ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

**РАСЧЁТНО-ГРАФИЧЕСКОЕ ЗАДАНИЕ**

по дисциплине “Защита информации” на тему

**Доказательство с нулевым знанием**

Вариант №1

|  |  |
| --- | --- |
| Выполнил студент | Тимофеев Дмитрий Александрович |
|  | Ф.И.О. |

|  |  |
| --- | --- |
| Группы | ИВ – 622 |

|  |  |  |
| --- | --- | --- |
| Работу принял |  | ассистент кафедры ПМиК Я. В. Петухова |
|  | | |

|  |  |
| --- | --- |
| Защищена | Оценка |

Новосибирск – 2019

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_bookmark0)

[ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 4](#_bookmark1)

[ДЕМОНСТРАЦИЯ 5](#_TOC_250000)

[ПРИЛОЖЕНИЕ 7](#_bookmark4)

ПОСТАНОВКА ЗАДАЧИ

В рамках расчётно-графического задания необходимо разработать программу, реализующую протокол доказательства с нулевым знанием для задачи «Раскраска графа».

Абонент Алиса, связывающаяся с другим абонентом Бобом, перед доказательством, должен иметь связный граф с раскрашенными вершинами. Раскраска считается правильной, если любые смежные вершины имели разные цвета. Алисе необходимо доказать Бобу, что раскраска графа правильная, без раскрытия сведений о раскраске Бобу.

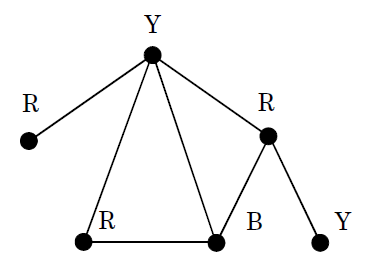
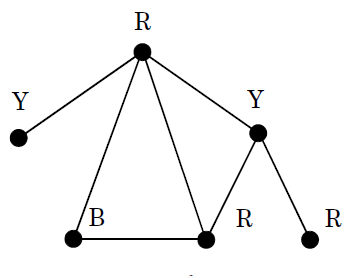
 

Рис.1 Правильная раскраска Рис.2 Неправильная раскраска

Перед реализацией протокола необходимо реализовать дополнительную программу для генерации графа. Граф случайно генерируется в программе (для определенности задается начальное зерно случайности) и записывается в файл в формате для программы GraphVis:

1. Граф содержит информацию о вершинах графа (ID и Цвет)
2. Граф содержит информацию о ребрах графа

Для реализации протокола доказательства используются параметры RSA для всех вершин, которые передаются Бобу по верифицированному каналу. В работе должны быть использованы такие процедуры, как быстрое возведение в степень, расширенный алгоритм Евклида, а также генерация больших простых чисел.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Алиса выбирает случайную перестановку имеющихся цветов и перекрашивает граф. Это не меняет корректность графа, так смежность вершин не меняется.

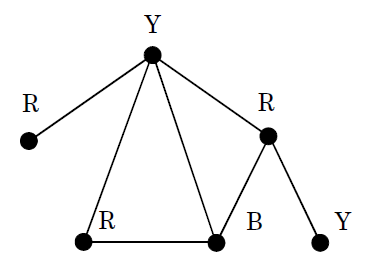
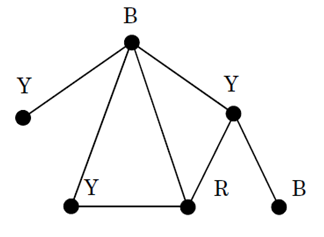
 

Рис.3 Исходный граф Рис.4 Граф после перестановки

1. Алиса для каждой вершины графа генерирует случайной большое число *r* и заменяет в нем младшие разряды кодом цвета. Например: 00 – R (красный), 01 – B (синий), 10 – Y (жёлтый).
2. Алиса для каждой вершины графа формирует данные из системы RSA: , и .
3. Алиса вычисляет

и посылает Бобу публичные ключи для каждой вершины графа.

1. Боб выбирает случайное ребро графа и сообщает его Алисе. В ответ Алиса передает Бобу ключи и . Боб вычисляет

и

и сравнивает младшие разряды, соответствующие кодам цвета вершины. Если значения совпали, то Алиса обманывает Боба, на этом этапе все заканчивается. Если нет, то возвращаемся на *Шаг 1*, до тех пор, пока все ребра графа не будут проверены.

ДЕМОНСТРАЦИЯ

Визуализация перекраски графа из 5 вершин и 9 ребер.

