Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра прикладной математики и кибернетики

Отчёт
по расчётно-графической работе
«Протокол доказательства с нулевым знанием для задачи “Раскраска графа”»

Выполнил:

студент группы ИП-111

Кузьменок Д. В.
ФИО студента

Работу проверила: Дьячкова И. С.
ФИО преподавателя

Новосибирск 2024 г.

Содержание

Задание

Задача о раскраске графа

В задаче о раскраске графа рассматривается граф с множеством вершин V и множеством ребер E (числа элементов в этих множествах будем обозначать через $|V|$ и $|E|$). Алиса знает правильную раскраску этого графа тремя красками (красной (R), синей (B) и желтой (Y)). Правильная раскраска — это такая, когда любые две вершины, соединенные одним ребром, окрашены разными цветами. Приведем пример (рис. 5.1).

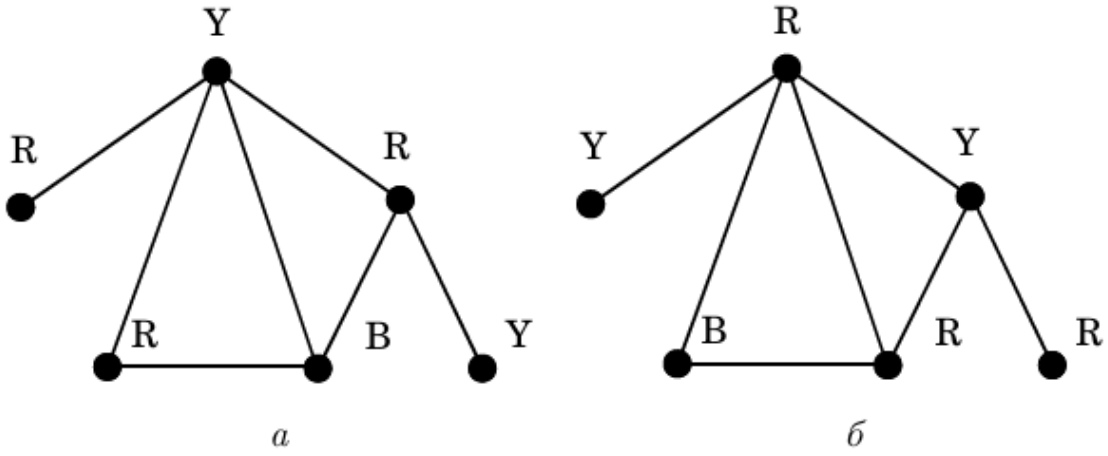


Рис. 5.1. Примеры раскрасок: *а* — правильная, *б* — неправильная

Для получения правильной раскраски графа тремя красками известны только экспоненциальные алгоритмы, т.е. такие, у которых время решения растет экспоненциально с ростом числа вершин и ребер в графе. Поэтому в случае больших $|V|$ и $|E|$ эта задача практически неразрешима.

Итак, Алиса знает (правильную) раскраску графа с большими $|V|$ и $|E|$. Она хочет доказать это Бобу, но так, чтобы он ничего не узнал об этой раскраске.

Протокол доказательства состоит из множества одинаковых этапов. Опишем сначала один этап.

Шаг 1. Алиса выбирает случайно перестановку Π из трех букв R, B, Y и перенумеровывает все вершины графа согласно этой перестановке. Очевидно, что раскраска останется верной. Например, если $\Pi = (\mathrm{Y}, \mathrm{R}, \mathrm{B})$, то граф слева на рис. 5.1 превращается в граф на рис. 5.2.

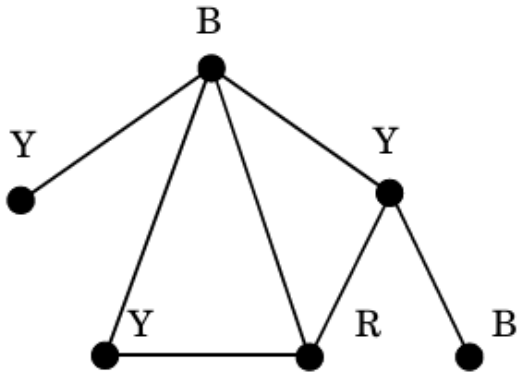


Рис. 5.2. Другой вариант раскраски

Шаг 2. Для каждой вершины v из множества V Алиса генерирует большое случайное число r и заменяет в нем два последних бита на 00, что соответствует красной вершине, 01 — синей, 10 — желтой.

