**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по практике**

Тема: Визуализация алгоритма Прима на языке Java

|  |  |  |
| --- | --- | --- |
| Студенты гр. 4383 |  | Дронников И.М.  Люшнина Е.А. |
| Руководитель |  | Шолохова О.М. |

Санкт-Петербург

2016

**ЗАДАНИЕ**

**на практику**

|  |  |  |
| --- | --- | --- |
| Студенты Дронников И.М.  Люшнина Е.А. | | |
| Группа 4383 | | |
| Тема практики: Визуализация алгоритма Прима | | |
| Задание на практику:  Создать программу, отображающую пошагово действия в алгоритме Прима. | | |
| Сроки прохождения практики: 22.06.2016 – 05.07.2016 | | |
| Дата сдачи отчета: 05.07.2016 | | |
| Дата защиты отчета: 05.07.2016 | | |
|  | | |
| Студенты |  | Дронников И.М.  Люшнина Е.А. |
| Руководитель |  | Шолохова О.М. |

**содержание**

[**ВВЕДЕНИЕ** 5](#_Toc455396249)

[**ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ** 6](#_Toc455396250)

[**ОПИСАНИЕ АЛГОРИТМА** 7](#_Toc455396251)

[**СТРУКТУРА ДАННЫХ** 8](#_Toc455396252)

[**ЗАКЛЮЧЕНИЕ** 10](#_Toc455396253)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 11](#_Toc455396254)

[**ПРИЛОЖЕНИЕ А** 12](#_Toc455396255)

[**КОД ПРОГРАММЫ** 12](#_Toc455396256)

# **ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ**

Разработать программу, позволяющую визуально пошагово продемонстрировать выполнение алгоритма Прима.

Требуется пошагово выводить информацию о добавляемых ребрах, весе и обоснование шага.

# **ОПИСАНИЕ АЛГОРИТМА**

Алгоритм Прима — алгоритм поиска минимального остовного дерева во взвешенном неориентированном связном графе.

Искомый минимальный остов строится постепенно, добавлением в него рёбер по одному. Изначально остов полагается состоящим из единственной вершины (её можно выбрать произвольно). Затем выбирается ребро минимального веса, исходящее из этой вершины, и добавляется в минимальный остов. После этого остов содержит уже две вершины, и теперь ищется и добавляется ребро минимального веса, имеющее один конец в одной из двух выбранных вершин, а другой — наоборот, во всех остальных, кроме этих двух. И так далее, т.е. всякий раз ищется минимальное по весу ребро, один конец которого — уже взятая в остов вершина, а другой конец — ещё не взятая, и это ребро добавляется в остов (если таких рёбер несколько, можно взять любое). Этот процесс повторяется до тех пор, пока остов не станет содержать все вершины (или, что то же самое, n-1 ребро).

В итоге будет построен остов, являющийся минимальным. Если граф был изначально не связан, то остов найден не будет (количество выбранных рёбер останется меньше n-1).

Начинаем с пустого множества U. Добавляем к U вершины, каждый раз находя ребро наименьшей стоимости между U и V-U.

Положить в U любую вершину; // изначально U - пусто.

while ( V-U не пусто )

{

Выбрать ребро (u, v) наименьшей стоимости,

u из U, v из V-U;

Добавить v к U (и убрать из V-U);

}

Данный алгоритм имеет сложность O(V2)

# **СТРУКТУРА ДАННЫХ**

Структура данных, используемая в программе представляет собой связный неориентированный граф.

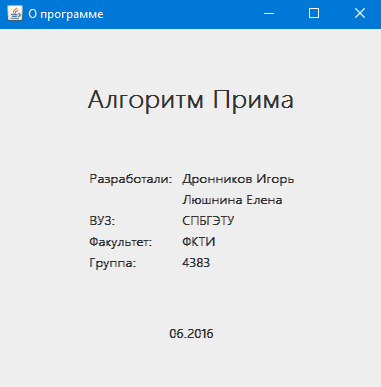
Данные для заполнения графа хранятся в текстовом файле.