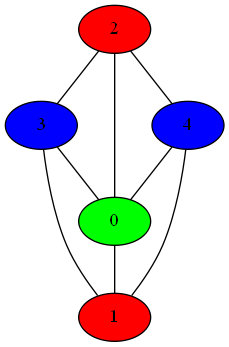
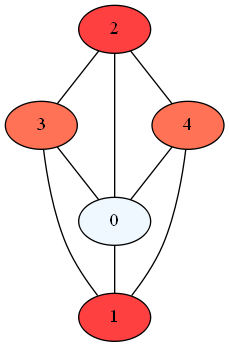


Рис. 5 Граф исходный Рис. 6 Граф после перекраски

Сохраненный граф в файле grath.dot с разметкой GraphVis.



Рис. 7 Файл grath.dot

Демонстрация работы программы

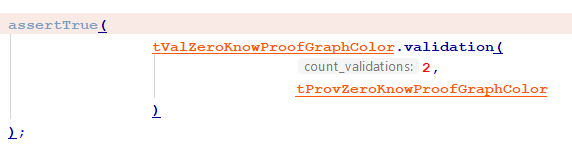


Рис. 9 Код проверки

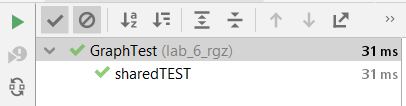


Рис. 10 вывод программы.

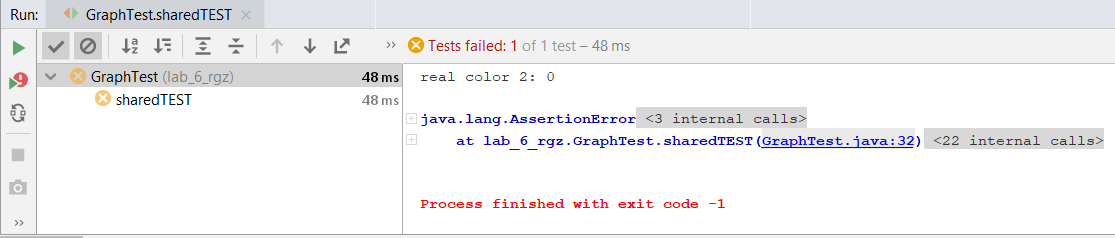


Рис. 11 Демонстрация неудачного выполнения программы (искусственно нарушена раскраска графа).

ПРИЛОЖЕНИЕ

**Класс Graph:**

package lab\_6\_rgz;

import javafx.util.Pair;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import java.util.Random;

import static java.lang.Math.abs;

public class Graph {

public Integer[] nodesArrInteger = null;

// public Map<Integer, Integer> edgesMapIntInt = null;

public ArrayList<Pair<Integer, Integer>> edgesArrayListPairIntInt = null;

public Graph(Integer[] nodesArrInteger, ArrayList<Pair<Integer, Integer>> edgesArrayListPairIntInt) {

this.nodesArrInteger = nodesArrInteger;

this.edgesArrayListPairIntInt = edgesArrayListPairIntInt;

}

static public Graph generate\_random\_graph(int count\_of\_nodes) {

//создаем массив вершин

Integer[] nodesArrBI = new Integer[count\_of\_nodes];

int max\_count\_edges = count\_of\_nodes \* count\_of\_nodes / 2;

Random tRandom = new Random();

tRandom.setSeed(134);//FIXME

int count\_edges = abs(tRandom.nextInt() % max\_count\_edges);

// int count\_edges = abs(max\_count\_edges);

// Map<Integer, Integer> edgesMapIntInt = new HashMap<Integer, Integer>();

ArrayList<Pair<Integer, Integer>> edgesArrayListPairIntInt = new ArrayList<Pair<Integer, Integer>>();

for (int i = 0; i < count\_edges; i++) {

// tRandom.setSeed(System.nanoTime() + tRandom.nextInt());

int from\_node = abs(tRandom.nextInt() % count\_of\_nodes);

int to\_node = abs(tRandom.nextInt() % count\_of\_nodes);

if (from\_node == to\_node) {

to\_node++;

to\_node = to\_node % count\_of\_nodes;

}

edgesArrayListPairIntInt.add(new Pair<Integer, Integer>(from\_node, to\_node));

}

return new Graph(nodesArrBI, edgesArrayListPairIntInt);