Шаг 3. Для каждой вершины v Алиса формирует данные, используемые в RSA, а именно, P_v, Q_v, $N_v = P_v Q_v$, c_v и d_v.

Шаг 4. Алиса вычисляет

$$Z_v = r_v^{d_v} \bmod N_v$$

и посылает Бобу значения N_v, d_v и Z_v для каждой вершины графа.

Шаг 5. Боб выбирает случайно одно ребро из множества E и сообщает Алисе, какое именно ребро он выбрал . В ответ Алиса высылает числа c_{v_1} и c_{v_2}, соответствующие вершинам этого ребра. После этого Боб вычисляет

$$\hat{Z}_{v_1} = Z_{v_1}^{c_{v_1}} \bmod N_{v_1} = r_{v_1}, \quad \hat{Z}_{v_2} = Z_{v_2}^{c_{v_2}} \bmod N_{v_2} = r_{v_2}$$

и сравнивает два младших бита в полученных числах. При правильной раскраске два младших бита в числах $\hat{Z}_{v_1}$ и $\hat{Z}_{v_2}$ должны быть различны. Если значения совпали, значит, Алиса пыталась обмануть Боба, и на этом все заканчивается. Если не совпали, то весь описанный процесс повторяется $a|E|$ раз, где $a > 0$ — параметр.

Информацию о графах необходимо считывать из файла. В файле описание графа будет определяться следующим образом:

1) в первой строке файла содержатся два числа n <1001 и m $\leq n^2$, количество вершин графа и количество рёбер соответственно;

2) в последующих m строках содержится информация о рёбрах графа, каждое из которых описывается с помощью двух чисел (номера вершин, соединяемых этим ребром);

3) в зависимости от варианта указывается необходимая дополнительная информация: в первом варианте перечисляются цвета вершин графа (этот пункт можно вынести в отдельный файл).

Реализация программы

Для реализации задания я создал 4 класса, каждый из которых выполняет свою функцию, необходимую для корректной работы программы.

- Класс MainOperations: предоставляет необходимые функции, используемые для генерации значений в протоколе RSA (быстрое возведение числа в степень по модулю; генерация простого числа с проверкой, основанной на теореме Ферма; генерация случайного большого числа; обобщённый алгоритм Евклида; поиск взаимно простого числа).
- Класс RSAMethods: генерирует необходимые для RSA значение (большие простые значение P и Q; N = P * Q; φ = (P - 1) * (Q - 1); D и C – открытый и секретный ключи соответственно).
- Классы Edge и Graph: первый позволяет создать структуру графа, путем сохранения значений Source и Destination ребра (отправная точка и точка прибытия); второй же – реализует алгоритм раскраски графа, используя алгоритм Уэлша-Пауэлла (метод GetSortedVerticesByDegree), где вершины сначала сортируются по убыванию инцидентных ребер, а затем идет присваивание цветов в зависимости от цвета соседа (метод ColorGraph).
- Класс Program: реализует чтение файла, сохранение рёбер, сохраняет цвета вершин и присваивание каждой вершине необходимых параметров (r, N, D, C, Z). Затем в методе BobChecks идет проверка корректности раскраски графа, где сравниваются последние биты у каждой вершины, путем возведения числа Z в степень C по модулю N.

Результаты работы

Результат работы программы:

Входной файл выглядит следующим образом:

```
12 25
0 3
0 5
0 7
0 11
1 6
1 9
1 10
1 11
2 4
2 5
2 8
2 9
3 5
3 6
3 8
4 5
4 7
4 10
5 8
6 8
6 11
7 10
7 11
8 9
9 10
```