Ограничения:

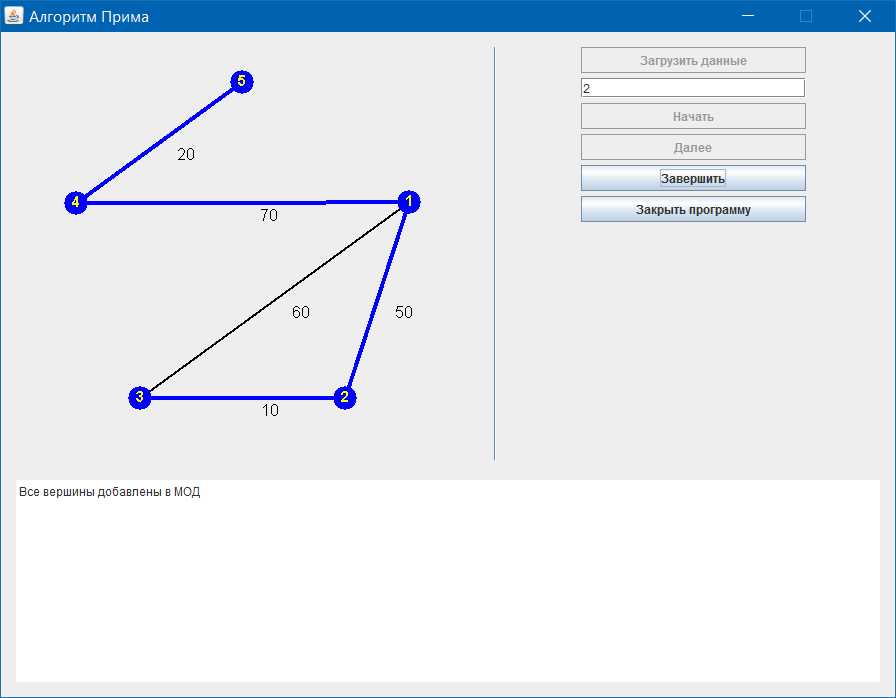
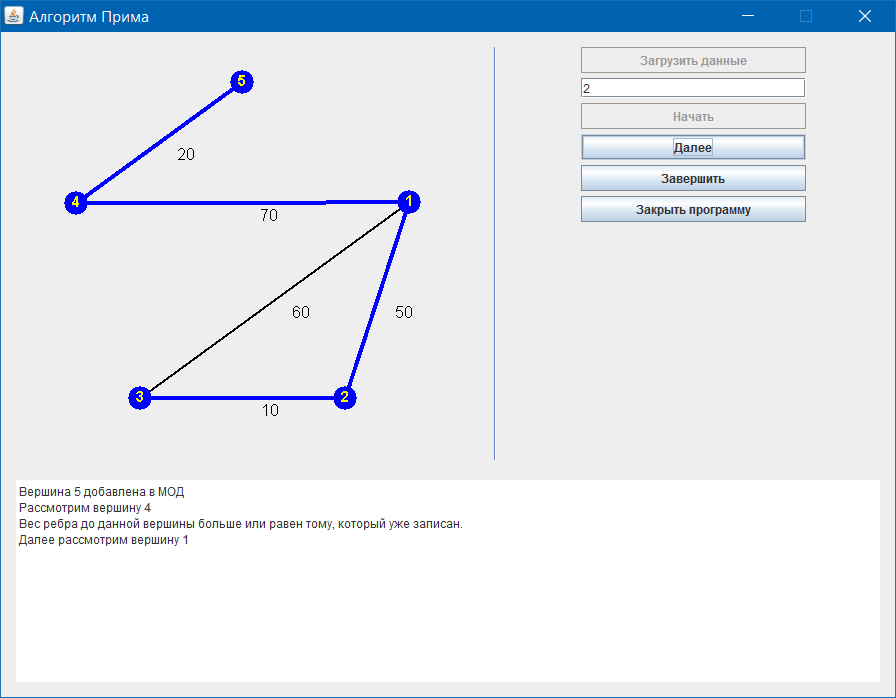
* Максимальное количество вершин в графе – 40
* Максимальный вес ребер – 1000