}

public void print\_to\_file\_dot(File outputFile) throws IOException {

FileWriter fileOut = new FileWriter(outputFile);

fileOut.write("strict graph {\n node [label=\"\\N\", style=filled];");

for (int i = 0; i < this.nodesArrInteger.length; i++) {

int color\_number = 0;

if (this.nodesArrInteger[i] != null) {

color\_number = this.nodesArrInteger[i] % this.title\_color\_\_ArrStr.length; //FIXME NEVER

color\_number \*= 17;

color\_number += 1;

color\_number \*= color\_number;

color\_number = color\_number % this.title\_color\_\_ArrStr.length;

}

String colorStr = this.title\_color\_\_ArrStr[color\_number];

fileOut.write(" " + i + "\t[fillcolor=" + colorStr + "];\n");

}

for (Pair<Integer, Integer> entry : this.edgesArrayListPairIntInt) {

Integer keyInt = entry.getKey();

Integer valueInt = entry.getValue();

fileOut.write(" " + keyInt + " -- " + valueInt + "\n");

}

fileOut.write("\n}");

fileOut.close();

}

static public void create\_image(String dotFile, String out\_image\_\_File) throws IOException, InterruptedException {

/\* //здесь "sleep 15" и есть ваша консольная команда

// Process proc = Runtime.getRuntime().exec("d:");

// Process proc = Runtime.getRuntime().exec(new String[] { "C:\\Windows\\System32\\cmd.exe"});

// Process proc = Runtime.getRuntime().exec(new String[] { "cp -help"});

// Process proc2 = Runtime.getRuntime().exec("dot -Tpng " + dotFile + " -o " + out\_image\_\_File);

// Process proc = Runtime.getRuntime().exec("sleep 15");

// Process proc = Runtime.getRuntime().exec("cp -help");

Process proc = Runtime.getRuntime().exec("c:");

proc.waitFor();

proc.destroy();\*/ //FIXME T:FUTURE

}

public void coloring\_graph() {

int current\_color = 0;

boolean exists\_no\_color\_nodes\_\_BOOL;

do {

exists\_no\_color\_nodes\_\_BOOL = false;

for (int i = 0; i < this.nodesArrInteger.length; i++) {

boolean this\_node\_coloring\_\_BOOL = this.nodesArrInteger[i] != null;

if (!this\_node\_coloring\_\_BOOL) {

if (exists\_are\_links\_to\_nodes\_of\_the\_same\_color(i, current\_color)) {

//пропускаем и отмечаем пропуск

exists\_no\_color\_nodes\_\_BOOL = true;

} else {

this.nodesArrInteger[i] = current\_color;

// this.nodesArrInteger[i] = 0; //DEBUG\_DELETE

}

}

}

current\_color++;

} while (exists\_no\_color\_nodes\_\_BOOL);

this.count\_color = current\_color;

}

public int count\_color = 0;

private boolean exists\_are\_links\_to\_nodes\_of\_the\_same\_color(int current\_pos, int current\_color) {

for (int j = 0; j < current\_pos; j++) {

if (this.nodesArrInteger[j] == null) {

continue;

}

//проверяем только окрашенные узлы, бесцетные узлы разумеется совпадать не будут по цвету

int color\_in\_j\_node = this.nodesArrInteger[j];

if (color\_in\_j\_node == current\_color) {

Pair<Integer, Integer> findPairIntInt = new Pair<>(current\_pos, j);

Pair<Integer, Integer> find\_reverse\_\_PairIntInt = new Pair<>(j, current\_pos);

boolean isset\_edge = this.edgesArrayListPairIntInt.contains(findPairIntInt)

|| this.edgesArrayListPairIntInt.contains(find\_reverse\_\_PairIntInt);

if (isset\_edge) {

return true;

}

}

}

return false;

