямое>\n");  break;  case (2):  Messeges.appendText("<Обратное>\n");  break;  case (3):  Messeges.appendText("<Мост>\n");  break;  }  }  if (alg.Order.get(alg.Order.size()-1).get(2) != 3)  Messeges.appendText(" В данном графе нет ни одного моста!");  // Активируем кнопку "Следующий шаг"  Next\_step.setDisable(false);  LabDFS = new ArrayList<>(); //ДОБАВЛЕНО  for (int i = 0; i < G.V; i++)  {  LabDFS.add(new Label("(" + i + ")"));  LabDFS.get(i).setVisible(false);  LabDFS.get(i).setLayoutX(CircArr.get(alg.revNum.get(i)).getCenterX() -15);  LabDFS.get(i).setLayoutY(CircArr.get(alg.revNum.get(i)).getCenterY() + 3);  LabDFS.get(i).setTextFill(Color.RED);  Pane1.getChildren().add(LabDFS.get(i));  }  Prev\_step.setDisable(true);  }  } catch (NumberFormatException ex) {  ex.printStackTrace();  error("Ошибка ввода.");  }  }  /\*\*  \* Функция отрисовки графа  \* @param G - граф  \* @param R - круг, на котором будут располагаться вершины графа  \*/  public void paint\_GRAPH ( graph G, Circle R){  CircArr = new ArrayList<>(); //точки  LabArr = new ArrayList<>(); //лейблы  LinArr = new ArrayList<>(); // линии  // Отрисовка вершин графа  double x1,y1;  for (int i =0; i < G.V; i++){  x1 = R.getRadius()\*Math.cos(2\*Math.PI/G.V\*i) + R.getCenterX();  y1 = R.getRadius()\*Math.sin(2\*Math.PI/G.V\*i) + R.getCenterY();  CircArr.add(new Circle(x1,y1,7,Color.DARKVIOLET));  LabArr.add(new Label(" " + i));  LabArr.get(i).setLayoutX(CircArr.get(i).getCenterX());  LabArr.get(i).setLayoutY(CircArr.get(i).getCenterY() + 3);  }  // Отрисовка ребер графа  boolean FLAGHOK = true;  for (int end = 0, counter = 0; end < G.V; end ++){  for (int KUK = 0; KUK < G.gr\_.get(end).size(); KUK++)  {  for (int TRALALA = 0; TRALALA < LinArr.size(); TRALALA++ )  {  if ((LinArr.get(TRALALA).getStartX() == CircArr.get(G.gr\_.get(end).get(KUK)).getCenterX()) &&  (LinArr.get(TRALALA).getStartY() == CircArr.get(G.gr\_.get(end).get(KUK)).getCenterY()) &&  (LinArr.get(TRALALA).getEndX() == CircArr.get(end).getCenterX()) &&  (LinArr.get(TRALALA).getEndY() == CircArr.get(end).getCenterY()))  {  FLAGHOK = false;  break;  }  }  if (FLAGHOK)  {  LinArr.add(new Line(CircArr.get(end).getCenterX(),CircArr.get(end).getCenterY(),CircArr.get(G.gr\_.get(end).get(KUK)).getCenterX(),CircArr.get(G.gr\_.get(end).get(KUK)).getCenterY()));  Pane1.getChildren().add(LinArr.get(counter));  counter++;  }  FLAGHOK = true;  }  }  for (int i = 0; i < G.V; i++)  {  Pane1.getChildren().add(LabArr.get(i));  Pane1.getChildren().add(CircArr.get(i));  }  }  /\*\*  \*  \* @param counter  \* @param alg  \*/  public void showRibs(int counter , Algorithm alg) // должен измениться каунтер  {  for(int j = 0; j < LinArr.size(); j++)  {  if ((( LinArr.get(j).getStartX() == CircArr.get(alg.Order.get(counter).get(0)).getCenterX() ) &&  (LinArr.get(j).getStartY() == CircArr.get(alg.Order.get(counter).get(0)).getCenterY()) &&  (LinArr.get(j).getEndX() == CircArr.get(alg.Order.get(counter).get(1)).getCenterX()) &&  (LinArr.get(j).getEndY() == CircArr.get(alg.Order.get(counter).get(1)).getCenterY())) ||  ((( LinArr.get(j).getEndX() == CircArr.get(alg.Order.get(counter).get(0)).getCenterX() ) &&  (LinArr.get(j).getEndY() == CircArr.get(alg.Order.get(counter).get(0)).getCenterY()) &&  (LinArr.get(j).getStartX() == CircArr.get(alg.Order.get(counter).get(1)).getCenterX()) &&  (LinArr.get(j).getStartY() == CircArr.get(alg.Order.get(counter).get(1)).getCenterY()))))  {  if (alg.Order.get(counter).get(2) == 1)  {  if (INDEX == 0)  {  LabDFS.get(INDEX).setVisible(true);  }  LabDFS.get(++INDEX).setVisible(true);  LinArr.get(j).setStrokeWidth(4);  LinArr.get(j).setStroke(Color.GREEN);  }  else if (alg.Order.get(counter).get(2) == 2)  {  LabDFS.get(INDEX).setVisible(true);  LinArr.get(j).setStrokeWidth(4);  LinArr.get(j).setStroke(Color.YELLOW);  //INDEX--;  }  else  {  LinArr.get(j).setStrokeWidth(4);  LinArr.get(j).setStroke(Color.RED);  }  }  }  }  /\*\*  \*  \* @param counter  \* @param alg  \*/  public void hideRibs (int counter, Algorithm alg) {  //counter--;  for(int j = 0; j < LinArr.size(); j++)  {  if ((( LinArr.get(j).getStartX() == CircArr.get(alg.Order.get(counter).get(0)).getCenterX() ) &&  (LinArr.get(j).getStartY() == CircArr.get(alg.Order.get(counter).get(0)).getCenterY()) &&  (LinArr.get(j).getEndX() == CircArr.get(alg.Order.get(counter).get(1)).getCenterX()) &&  (LinArr.get(j).getEndY() == CircArr.get(alg.Order.get(counter).get(1)).getCenterY())) ||  ((( LinArr.get(j).getEndX() == CircArr.get(alg.Order.get(counter).get(0)).getCenterX() ) &&  (LinArr.get(j).getEndY() == CircArr.get(alg.Order.get(counter).get(0)).getCenterY()) &&  (LinArr.get(j).getStartX() == CircArr.get(alg.Order.get(counter).get(1)).getCenterX()) &&  (LinArr.get(j).getStartY() == CircArr.get(alg.Order.get(counter).get(1)).getCenterY()))))  {  if (LinArr.get(j).getStroke() == Color.RED)  {  //INDEX--;  LinArr.get(j).setStrokeWidth(4);  LinArr.get(j).setStroke(Color.GREEN);  }  else  {  if (LinArr.get(j).getStroke() == Color.GREEN) INDEX--; //проверить  LinArr.get(j).setStrokeWidth(1);  LinArr.get(j).setStroke(Color.BLACK);  }  }  }  }  /\*\*  \*  \*/  public void incCounter() {  if (alg.Numeration > counter + 1 ) {  if (Prev\_step.isDisable()) {  Prev\_step.setDisable(false);  }  counter++;  showRibs(counter, alg);  if (alg.Numeration <= counter + 1)  {  Next\_step.setDisable(true);  // LabDFS.get(counter).setVisible(true);  }  }  }  public void decCounter() {  if (counter >= 0) {  if (Next\_step.isDisable()) {  Next\_step.setDisable(false);  }  hideRibs(counter, alg);  counter--;  if (counter == -1) {  Prev\_step.setDisable(true);  }  }  }  /\*\*  \*  \* @param s  \*/  public void error(String s) {  Alert alert = new Alert(Alert.AlertType.ERROR);  alert.setTitle("Error Message");  alert.setHeaderText(s);  alert.showAndWait();  }  public void EGG(String s) {  Alert alert = new Alert(Alert.AlertType.WARNING);  alert.setTitle("КЕК");  alert.setHeaderText(s);  alert.showAndWait();  }  } |

Main.java

|  |
| --- |