**ТЕСТИРОВАНИЕ**

# 



# 



# **ЗАКЛЮЧЕНИЕ**

В ходе работы над данным проектом были получены практические навыки по работе с языком Java. Так же изучена библиотека для создания графического интерфейса Swing. Программа, отвечающая поставленным требованиям и выполняющая поставленную задачу разработана успешно.

# **ПРИЛОЖЕНИЕ А**

## **КОД ПРОГРАММЫ**

Algorithm.java

*/\*\*  
 \* Created by Lyushnina Elena on 26.06.2016.  
 \*/***import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.util.Scanner;  
  
*/\*\*  
 \* Класс Algorithm реализует алгоритм Прима.  
 \* Он поддерживает следующие операции: считывание графа,  
 \* readData, подготовку к выполнению алгоритма, start  
 \* и выполнение шага алгоритма, executeStepAlgorithm.  
 \*/***public class** Algorithm {  
 **private final static int *MAX\_INT*** = 1000;  
 **private** Graph **G**; *// исходный граф* **private int startVertex**; *// начальная вершина* **private int numVertices**; *// количество вершин* **private int**[] **parent**;  
 **private boolean**[] **intree**; *// вершина в дереве?* **private int**[] **distance**; *// стоимость добавления к дереву* **private** Graph.Edge **p**; *// временный указатель* **private int v**; *// текущая вершина для обработки* **private** String **textInfo**;  
  
 */\*\*  
 \*  
 \*/* **public** Algorithm() {  
 **G** = **new** Graph();  
 **startVertex** = 0;  
  
 **intree** = **new boolean**[**G**.**edges**.**length**];  
 **distance** = **new int**[**G**.**edges**.**length**];  
 **parent** = **new int**[**G**.**edges**.**length**];  
  
 **textInfo** = **""**;  
 }  
 **public int** getMAXV(){**return G**.***MAX\_V***;}  
 **public int**[] getParent() {  
 **return parent**;  
 }  
  
 **public** Graph.Edge getEdge(**int** j) {**return G**.**edges**[j];}  
  
 **public int** getNumVertices() {**return numVertices**;}  
  
 **public int** getLength() {  
 **return G**.**edges**.**length**;  
 }  
  
  
 **public** String getTextInfo() {  
 **return textInfo**;  
 }  
  
 */\*\*  
 \* Считывание данных из файла  
 \*/* **public void** readData() {  
 File f = **new** File(**"input.txt"**);  
 **try** {  
 Scanner scan = **new** Scanner(f);  
  
 **numVertices** = scan.nextInt();  
 **if** (**numVertices** <= 0) **throw new** IllegalArgumentException();  
  
 **for** (**int** i = 0; i < **numVertices**; ++i) {  
 **int** vertex\_x = scan.nextInt();  
 **if** (vertex\_x <= 0) **throw new** IllegalArgumentException();  
  
 **int** numEdges = scan.nextInt();  
 **if** (numEdges < 0) **throw new** IllegalArgumentException();  
  
 **for** (**int** j = 0; j < numEdges; ++j) {  
  
 **int** vertex\_y = scan.nextInt();  
 **if** (vertex\_y <= 0) **throw new** IllegalArgumentException();  
  
 **int** \_weight = scan.nextInt();  
 **if** (\_weight <= 0) **throw new** IllegalArgumentException();  
  
 **G**.addEdge(vertex\_x, vertex\_y, \_weight);  
 }  
 }  
 System.***out***.println(**"Данные считаны успешно"**);  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Подготовка к выполнению алгоритма, которая  
 \* заключается в инициализации необходимых полей  
 \* начальными значениями.  
 \*  
 \** ***@param s*** *начальная вершина  
 \*/* **public void** start(**int** s) {  
 **startVertex** = s;  
 **for** (**int** i = 1; i <= **numVertices**; ++i) {  
 **intree**[i] = **false**;  
 **distance**[i] = ***MAX\_INT***;  
 **parent**[i] = -1;  
 }  
  