При правильной раскраске графа:

```
Вершины с R и их двоичные коды:
Вершина 1: R = 591503801, Цвет = BLACK, Двоичный код = 01
N = 60105269346280051, C = 34345867759701943, D = 7
Z = 5399776069367829
C * D % phi = 1
Вершина 2: R = 687602400, Цвет = YELLOW, Двоичный код = 00
N = 100121018681745233, C = 60072610774840841, D = 5
Z = 96981543641894471
C * D % phi = 1
Вершина 3: R = 380467110, Цвет = BLUE, Двоичный код = 10
N = 167541513013296361, C = 33508302435590465, D = 5
Z = 120662899062893649
C * D % phi = 1
Вершина 4: R = 223117710, Цвет = BLUE, Двоичный код = 10
N = 49922244164927311, C = 33281495591054267, D = 3
Z = 20366553645814793
C * D % phi = 1
Вершина 5: R = 944114201, Цвет = BLACK, Двоичный код = 01
N = 879204508742720527, C = 703363605492847853, D = 5
Z = 507991498845611402
C * D % phi = 1
Вершина 6: R = 432646500, Цвет = YELLOW, Двоичный код = 00
N = 349050802599328327, C = 232700534250130331, D = 3
Z = 235201344598397672
C * D % phi = 1
Вершина 7: R = 648823711, Цвет = RED, Двоичный код = 11
N = 360424767574356389, C = 163829439200007191, D = 11
Z = 163537997913691883
```

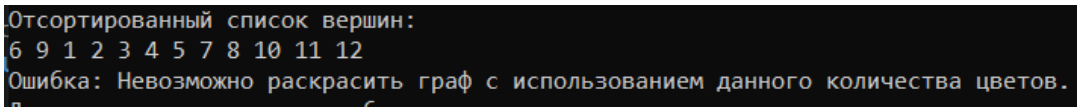
```
Вершина 8: R = 772442200, Цвет = YELLOW, Двоичный код = 00
N = 38672346043941989, C = 33147724684002223, D = 7
Z = 33913467001931916
C * D % phi = 1
Вершина 9: R = 127178401, Цвет = BLACK, Двоичный код = 01
N = 159004665595074107, C = 72274847629229891, D = 11
Z = 92859537581949832
C * D % phi = 1
Вершина 10: R = 938711211, Цвет = RED, Двоичный код = 11
N = 325331958988239041, C = 130132783113353933, D = 5
Z = 256124861453116221
C * D % phi = 1
Вершина 11: R = 797616010, Цвет = BLUE, Двоичный код = 10
N = 2434627951989405l9, C = 48692558838831869, D = 5
Z = 50782318576081052
C * D % phi = 1
Вершина 12: R = 39676710, Цвет = BLUE, Двоичный код = 10
N = 532913472325321187, C = 38757343105441571, D = 11
Z = 4814240947859437
C * D % phi = 1
```

```
Начинаю проверку раскраски графа Алисой.
Вершина 1: Z = 5399776069367829, C = 34345867759701943, N = 60105269346280051, результат: 591503801, последние два бита: 01
Вершина 4: Z = 20366553645814793, C = 33281495591054267, N = 49922244164927311, результат: 223117710, последние два бита: 10

Вершина 1: Z = 5399776069367829, C = 34345867759701943, N = 60105269346280051, результат: 591503801, последние два бита: 01
Вершина 6: Z = 235201344598397672, C = 232700534250130331, N = 349050802599328327, результат: 432646500, последние два бита: 00

Вершина 1: Z = 5399776069367829, C = 34345867759701943, N = 60105269346280051, результат: 591503801, последние два бита: 01
Вершина 8: Z = 33913467001931916, C = 33147724684002223, N = 38672346043941989, результат: 772442200, последние два бита: 00

Вершина 1: Z = 5399776069367829, C = 34345867759701943, N = 60105269346280051, результат: 591503801, последние два бита: 01
Вершина 12: Z = 4814240947859437, C = 38757343105441571, N = 53291347232532187, результат: 39676710, последние два бита: 10

Вершина 2: Z = 96981543641894471, C = 60072610774840841, N = 100121018681745233, результат: 687602400, последние два бита: 00
Вершина 7: Z = 163537997913691883, C = 163829439200007191, N = 360424767574356389, результат: 648823711, последние два бита: 11

Вершина 2: Z = 96981543641894471, C = 60072610774840841, N = 100121018681745233, результат: 687602400, последние два бита: 00
Вершина 10: Z = 256124861453116221, C = 130132783113353933, N = 325331958988239041, результат: 938711211, последние два бита: 11

Вершина 2: Z = 96981543641894471, C = 60072610774840841, N = 100121018681745233, результат: 687602400, последние два бита: 00
Вершина 11: Z = 50782318576081052, C = 48692558838831869, N = 243462795198940519, результат: 797616010, последние два бита: 10

Вершина 2: Z = 96981543641894471, C = 60072610774840841, N = 100121018681745233, результат: 687602400, последние два бита: 00
Вершина 12: Z = 4814240947859437, C = 38757343105441571, N = 53291347232532187, результат: 39676710, последние два бита: 10

Вершина 3: Z = 120662899062893649, C = 33508302435590465, N = 167541513013296361, результат: 380467110, последние два бита: 10
Вершина 5: Z = 507991498845611402, C = 703363605492847853, N = 879204508742720527, результат: 944114201, последние два бита: 01

Вершина 3: Z = 120662899062893649, C = 33508302435590465, N = 167541513013296361, результат: 380467110, последние два бита: 10
Вершина 6: Z = 235201344598397672, C = 232700534250130331, N = 349050802599328327, результат: 432646500, последние два бита: 00
```