}

static private String[] title\_color\_\_ArrStr = {

"white",

"aliceblue",

"antiquewhite",

"antiquewhite1",

"antiquewhite2",

"antiquewhite3",

"antiquewhite4",

"aquamarine",

"aquamarine1",

"aquamarine2",

"aquamarine3",

"aquamarine4",

"azure",

"azure1",

"azure2",

"azure3",

"azure4",

"beige",

"bisque",

"bisque1",

"bisque2",

"bisque3",

"bisque4",

"black",

"blanchedalmond",

"blue ",

"blue1",

"blue2",

"blue3",

"blue4",

"blueviolet",

"brown",

"brown1",

"brown2",

"brown3",

"brown4",

"burlywood",

"burlywood1",

"burlywood2",

"burlywood3",

"burlywood4",

"cadetblue",

"cadetblue1",

"cadetblue2",

"cadetblue3",

"cadetblue4",

"chartreuse",

"chartreuse1",

"chartreuse2",

"chartreuse3",

"chartreuse4",

"chocolate",

"chocolate1",

"chocolate2",

"chocolate3",

"chocolate4",

"coral",

"coral1",

"coral2",

"coral3",

"coral4",

"cornflowerblue",

"cornsilk",

"cornsilk1",

"cornsilk2",

"cornsilk3",

"cornsilk4",

"crimson",

"cyan",

"cyan1",

"cyan2",

"cyan3",

"cyan4",

"darkgoldenrod",

"darkgoldenrod1",

"darkgoldenrod2",

"darkgoldenrod3",

"darkgoldenrod4",

"darkgreen",

"darkkhaki",

"darkolivegreen",

"darkolivegreen1",

"darkolivegreen2",

"darkolivegreen3",

"darkolivegreen4",

"darkorange",

"darkorange1",

"darkorange2",

"darkorange3",

"darkorange4",

"darkorchid",

"darkorchid1",

"darkorchid2",

"darkorchid3",

"darkorchid4",

"darksalmon",

"darkseagreen",

"darkseagreen1",

"darkseagreen2",

"darkseagreen3",

"darkseagreen4",

"darkslateblue",

"darkslategray",

"darkslategray1",

"darkslategray2",

"darkslategray3",

"darkslategray4",

"darkslategrey",

"darkturquoise",

"darkviolet",

"deeppink",

"deeppink1",

"deeppink2",

"deeppink3",

"deeppink4",

"deepskyblue",

"deepskyblue1",

"deepskyblue2",

"deepskyblue3",

"deepskyblue4",

"dimgray",

"dimgrey",

"dodgerblue",

"dodgerblue1",

"dodgerblue2",

"dodgerblue3",

"dodgerblue4",

"firebrick",

"firebrick1",

"firebrick2",

"firebrick3",

"firebrick4",

"floralwhite",

"forestgreen",

"gainsboro",

"ghostwhite",

"gold",

"gold1",

"gold2",

"gold3",

"gold4",

"goldenrod",

"goldenrod1",

"goldenrod2",

"goldenrod3",

"goldenrod4"

};

}

**Класс ProvingZeroKnowledgeProofGraphColoring:**

package lab\_6\_rgz;

import javafx.util.Pair;

import lab\_1.MyBigInteger;

import lab\_1.PrimeNumber;

import java.math.BigInteger;

import java.util.ArrayList;

import java.util.Random;

import static java.lang.Math.abs;

public class ProvingZeroKnowledgeProofGraphColoring {

BigInteger global\_max\_\_BI = BigInteger.valueOf(10000);

Graph coloringGraph;

Random sharedRandom = new Random();

public int size\_bit\_write;

private BigInteger[] cArrBI = null;

public int getSize\_bit\_write() {

return size\_bit\_write;

}

public ProvingZeroKnowledgeProofGraphColoring(Graph coloringGraph) {

this.coloringGraph = coloringGraph;

sharedRandom.setSeed(13);

}

public BigInteger[][] get\_crypt\_graph() throws Exception {

int[] mix\_\_Arr\_int = mixing\_array(this.coloringGraph.count\_color);

//генерация больших чисел с заменой последних битов на цвета

BigInteger[] musor\_p\_color\_\_ArrBI = new BigInteger[this.coloringGraph.nodesArrInteger.length];

int size\_bit\_write = BigInteger.valueOf(this.coloringGraph.count\_color + 1).bitLength(); //LINK55412542542412541

this.size\_bit\_write = size\_bit\_write;

for (int i = 0; i < musor\_p\_color\_\_ArrBI.length; i++) {

musor\_p\_color\_\_ArrBI[i] = MyBigInteger.generae\_diapason(BigInteger.ZERO, global\_max\_\_BI);

musor\_p\_color\_\_ArrBI[i].shiftLeft(size\_bit\_write);

BigInteger colorBI = BigInteger.valueOf(this.coloringGraph.nodesArrInteger[i]);

BigInteger color\_no\_equal\_0\_\_BI = colorBI.add(BigInteger.ONE); //LINK55412542542412541

musor\_p\_color\_\_ArrBI[i].add(color\_no\_equal\_0\_\_BI);

}

//формирование для каждой вершины RSA данных

this.cArrBI = new BigInteger[musor\_p\_color\_\_ArrBI.length];