 **distance**[**startVertex**] = 0;  
 **v** = **startVertex**;  
 }  
  
 */\*\*  
 \* Выполняет шаг алгоритма  
 \*  
 \** ***@return*** *false, если еще не все вершины добавлены в граф;  
 \* в противном случае - true  
 \*/* **public boolean** executeStepAlgorithm() {  
 **textInfo** = **""**;  
 **if** (!**intree**[**v**]) {  
 **int** w; *// кондидат на следующую вершину* **int** weight; *// вес ребра* **int** dist; *// наилучшее текущее расстояние от начала* **textInfo** += **"Вершина "** + **v** + **" добавлена в МОД\n"**;  
  
 **intree**[**v**] = **true**;  
 **p** = **G**.**edges**[**v**];  
 **while** (**p** != **null**) {  
 w = **p**.getY();  
 **if** (w != 0) {  
 **textInfo** += **"Рассмотрим вершину "** + w + **"\n"**;  
 weight = **p**.getWeight();  
 **if** (**distance**[w] > weight && !**intree**[w]) {  
 **distance**[w] = weight;  
 **parent**[w] = **v**;  
 **textInfo** += **"Вес ребра до данной вершины меньше чем тот, который уже записан и эта вершина еще не добавлена в дерево. Заменяем вес на новый.\n"**;  
 } **else** {  
 **if** (**distance**[w] > weight)  
 **textInfo** += **"Вес ребра до данной вершины больше или равен тому, который уже записан.\n"**;  
 **else textInfo** += **"Данная вершина уже включена в МОД.\n"**;  
 }  
 }  
 **p** = **p**.getNext();  
 }  
 **v** = 1;  
 dist = ***MAX\_INT***;  
 **for** (**int** i = 1; i <= **numVertices**; ++i) {  
 **if** (!**intree**[i] && dist > **distance**[i] && **parent**[i] != -1) {  
 dist = **distance**[i];  
 **v** = i;  
 }  
 }  
 **textInfo** += **"Далее рассмотрим вершину "** +**v**;  
 **return false**;  
 } **else** {  
 **textInfo** += **"Все вершины добавлены в МОД\n"**;  
 **return true**;  
 }  
 }  
  
 **public void** clear(){  
 **G**.clear();  
 **startVertex** = 0;  
 **numVertices** = 0;  
 **for** (**int** i = 0; i < **parent**.**length**; ++i){  
 **parent**[i] = -1;  
 **intree**[i] = **false**;  
 **distance**[i] = ***MAX\_INT***;  
 }  
 **p** = **null**;  
 **v** = 0;  
  
 **textInfo** = **""**;  
 }  
}

Graph.java

*/\*\*  
 \* Created by Lyushnina Elena on 24.06.2016.  
 \*/  
  
/\*\*  
 \* Класс Graph представляет собой ориетированный граф, для  
 \* реализации которого использовалась обощенная коллекция ArrayList.  
 \* В ней содержатся элементы вложенного класса Edge, представляющего  
 \* из себя направленное ребро графа.  
 \* Класс поддерживает операции добавления ребра и очистки графа.  
 \* Операция clear работает за константное время. Операция addEdge работает  
 \* работает за время O(V).  
 \*  
 \** ***@author*** *Lyushnina Elena  
 \*/***public class** Graph **implements** Graph\_interface {  
 **protected final static int *MAX\_V*** = 40;  
  