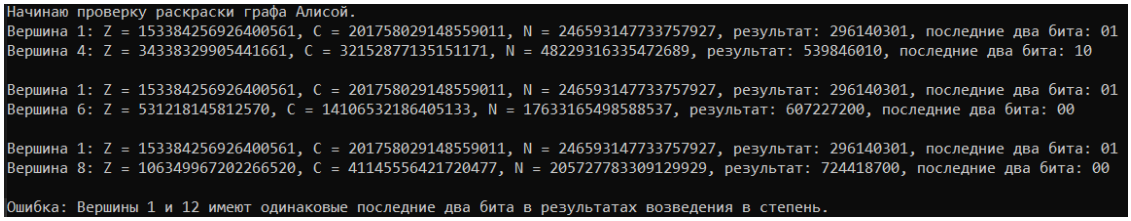
```
Вершина 3: Z = 120662899062893649, C = 33508302435590465, N = 167541513013296361, результат: 380467110, последние два бита: 10
Вершина 9: Z = 92859537581949832, C = 72274847629229891, N = 159004665595074107, результат: 127178401, последние два бита: 01

Вершина 3: Z = 120662899062893649, C = 33508302435590465, N = 167541513013296361, результат: 380467110, последние два бита: 10
Вершина 10: Z = 256124861453116221, C = 130132783113353933, N = 325331958988239041, результат: 938711211, последние два бита: 11

Вершина 4: Z = 20366553645814793, C = 33281495591054267, N = 49922244164927311, результат: 223117710, последние два бита: 10
Вершина 6: Z = 235201344598397672, C = 232700534250130331, N = 349050802599328327, результат: 432646500, последние два бита: 00

Вершина 4: Z = 20366553645814793, C = 33281495591054267, N = 49922244164927311, результат: 223117710, последние два бита: 10
Вершина 7: Z = 163537997913691883, C = 163829439200007191, N = 360424767574356389, результат: 648823711, последние два бита: 11

Вершина 4: Z = 20366553645814793, C = 33281495591054267, N = 49922244164927311, результат: 223117710, последние два бита: 10
Вершина 9: Z = 92859537581949832, C = 72274847629229891, N = 159004665595074107, результат: 127178401, последние два бита: 01

Вершина 5: Z = 507991498845611402, C = 703363605492847853, N = 879204508742720527, результат: 944114201, последние два бита: 01
Вершина 6: Z = 235201344598397672, C = 232700534250130331, N = 349050802599328327, результат: 432646500, последние два бита: 00

Вершина 5: Z = 507991498845611402, C = 703363605492847853, N = 879204508742720527, результат: 944114201, последние два бита: 01
Вершина 8: Z = 33913467001931916, C = 33147724684002223, N = 38672346043941989, результат: 772442200, последние два бита: 00

Вершина 5: Z = 507991498845611402, C = 703363605492847853, N = 879204508742720527, результат: 944114201, последние два бита: 01
Вершина 11: Z = 50782318576081052, C = 48692558838831869, N = 243462795198940519, результат: 797616010, последние два бита: 10

Вершина 6: Z = 235201344598397672, C = 232700534250130331, N = 349050802599328327, результат: 432646500, последние два бита: 00
Вершина 9: Z = 92859537581949832, C = 72274847629229891, N = 159004665595074107, результат: 127178401, последние два бита: 01

Вершина 7: Z = 163537997913691883, C = 163829439200007191, N = 360424767574356389, результат: 648823711, последние два бита: 11
Вершина 9: Z = 92859537581949832, C = 72274847629229891, N = 159004665595074107, результат: 127178401, последние два бита: 01
```

```
Вершина 7: Z = 163537997913691883, C = 163829439200007191, N = 360424767574356389, результат: 648823711, последние два бита: 11
Вершина 9: Z = 92859537581949832, C = 72274847629229891, N = 159004665595074107, результат: 127178401, последние два бита: 01

Вершина 7: Z = 163537997913691883, C = 163829439200007191, N = 360424767574356389, результат: 648823711, последние два бита: 11
Вершина 12: Z = 4814240947859437, C = 38757343105441571, N = 53291347232532187, результат: 39676710, последние два бита: 10

Вершина 8: Z = 33913467001931916, C = 33147724684002223, N = 38672346043941989, результат: 772442200, последние два бита: 00
Вершина 11: Z = 50782318576081052, C = 48692558838831869, N = 243462795198940519, результат: 797616010, последние два бита: 10

Вершина 8: Z = 33913467001931916, C = 33147724684002223, N = 38672346043941989, результат: 772442200, последние два бита: 00
Вершина 12: Z = 4814240947859437, C = 38757343105441571, N = 53291347232532187, результат: 39676710, последние два бита: 10

Вершина 9: Z = 92859537581949832, C = 72274847629229891, N = 159004665595074107, результат: 127178401, последние два бита: 01
Вершина 10: Z = 256124861453116221, C = 130132783113353933, N = 325331958988239041, результат: 938711211, последние два бита: 11

Вершина 10: Z = 256124861453116221, C = 130132783113353933, N = 325331958988239041, результат: 938711211, последние два бита: 11
Вершина 11: Z = 50782318576081052, C = 48692558838831869, N = 243462795198940519, результат: 797616010, последние два бита: 10

Граф правильно раскрашен.
```