BigInteger[][] resArrArrBI = new BigInteger[musor\_p\_color\_\_ArrBI.length][];

for (int i = 0; i < musor\_p\_color\_\_ArrBI.length; i++) {

resArrArrBI[i] = encrypt\_node(musor\_p\_color\_\_ArrBI[i]);

this.cArrBI[i] = resArrArrBI[i][3];

resArrArrBI[i][3] = BigInteger.ZERO;

}

return resArrArrBI;

}

public BigInteger[] sending\_secret\_keys\_for\_vertices\_around\_given\_edge(

Pair<Integer, Integer> edgePariIntInt

) {

int index\_1 = edgePariIntInt.getKey();

int index\_2 = edgePariIntInt.getValue();

BigInteger[] resArrBI = new BigInteger[2];

System.out.println("real color 1: " + this.coloringGraph.nodesArrInteger[index\_1]);

System.out.println("real color 2: " + this.coloringGraph.nodesArrInteger[index\_2]);

resArrBI[0] = this.cArrBI[index\_1];

resArrBI[1] = this.cArrBI[index\_2];

this.cArrBI = null;

return resArrBI;

}

private BigInteger[] encrypt\_node(BigInteger rand\_p\_color\_\_BI) { //return n, d, z, c

BigInteger max\_diapason\_\_BI = global\_max\_\_BI;

BigInteger p1BI = PrimeNumber.generate(max\_diapason\_\_BI);

BigInteger p2BI;

boolean p2BI\_\_equals\_\_p2BI\_\_BOOL;

do {

p2BI = PrimeNumber.generate(max\_diapason\_\_BI);

p2BI\_\_equals\_\_p2BI\_\_BOOL = p1BI.compareTo(p2BI) == 0;

} while (p2BI\_\_equals\_\_p2BI\_\_BOOL);

BigInteger nBI = p1BI.multiply(p2BI);

BigInteger p1\_minus\_1\_\_BI = p1BI.subtract(BigInteger.ONE);

BigInteger p2\_minus\_1\_\_BI = p2BI.subtract(BigInteger.ONE);

BigInteger fi\_from\_n\_\_BI = p1\_minus\_1\_\_BI.multiply(p2\_minus\_1\_\_BI);

//GCD(dBI, fi\_from\_n\_\_BI) = 1

BigInteger dBI = PrimeNumber.coprime\_generate(fi\_from\_n\_\_BI, global\_max\_\_BI); //публичное: e - rand() нечетное и НОД(fi\_from\_n, e) == 1

//cd mod fi = 1

BigInteger cBI = dBI.modInverse(fi\_from\_n\_\_BI);

BigInteger zBI = rand\_p\_color\_\_BI.modPow(dBI, nBI);

BigInteger[] res\_\_Arr\_int = {nBI, dBI, zBI, cBI};

return res\_\_Arr\_int;

}

private int[] mixing\_array(int count\_color) {

int[] res\_\_Arr\_int = new int[count\_color];

for (int i = 0; i < count\_color; i++) {

res\_\_Arr\_int[i] = i;

}

for (int i = 0; i < count\_color; i++) {

int from\_key = abs(sharedRandom.nextInt()) % count\_color;

int to\_key = abs(sharedRandom.nextInt()) % count\_color;

int from\_value = res\_\_Arr\_int[from\_key];

int to\_value = res\_\_Arr\_int[to\_key];

res\_\_Arr\_int[to\_key] = from\_value;

res\_\_Arr\_int[from\_key] = to\_value;

}

return res\_\_Arr\_int;

}

}

**Класс ValidatingZeroKnowledgeProofGraphColoring:**

package lab\_6\_rgz;

import javafx.util.Pair;

import java.math.BigInteger;

import java.util.Random;

import static java.lang.Math.abs;

public class ValidatingZeroKnowledgeProofGraphColoring {

private Graph ishodniiGraph;

public ValidatingZeroKnowledgeProofGraphColoring(Graph ishodniiGraph) {

this.ishodniiGraph = ishodniiGraph;

}

public boolean validation(

int count\_validations,

ProvingZeroKnowledgeProofGraphColoring tProvZerKnowProofGraphColor

) throws Exception {

for (int i = 0; i < count\_validations; i++) {

BigInteger[][] ecrypt\_graph\_\_ArrArrBI = tProvZerKnowProofGraphColor.get\_crypt\_graph();

Pair<Integer, Integer> rand\_edge\_\_PairIntInt =

choice\_of\_random\_edge(this.ishodniiGraph);