 **public static class** Edge {  
 **private int x**;  
 **private int y**;  
 **private int weight**;  
 **private** Edge **next**;  
  
 **public** Edge(**int** \_x, **int** \_y, **int** \_weight) {  
 **x** = \_x;  
 **y** = \_y;  
 **weight** = \_weight;  
 **next** = **null**;  
 }  
  
 **public int** getX() {  
 **return x**;  
 }  
  
 **public int** getY() {  
 **return y**;  
 }  
  
 **public int** getWeight() {  
 **return weight**;  
 }  
  
 **public** Edge getNext() {  
 **return next**;  
 }  
 }  
  
 **public** Edge[] **edges**;  
  
 */\*\*  
 \* Инициализирует пустой граф  
 \*/* **public** Graph() {  
 **edges** = **new** Edge[***MAX\_V*** + 1];  
 clear();  
 }  
  
 **public void** clear() {  
 **for** (**int** i = 0; i < **edges**.**length**; ++i) {  
 Edge e = **new** Edge(0, 0, 0);  
 **edges**[i] = e;  
 }  
 }  
  
 **public void** addEdge(**int** \_x, **int** \_y, **int** \_weight) {  
 Edge e = **new** Edge(\_x, \_y, \_weight);  
  
 e.**next** = **edges**[\_x];  
 **edges**[\_x] = e;  
  
 Edge \_e = **new** Edge(\_y, \_x, \_weight);  
  
 \_e.**next** = **edges**[\_y];  
 **edges**[\_y] = \_e;  
 }  
}

Graph\_inerface.java

*/\*\*  
 \* Created by Lyushnina Elena on 25.06.2016.  
 \*/***public interface** Graph\_interface {  
 */\*\*  
 \* Удаляет все вершины и ребра из графа.  
 \*/* **void** clear();  
  
 */\*\*  
 \* Добавляет ребро в граф  
 \*  
 \** ***@param \_x*** *начало дуги  
 \** ***@param \_y*** *конец дуги  
 \** ***@param \_weight*** *вес дуги  
 \*/* **void** addEdge(**int** \_x, **int** \_y, **int** \_weight);  
}

MainActivity.java

*/\*\*  
 \* Created by dronn on 26.06.2016.  
 \*/***public class** MainActivity {  
 **public static void** main(String[] args){  
  
 MainWindow main\_window = **new** MainWindow();  
 }  
}

AboutWindow.java

**import** javax.swing.\*;  
  
*/\*\*  
 \* Created by dronn on 03.07.2016.  
 \*/***public class** AboutWindow **extends** JFrame{  
 **private** JPanel **rootPanel**;  
  
 */\*\*  
 \* Конструктор окна О программе  
 \*/* **public** AboutWindow(){  
 **super**(**"О программе"**);  
 setContentPane(**rootPanel**);  
  
 setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  
  
 setSize(400,400);  
 setVisible(**true**);  
 }  
}

MainWindow.java

**import** javax.swing.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
  
*/\*\*  
 \* Created by dronn on 26.06.2016.  
 \*/***public class** MainWindow **extends** JFrame {  
 **private** JPanel **rootPanel**;  
 **private** JButton **startButton**; *//Начать программу* **private** JButton **aboutButton**; *//О программе* **private** JButton **exitButton**; *//Вывод  
  
 /\*\*  
 \* Конструктор начального окна  
 \*  
 \*/* **public** MainWindow() {  
 **super**(**"Алгоритм Прима"**);  
 setContentPane(**rootPanel**);  
 setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 ActionListener startActionListener = **new** StartButtonActionListener();  
 ActionListener aboutActionListener = **new** AboutButtonActionListener();  
 ActionListener exitActionListener = **new** ExitButtonActionListener();  
  