Также алгоритм может самостоятельно определить, достаточно ли будет цветов для раскраски текущего графа. В случае, если цветов мало, будет выведено следующее сообщение:



Если же мы специально захотим испортить раскраску графа, сделав её неправильной, то Боб при проверке выведет информацию, какие две вершины были неправильно раскрашены:



Листинг программы

MainOperations.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;

namespace RGR
{
    class MainOperations
    {
        public static ulong FastPow(ulong a, ulong x, ulong p)
        {
            ulong result = 1;
            List<ulong> temp = new List<ulong> { a % p };
            ulong t = (ulong)Math.Floor(Math.Log(x, 2));
            List<ulong> binaryExponent = ToBinary(x);
            for (int i = 1; i <= (int)t; i++)
            {
                temp.Add((temp[i - 1] * temp[i - 1]) % p);
            }
            for (int i = 0; i <= (int)t; i++)
            {
                if (binaryExponent[i] != 0)
                {
                    result = (result * temp[i]) % p;
                }
            }
            return result;
        }

        public static BigInteger FastPow1(BigInteger a, BigInteger x,
BigInteger p)
        {
            BigInteger result = 1;
            List<BigInteger> temp = new List<BigInteger> { a % p };
            long t = (long)Math.Floor(Math.Log((double)x, 2));