System.out.println(rand\_edge\_\_PairIntInt.getKey() + " ---- " + rand\_edge\_\_PairIntInt.getValue()); //DEBUG\_DELETE

BigInteger[] secret\_keys\_\_ArrBI =

tProvZerKnowProofGraphColor.sending\_secret\_keys\_for\_vertices\_around\_given\_edge(

rand\_edge\_\_PairIntInt

);

BigInteger[] crypt\_keys\_of\_1\_node\_\_ArrBI = ecrypt\_graph\_\_ArrArrBI[rand\_edge\_\_PairIntInt.getKey()];

BigInteger[] crypt\_keys\_of\_2\_node\_\_ArrBI = ecrypt\_graph\_\_ArrArrBI[rand\_edge\_\_PairIntInt.getValue()];

BigInteger one\_n\_\_BI = crypt\_keys\_of\_1\_node\_\_ArrBI[0];

BigInteger one\_d\_\_BI = crypt\_keys\_of\_1\_node\_\_ArrBI[1];

BigInteger one\_z\_\_BI = crypt\_keys\_of\_1\_node\_\_ArrBI[2];

BigInteger one\_c\_\_BI = secret\_keys\_\_ArrBI[0];

BigInteger two\_n\_\_BI = crypt\_keys\_of\_2\_node\_\_ArrBI[0];

BigInteger two\_d\_\_BI = crypt\_keys\_of\_2\_node\_\_ArrBI[1];

BigInteger two\_z\_\_BI = crypt\_keys\_of\_2\_node\_\_ArrBI[2];

BigInteger two\_c\_\_BI = secret\_keys\_\_ArrBI[1];

BigInteger one\_color\_and\_musor\_\_BI = one\_z\_\_BI.modPow(one\_c\_\_BI, one\_n\_\_BI);

BigInteger two\_color\_and\_musor\_\_BI = two\_z\_\_BI.modPow(two\_c\_\_BI, two\_n\_\_BI);

int size\_bit\_write = tProvZerKnowProofGraphColor.getSize\_bit\_write();

BigInteger cut\_musor\_bits\_\_BI = BigInteger.valueOf(2).modPow(

BigInteger.valueOf(size\_bit\_write),

one\_n\_\_BI

); //FIXME по хорошему бы еще единичку приплюсовать

BigInteger one\_color\_\_BI = one\_color\_and\_musor\_\_BI.mod(cut\_musor\_bits\_\_BI);

BigInteger two\_color\_\_BI = two\_color\_and\_musor\_\_BI.mod(cut\_musor\_bits\_\_BI);

boolean the\_colors\_of\_the\_neighboring\_vertices\_are\_equal =

one\_color\_\_BI.equals(two\_color\_\_BI);

if (the\_colors\_of\_the\_neighboring\_vertices\_are\_equal)

return false;

}

return true;

}

private Pair<Integer, Integer> choice\_of\_random\_edge(Graph inGraph) {

Random tRandom = new Random();

int number = abs(tRandom.nextInt()) % inGraph.edgesArrayListPairIntInt.size();

return inGraph.edgesArrayListPairIntInt.get(number);

}

}

**Класс GraphTest:**

package lab\_6\_rgz;

import org.junit.Test;

import java.io.File;

import java.io.IOException;

import static org.junit.Assert.\*;

public class GraphTest {

@Test

public void sharedTEST() throws Exception {

Graph tGraph = Graph.generate\_random\_graph(5);

File output\_dot\_\_File = new File("tests\_files/lab\_6\_rgz/print\_to\_file\_test.dot");

File image\_output\_\_File = new File("tests\_files/lab\_6\_rgz/print\_to\_file\_test.png");

ValidatingZeroKnowledgeProofGraphColoring tValZeroKnowProofGraphColor =

new ValidatingZeroKnowledgeProofGraphColoring(tGraph);

tGraph.coloring\_graph();

Graph coloringGraph = tGraph;

coloringGraph.print\_to\_file\_dot(output\_dot\_\_File);

Graph.create\_image(output\_dot\_\_File.getAbsolutePath(), image\_output\_\_File.getAbsolutePath());

ProvingZeroKnowledgeProofGraphColoring tProvZeroKnowProofGraphColor =

new ProvingZeroKnowledgeProofGraphColoring(coloringGraph);

assertTrue(

tValZeroKnowProofGraphColor.validation(

2,

tProvZeroKnowProofGraphColor

)

);

}

}