| package sample;  /\*\*  \* Created by ДРЮС on 27.06.2016.  \*/  import javafx.application.Application;  import javafx.fxml.FXMLLoader;  import javafx.scene.Parent;  import javafx.scene.Scene;  import javafx.stage.Stage;  import javafx.scene.image.ImageView;  import javafx.scene.image.Image;  import javafx.scene.layout.HBox;  public class Main extends Application {  @Override  public void start(Stage primaryStage) throws Exception{  Parent root = FXMLLoader.load(getClass().getResource("sample\_1.fxml"));  primaryStage.setTitle("Алгоритм нахождения мостов в графе");  primaryStage.setScene(new Scene(root, 1200, 700));  primaryStage.setResizable(false ); //не будет меняться  primaryStage.show();  }  public static void main(String[] args) {  launch(args);  }  } |

Sample\_1.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <?import java.lang.\*?>  <?import javafx.geometry.\*?>  <?import javafx.geometry.Insets?>  <?import javafx.scene.\*?>  <?import javafx.scene.control.\*?>  <?import javafx.scene.control.Button?>  <?import javafx.scene.control.Label?>  <?import javafx.scene.effect.\*?>  <?import javafx.scene.image.\*?>  <?import javafx.scene.layout.\*?>  <?import javafx.scene.layout.GridPane?>  <?import javafx.scene.paint.\*?>  <?import javafx.scene.shape.\*?>  <?import javafx.scene.text.\*?>  <GridPane fx:id="Main\_stage" alignment="CENTER" blendMode="SRC\_OVER" hgap="10.0" minHeight="700.0" minWidth="1000.0" prefHeight="700.0" prefWidth="1200.0" vgap="10.0" xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/2.2" fx:controller="sample.Controller">  <children>  <ImageView id="fon5" fitHeight="1020.0" fitWidth="1980.0" pickOnBounds="true" visible="true" GridPane.columnIndex="0" GridPane.rowIndex="0">  <image>  <Image url="@fon5.png" preserveRatio="true" smooth="true" />  </image>  </ImageView>  <VBox id="" fx:id="virtual\_sex\_SMS" alignment="BOTTOM\_CENTER" minHeight="200.0" prefHeight="490.0" prefWidth="609.0001220703125" GridPane.columnIndex="0" GridPane.rowIndex="0">  <children>  <Pane fx:id="Pane1" blendMode="HARD\_LIGHT" prefHeight="210.0" prefWidth="609.0">  <children>  <Circle id="" fx:id="Circle1" blendMode="MULTIPLY" centerX="600.0" centerY="30.0" fill="WHITE" layoutX="88.0" layoutY="88.0" radius="200.0" stroke="BLACK" strokeType="INSIDE" visible="false" />  </children>  <VBox.margin>  <Insets bottom="100.0" />  </VBox.margin>  </Pane>  <HBox alignment="BOTTOM\_CENTER" maxWidth="-1.0" minHeight="45.0" padding="$x3" prefHeight="56.0" prefWidth="609.0">  <children>  <Button fx:id="Generation" blendMode="HARD\_LIGHT" cache="false" cacheHint="DEFAULT" cancelButton="false" defaultButton="false" depthTest="INHERIT" maxHeight="50.0" mnemonicParsing="false" onAction="#Generation\_kek" opacity="1.0" prefHeight="41.0" prefWidth="146.0" text="Генерация графа" textAlignment="LEFT" textFill="BLACK" textOverrun="ELLIPSIS" wrapText="false">  <effect>  <Lighting diffuseConstant="2.0" specularConstant="0.0" specularExponent="40.0" surfaceScale="0.0">  <bumpInput>  <Shadow />  </bumpInput>  <light>  <javafx.scene.effect.Light.Spot pointsAtZ="-100.0" x="100.0" y="100.0" z="100.0" />  </light>  </Lighting>  </effect>  <font>  <Font size="15.0" fx:id="x1" />  </font>  <HBox.margin>  <Insets left="50.0" />  </HBox.margin>  </Button>  <TextField id="amount" fx:id="Amount" blendMode="SRC\_OVER" editable="true" maxHeight="50.0" maxWidth="155.0" minHeight="0.0" minWidth="10.0" prefHeight="46.0" prefWidth="146.0" promptText="Введите