            List<BigInteger> binaryExponent = ToBinary1(x);

            for (int i = 1; i <= t; i++)
            {
                temp.Add((temp[i - 1] * temp[i - 1]) % p);
            }

            for (int i = 0; i <= t; i++)
            {
                if (binaryExponent[i] != 0)
                {
                    result = (result * temp[i]) % p;
                }
            }
            return result;
        }

        private static List<ulong> ToBinary(ulong x)
```

```
        {
            List<ulong> result = new List<ulong>();
            while (x != 0)
            {
                result.Add(x & 1);
                x = x >> 1;
            }
            return result;
        }

        public static List<BigInteger> ToBinary1(BigInteger x)
        {
            List<BigInteger> result = new List<BigInteger>();
            while (x != 0)
            {
                result.Add(x & 1);
                x = x >> 1;
            }
            return result;
        }

        public static ulong GenerateModule(ulong left, ulong right, Random
rnd)
        {
            //Random rnd = new Random();
            ulong p = 0;
            ulong range = right - left + 1;
            while (true)
            {
                //p = rnd.Next((int)left, (int)right);
                p = (ulong)(rnd.NextDouble() * range) + left;
                if (IsPrime(p))
                {
                    return p;
                }
            }
        }

        //Ферма
        public static bool IsPrime(ulong number)
        {
            Random rnd = new Random();
            if (number <= 1) return false;
            else if (number == 2) return true;
            for (ulong i = 0; i < 100; i++)
            {
                ulong a = (ulong)rnd.Next(2, (int)number - 1);
                if (FastPow(a, number - 1, number) != 1 || Gcd(number, a) !=
1) return false;
            }
            return true;
        }

        public static ulong Gcd(ulong a, ulong b)
        {
            while (b != 0)
            {
                ulong r = a % b;
                a = b;
                b = r;
```

```
            }
            return a;
        }

        public static BigInteger Gcd1(BigInteger a, BigInteger b)
        {
            while (b != 0)
            {
                BigInteger r = a % b;
                a = b;
                b = r;
            }
            return a;
        }

        public static ulong GenerateExponent(ulong left, ulong right, Random
rnd)
        {
            return (ulong)rnd.Next((int)left, (int)right);
        }

        public static (BigInteger, BigInteger, BigInteger)
EvklidSolve1(BigInteger a, BigInteger b)
        {
            (BigInteger, BigInteger, BigInteger) U = (a, 1, 0);
            (BigInteger, BigInteger, BigInteger) V = (b, 0, 1);

            while (V.Item1 != 0)
            {
                BigInteger q = U.Item1 / V.Item1;
                (BigInteger, BigInteger, BigInteger) T = (U.Item1 % V.Item1,
U.Item2 - q * V.Item2, U.Item3 - q * V.Item3);
                U = V;
                V = T;
            }

            return U;
        }

        public static ulong GenerateCoprime(BigInteger p)
        {
            ulong result = 0;
            for (ulong i = 2; i < p; i++)
            {
                if (MainOperations.Gcd1(p, i) == 1)
                {
                    result = i;
                    break;
                }
            }
            return result;
        }
    }
}
```

Graph.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RGR
{
    //класс ребра
    public class Edge
    {
        public int Source { get; set; }
        public int Destination { get; set; }

        public Edge(int source, int destination)
        {
            Source = source;
            Destination = destination;
        }
    }

    public class Graph
    {
        public static string[] colors = { "YELLOW", "RED", "BLUE", "BLACK" };
        public static Dictionary<int, string> ColorGraph(List<Edge> edges,
int numVertices)
        {
            // Перемешиваем цвета случайным образом
            Random rnd = new Random();
            colors = colors.OrderBy(x => rnd.Next()).ToArray();

            // Словарь для хранения цвета каждой вершины
            Dictionary<int, string> vertexColors = new Dictionary<int,
string>();

            // Инициализация всех вершин как не раскрашенных
            for (int i = 0; i < numVertices; i++)
            {
                vertexColors[i] = null;
            }

            // Сортируем вершины по степени
            List<int> sortedVertices = GetSortedVerticesByDegree(edges,
numVertices);

            Console.WriteLine("Отсортированный список вершин:");
            foreach (var vertex in sortedVertices)
            {
                Console.Write($"{vertex + 1} ");
            }
            Console.WriteLine();

            // Раскрашиваем вершины по порядку, начиная с вершины с
максимальной степенью
            foreach (int vertex in sortedVertices)
            {
                // Список цветов, которые уже использованы смежными вершинами
                List<string> usedColors = new List<string>();
```

```
                // Проверяем цвета соседей
                foreach (var edge in edges.Where(e => e.Source == vertex ||
e.Destination == vertex))
                {
                    int neighbor = (edge.Source == vertex) ? edge.Destination
: edge.Source;
                    if (vertexColors.ContainsKey(neighbor) &&
vertexColors[neighbor] != null)
                    {
                        usedColors.Add(vertexColors[neighbor]);
                    }
                }

                // Проверяем, есть ли доступный цвет
                if (!colors.Except(usedColors).Any())
                {
                    // Если нет доступного цвета, то не удается раскрасить
граф
                    throw new Exception("Невозможно раскрасить граф с
использованием данного количества цветов.");
                }

                // Выбираем первый доступный цвет
                foreach (var color in colors)
                {
                    if (!usedColors.Contains(color))
                    {
                        vertexColors[vertex] = color;
                        break;
                    }
                }
            }

            return vertexColors;
        }
        // Метод для получения отсортированного списка вершин по степени
        private static List<int> GetSortedVerticesByDegree(List<Edge> edges,
int numVertices)
        {
            // Создаем словарь для хранения степеней вершин
            Dictionary<int, int> degrees = new Dictionary<int, int>();
            for (int i = 0; i < numVertices; i++)
            {
                degrees[i] = 0;
            }

            // Подсчитываем степени вершин
            foreach (var edge in edges)
            {
                degrees[edge.Source]++;
                degrees[edge.Destination]++;
            }

            // Сортируем вершины по степени
            return degrees.OrderByDescending(x => x.Value).Select(x =>
x.Key).ToList();
        }
    }
}
```