 **startButton**.addActionListener(startActionListener);  
 **aboutButton**.addActionListener(aboutActionListener);  
 **exitButton**.addActionListener(exitActionListener);  
  
 setSize(500,500);  
 setVisible(**true**);  
 }  
  
 **public class** StartButtonActionListener **implements** ActionListener {  
 */\*\*  
 \* Обработчик действия кнопки startButton  
 \*  
 \** ***@param e*** *- startActionListener  
 \*/* **public void** actionPerformed(ActionEvent e) {  
 PrimAlg alg\_window = **new** PrimAlg();  
 }  
 }  
  
 **public class** AboutButtonActionListener **implements** ActionListener {  
 */\*\*  
 \* Обработчик действия кнопки aboutButton  
 \*  
 \** ***@param e*** *- aboutActionListener  
 \*/* **public void** actionPerformed(ActionEvent e) {  
 AboutWindow about\_window = **new** AboutWindow();  
 }  
 }  
  
 **public class** ExitButtonActionListener **implements** ActionListener {  
 */\*\*  
 \* Обработчик действия кнопки exitButton  
 \*  
 \** ***@param e*** *- exitActionListener  
 \*/* **public void** actionPerformed(ActionEvent e) {  
 System.*exit*(1);  
 }  
 }  
}

PrimAlg.java

**import** java.awt.\*;  
**import** javax.swing.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
  
*/\*\*  
 \* Created by dronn on 30.06.2016.  
 \*/***public class** PrimAlg **extends** JFrame {  
 **private** JPanel **rootPanel**;  
 **private** JButton **loadButton**; *//Загрузить данные* **private** JButton **startButton**; *//Начать* **private** JButton **nextButton**; *//Далее* **private** JButton **finishButton**; *//Завершить* **private** JTextField **startParam**; *//Начальная вершина* **private** JTextPane **outputText**; *//Информационная панель* **private** JButton **exitButton**; *//Закрыть окно* **private** Algorithm **prim**; *//Алгоритм* **private int startParamNum**; *//Начальная вершина - парамент* **private int windowSizeH** = 700; *//Высота окна* **private int windowSizeW** = 900; *//Шинира окна  
  
 /\*\*  
 \* Конструктор окна с визуализацией алгоритма  
 \*  
 \*/* **public** PrimAlg() {  
 **super**(**"Алгоритм Прима"**);  
 setContentPane(**rootPanel**);  
  
 setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  
 **prim** = **new** Algorithm();  
  
 ActionListener loadActionListener = **new** LoadButtonActionListener();  
 ActionListener startActionListener = **new** StartButtonActionListener();  
 ActionListener nextActionListener = **new** NextButtonActionListener();  
 ActionListener finishActionListener = **new** FinishButtonActionListener();  
 ActionListener exitActionListener = **new** ExitButtonActionListener();  
  
 **loadButton**.addActionListener(loadActionListener);  
 **startButton**.addActionListener(startActionListener);  
 **nextButton**.addActionListener(nextActionListener);  
 **finishButton**.addActionListener(finishActionListener);  
 **exitButton**.addActionListener(exitActionListener);  
  
 **startButton**.setEnabled(**false**);  
 **nextButton**.setEnabled(**false**);  
 **finishButton**.setEnabled(**false**);  
  
 setSize(**windowSizeW**, **windowSizeH**);  
 setResizable(**false**);  
 setVisible(**true**);  
 }  
  
 */\*\*  
 \* Визуализация основной структуры данных  
 \*  
 \*/* **public void** paintGraph(){  
 clear();  
 Graphics2D gfx = (Graphics2D)**rootPanel**.getGraphics();  
 gfx.setColor(Color.***black***);  
  
 **double** pi=3.14;  
 **double** angle = 2\*pi /(**prim**.getNumVertices());  
 **int** r=**windowSizeH**/4,  
 u=**windowSizeW**/5+50,  
 v=**windowSizeH**/3-20,  
 rPoint = 24;  
  