число вершин" text="">  <HBox.margin>  <Insets left="30.0" />  </HBox.margin>  </TextField>  <VBox prefHeight="51.0" prefWidth="147.0">  <children>  <HBox minWidth="170.0" prefHeight="20.0" prefWidth="170.0">  <children>  <Label maxWidth="70.0" minHeight="20.0" minWidth="70.0" prefWidth="70.0" text="Разряженый">  <HBox.margin>  <Insets left="10.0" />  </HBox.margin>  </Label>  <Label text="Плотный">  <HBox.margin>  <Insets left="20.0" top="2.0" />  </HBox.margin>  </Label>  </children>  <VBox.margin>  <Insets fx:id="x3" />  </VBox.margin>  </HBox>  <Slider fx:id="PIMPSlide" blendMode="HARD\_LIGHT" max="1.0" maxWidth="-Infinity" min="0.0" prefWidth="100.0" showTickLabels="false" showTickMarks="false" value="0.0">  <VBox.margin>  <Insets left="30.0" />  </VBox.margin>  </Slider>  </children>  <padding>  <Insets fx:id="x3" />  </padding>  <HBox.margin>  <Insets left="20.0" top="5.0" />  </HBox.margin>  </VBox>  </children>  <VBox.margin>  <Insets bottom="20.0" fx:id="x4" />  </VBox.margin>  </HBox>  <HBox alignment="BOTTOM\_CENTER" minHeight="43.0" padding="$x3" prefHeight="110.0" prefWidth="609.0" VBox.margin="$x4">  <children>  <TextArea fx:id="Messeges" blendMode="SRC\_ATOP" disable="false" editable="false" focusTraversable="true" maxHeight="105.0" maxWidth="430.0" minHeight="100.0" opacity="0.8" prefHeight="105.0" prefRowCount="10" prefWidth="430.0" promptText="" style="" text="Область диагноститечских сообщений:&#10;" visible="true" wrapText="false">  <HBox.margin>  <Insets bottom="5.0" left="20.0" />  </HBox.margin>  </TextArea>  <VBox prefHeight="152.0" prefWidth="208.99993896484375">  <children>  <Button fx:id="Next\_step" blendMode="SCREEN" cancelButton="false" defaultButton="false" disable="true" font="$x1" mnemonicParsing="false" onAction="#incCounter" prefHeight="41.0" prefWidth="146.0" text="Следующий шаг" textFill="BLACK" />  <Button fx:id="Prev\_step" blendMode="SCREEN" defaultButton="false" disable="true" font="$x1" mnemonicParsing="false" onAction="#decCounter" prefHeight="41.0" prefWidth="146.00009999999747" text="Предыдущий шаг" textFill="BLACK" wrapText="false">  <VBox.margin>  <Insets top="20.0" />  </VBox.margin>  </Button>  </children>  <HBox.margin>  <Insets left="20.0" fx:id="x2" />  </HBox.margin>  </VBox>  </children>  </HBox>  </children>  </VBox>  </children>  <columnConstraints>  <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" />  </columnConstraints>  <rowConstraints>  <RowConstraints minHeight="10.0" vgrow="SOMETIMES" />  </rowConstraints>  </GridPane> |

# **ВЫВОД**

После выполнения данного проекта был построен механизм по генерации неориентированных связных графов разной плотности ребер. Был разработан алгоритм по поиску мостов в неориентированных связных графах, основанный на ПВГ и на ПВГ-модифицированном алгоритмах. Была создана визуализация алгоритма с возможностью пошагового отслеживания работы (добавление разного типа ребер). Шаг можно выбирать как следующий, так и предыдущий. Окно графического интерфейса было выполнено при помощи языка JAVA и средства для создания GUI JAVAFX.