RSAMethods.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;

namespace RGR
{
    class RSAMethods
    {
        internal BigInteger N;
        internal BigInteger C;
        internal ulong D;
        internal BigInteger phi;
        private static Random rnd = new Random();
        public RSAMethods()
        {

            ulong P = MainOperations.GenerateModule(1000000, 1000000000,
rnd);
            ulong Q = MainOperations.GenerateModule(1000000, 1000000000,
rnd);

            N = P * Q;
            phi = (P - 1) * (Q - 1);
            D = MainOperations.GenerateCoprime(phi);

            C = MainOperations.EvklidSolve1(D, phi).Item2;

            if(C < 0)
            {
                C += phi;
            }
        }
    }
}
```

Program.cs:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Numerics;

namespace RGR
{
    public class Program
    {
        static void Main(string[] args)
        {
            List<Edge> edges = new List<Edge>();
            int numVertices = 0;
            List<Dictionary<string, BigInteger>> informationVertexes = new
List<Dictionary<string, BigInteger>>();
            List<BigInteger> zVertex = new List<BigInteger>();

            try
            {
                string filePath = "graph.txt";
                string[] lines = File.ReadAllLines(filePath);

                string[] firstLine = lines[0].Split(' ');
                numVertices = int.Parse(firstLine[0]);
                int numEdges = int.Parse(firstLine[1]);

                for (int i = 1; i <= numEdges; i++)
                {
                    string[] edgeData = lines[i].Split(' ');
                    int source = int.Parse(edgeData[0]);
                    int destination = int.Parse(edgeData[1]);
                    edges.Add(new Edge(source, destination));
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Ошибка при чтении файла: {ex.Message}");
                return;
            }

            try
            {
                Dictionary<int, string> vertexColors =
Graph.ColorGraph(edges, numVertices);
                //vertexColors[11] = "BLUE";

                Console.WriteLine("Раскраска графа:");
                for (int i = 0; i < numVertices; i++)
                {
                    Console.WriteLine($"Вершина {i + 1}: {vertexColors[i]}");
                }

                List<ulong> vertexesWithR = new List<ulong>(numVertices);
                Random random = new Random();
                for (int i = 0; i < numVertices; i++)
                {
```

```
                    ulong r = MainOperations.GenerateExponent(1000000,
1000000000, random);
                    Console.WriteLine($"r = {r}");
                    vertexesWithR.Add(r);
                }

                Dictionary<string, string> colorToBinary = new
Dictionary<string, string>();
                for (int i = 0; i < Graph.colors.Length; i++)
                {
                    string binaryCode = Convert.ToString(i, 2).PadLeft(2,
'0');
                    colorToBinary.Add(Graph.colors[i], binaryCode);
                }

                for(int i = 0; i < numVertices; i++)
                {
                    string binaryColor = colorToBinary[vertexColors[i]];
                    string tempNumber = vertexesWithR[i].ToString();
                    tempNumber = tempNumber.Substring(0, tempNumber.Length -
2) + binaryColor;

                    vertexesWithR[i] = ulong.Parse(tempNumber);
                }

                for(int i = 0; i < numVertices; i++)
                {
                    RSAMethods rsa = new RSAMethods();

                    Dictionary<string, BigInteger> vertexInfo = new
Dictionary<string, BigInteger>()