 *//отрисовка ребер графа* Font SegoiUI = **new** Font(**"Segoi UI Light"**, Font.***PLAIN***, 17);  
 gfx.setFont(SegoiUI);  
 gfx.setStroke(**new** BasicStroke(2.0f)); *// толщина равна 2* **for** (**int** j = 1; j < **prim**.getLength(); ++j){  
 Graph.Edge e = **prim**.getEdge(j);  
 **while** (e != **null**) {  
 **if** (e.getX()!=0 && e.getY()!=0) {  
 **double** x1 = r \* Math.*cos*(angle \* e.getX() - pi / 2) + u + rPoint / 2;  
 **double** y1 = r \* Math.*sin*(angle \* e.getX() - pi / 2) + v + rPoint / 2;  
 **double** x2 = r \* Math.*cos*(angle \* e.getY() - pi / 2) + u + rPoint / 2;  
 **double** y2 = r \* Math.*sin*(angle \* e.getY() - pi / 2) + v + rPoint / 2;  
  
 gfx.drawLine((**int**)x1, (**int**)y1, (**int**)x2, (**int**)y2);  
  
 x1 = (x1 + x2) / 2 + 10;  
 y1 = (y1 + y2) / 2 + 10;  
 gfx.drawString(String.*valueOf*(e.getWeight()), (**int**)x1 + rPoint/3, (**int**)y1 + rPoint/3);  
 }  
 e = e.getNext();  
 }  
 }  
  
 *//отрисовка вершин* SegoiUI = **new** Font(**"Segoi UI Light"**, Font.***BOLD***, 15);  
 gfx.setFont(SegoiUI);  
 **for** (**int** j = 1; j<=**prim**.getNumVertices(); j++){  
 gfx.setColor(Color.***black***);  
 **double** x = r\*Math.*cos*(angle\*(j)-pi/2) + u;  
 **double** y = r\*Math.*sin*(angle\*(j)-pi/2) + v;  
 gfx.fillOval((**int**)x, (**int**)y, rPoint, rPoint); *// рисуем круг* gfx.setColor(Color.***red***);  
 gfx.drawString(String.*valueOf*(j), (**int**)x+ rPoint/3, (**int**)y+ rPoint/3\*2);  
 }  
 }  
  
 */\*\*  
 \* Визуализация каждого шага алгортима  
 \*  
 \*/* **public void** paintStepGraph(){  
 Graphics2D gfx = (Graphics2D)**rootPanel**.getGraphics();  
 gfx.setStroke(**new** BasicStroke(4.0f)); *// толщина равна 4* Font SegoiUI = **new** Font(**"Segoi UI Light"**, Font.***BOLD***, 15);  
 gfx.setFont(SegoiUI);  
 **double** pi=3.14;  
 **double** angle = 2\*pi /(**prim**.getNumVertices());  
 **int** r=**windowSizeH**/4,  
 u=**windowSizeW**/5+50,  
 v=**windowSizeH**/3-20,  
 rPoint = 24;  
  
 **int** parent[] = **prim**.getParent();  
 **for** (**int** j = 1; j < **prim**.getMAXV(); ++j) {  
 gfx.setColor(Color.***blue***);  
 **if** (parent[j] != -1 && parent[j] != 0) {  
 *//отрисовка ребер* **double** x1 = r \* Math.*cos*(angle \* parent[j] - pi / 2) + u + rPoint / 2;  
 **double** y1 = r \* Math.*sin*(angle \* parent[j] - pi / 2) + v + rPoint / 2;  
 **double** x2 = r \* Math.*cos*(angle \* j - pi / 2) + u + rPoint / 2;  
 **double** y2 = r \* Math.*sin*(angle \* j - pi / 2) + v + rPoint / 2;  
  
 gfx.drawLine((**int**) x1, (**int**)y1, (**int**)x2, (**int**)y2);  
  