                    {
                        {"N", rsa.N },
                        {"D", rsa.D },
                        {"C", rsa.C },
                        {"phi", rsa.phi }
                    };

                    informationVertexes.Add(vertexInfo);
                    Console.WriteLine($"R = {vertexesWithR[i]}, D =
{informationVertexes[i]["D"]}, N = {informationVertexes[i]["N"]}, C =
{informationVertexes[i]["C"]}, phi = {informationVertexes[i]["phi"]}");
                    zVertex.Add(MainOperations.FastPow1(vertexesWithR[i],
informationVertexes[i]["D"], informationVertexes[i]["N"]));
                }

                Console.WriteLine("\nВершины с R и их двоичные коды:");
                for (int i = 0; i < numVertices; i++)
                {
                    Console.WriteLine($"Вершина {i + 1}: R =
{vertexesWithR[i]}, Цвет = {vertexColors[i]}, Двоичный код =
{colorToBinary[vertexColors[i]]}");
                    Console.WriteLine($"N = {informationVertexes[i]["N"]}, C
= {informationVertexes[i]["C"]}, D = {informationVertexes[i]["D"]}");
                    Console.WriteLine($"Z = {zVertex[i]}");
                    Console.WriteLine($"C * D % phi =
{MainOperations.FastPow1(informationVertexes[i]["C"] *
informationVertexes[i]["D"], 1, informationVertexes[i]["phi"])}");
                }
```

```
                BobChecks(edges, informationVertexes, zVertex);
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Ошибка: {ex.Message}");
            }
        }

        static void BobChecks(List<Edge> edges, List<Dictionary<string,
BigInteger>> informationVertexes, List<BigInteger> zVertex)
        {
            string lastTwoBits_r1 = "", lastTwoBits_r2 = "";
            Console.WriteLine("Начинаю проверку раскраски графа Алисой.");
            for (int i = 0; i < edges.Count; i++)
            {
                int source = edges[i].Source;
                int destination = edges[i].Destination;

                BigInteger nSource = informationVertexes[source]["N"];
                BigInteger cSource = informationVertexes[source]["C"];

                BigInteger nDestination =
informationVertexes[destination]["N"];
                BigInteger cDestination =
informationVertexes[destination]["C"];

                BigInteger _r1 = MainOperations.FastPow1(zVertex[source],
cSource, nSource);
                BigInteger _r2 =
MainOperations.FastPow1(zVertex[destination], cDestination, nDestination);

                string _r1String = _r1.ToString();
                string _r2String = _r2.ToString();

                // Проверка на равенство последних двух битов
                if (_r1String.Length >= 2 && _r2String.Length >= 2)
                {
                    lastTwoBits_r1 = _r1String.Substring(_r1String.Length -
2);
                    lastTwoBits_r2 = _r2String.Substring(_r2String.Length -
2);

                    if (lastTwoBits_r1 == lastTwoBits_r2)
                    {
                        Console.WriteLine($"Ошибка: Вершины {source + 1} и
{destination + 1} имеют одинаковые последние два бита в результатах
возведения в степень.");
                        return;
                    }
                }
                Console.WriteLine($"Вершина {source + 1}: Z =
{zVertex[source]}, C = {cSource}, N = {nSource}, результат: {_r1}, последние
два бита: {lastTwoBits_r1}");
                Console.WriteLine($"Вершина {destination + 1}: Z =
{zVertex[destination]}, C = {cDestination}, N = {nDestination}, результат:
{_r2}, последние два бита: {lastTwoBits_r2}");
                Console.WriteLine();
            }

            Console.WriteLine("Граф правильно раскрашен.");
```

```
            }
        }
    }
```