 *//отрисовка двух добавленых вершин* **double** x = r \* Math.*cos*(angle \* parent[j] - pi / 2) + u;  
 **double** y = r \* Math.*sin*(angle \* parent[j] - pi / 2) + v;  
 gfx.fillOval((**int**)x, (**int**)y, rPoint, rPoint); *// рисуем круг* gfx.setColor(Color.***yellow***);  
 gfx.drawString(String.*valueOf*(parent[j]), (**int**)x+ rPoint/3, (**int**)y+ rPoint/3\*2);  
  
 gfx.setColor(Color.***blue***);  
 x = r \* Math.*cos*(angle \* j - pi / 2) + u;  
 y = r \* Math.*sin*(angle \* j - pi / 2) + v;  
 gfx.fillOval((**int**)x, (**int**)y, rPoint, rPoint); *// рисуем круг* gfx.setColor(Color.***yellow***);  
 gfx.drawString(String.*valueOf*(j), (**int**)x+ rPoint/3, (**int**)y+ rPoint/3\*2);  
 }  
 }  
 }  
  
 */\*\*  
 \* Очистка области для вывода  
 \*  
 \*/* **public void** clear(){  
 Graphics2D gfx = (Graphics2D)**rootPanel**.getGraphics();  
 Color color\_gray = **new** Color(238,238,238);  
 gfx.setColor(color\_gray);  
 gfx.fillRect(0,0,**windowSizeW**/4\*2,**windowSizeH**/5\*3);  
 }  
  
  
 **public class** LoadButtonActionListener **implements** ActionListener {  
 */\*\*  
 \* Обработчик действия кнопки loadButton  
 \*  
 \** ***@param e*** *- loadActionListener  
 \*/* **public void** actionPerformed(ActionEvent e) {  
 **prim**.readData();  
 **loadButton**.setEnabled(**false**);  
 **startButton**.setEnabled(**true**);  
 **finishButton**.setEnabled(**true**);  
 paintGraph();  
 }  
 }  
 **public class** StartButtonActionListener **implements** ActionListener {  
 */\*\*  
 \* Обработчик действия кнопки startButton  
 \*  
 \** ***@param e*** *- startActionListener  
 \*/* **public void** actionPerformed(ActionEvent e) {  
 **startParamNum** = Integer.*parseInt*(**startParam**.getText());  
 **prim**.start(**startParamNum**);  
 paintGraph();  
 **nextButton**.setEnabled(**true**);  
 **startButton**.setEnabled(**false**);  
 }  
 }  
 **public class** NextButtonActionListener **implements** ActionListener {  
 */\*\*  
 \* Обработчик действия кнопки nextButton  
 \*  
 \** ***@param e*** *- nextActionListener  
 \*/* **public void** actionPerformed(ActionEvent e) {  
 **boolean** check = **prim**.executeStepAlgorithm();  
 paintGraph();  
 paintStepGraph();  
 **if** (check) **nextButton**.setEnabled(**false**);  
 **outputText**.setText(**prim**.getTextInfo());  
 }  
 }  
 **public class** FinishButtonActionListener **implements** ActionListener {  
 */\*\*  
 \* Обработчик действия кнопки finishButton  
 \*  
 \** ***@param e*** *- finishActionListener  
 \*/* **public void** actionPerformed(ActionEvent e) {  
 **prim**.clear();  
 **loadButton**.setEnabled(**true**);  
 **startButton**.setEnabled(**false**);  
 **nextButton**.setEnabled(**false**);  
 **finishButton**.setEnabled(**false**);  
 clear();  
 **outputText**.setText(**""**);  
 }  
 }  
 **public class** ExitButtonActionListener **implements** ActionListener {  
 */\*\*  
 \* Обработчик действия кнопки exitButton  
 \*  
 \** ***@param e*** *- exitActionListener  
 \*/* **public void** actionPerformed(ActionEvent e) {  
 dispose();  
 }  
 }  
}