

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

Кафедра прикладной математики и кибернетики

Расчётно-графическое задание по предмету «Защита информации»
«Доказательство с нулевым знанием»
Вариант 1

Выполнил:
студент группы ИП-713
Михеев Никита Алексеевич

Работу проверила:
ассистент кафедры ПМиК
Петухова Я. В.

Новосибирск 2020 г.

Оглавление

1. Постановка задачи	3
2. Теоретические сведения.....	3
3. Реализация	4
4. Скриншоты работы программы	5
5. Список литературы	7
6. Листинг	8

1. Постановка задачи

Необходимо написать программу, реализующую протокол доказательства с нулевым знанием для задачи «Раскраска графа». Задача является NP-полной и не имеет быстрых методов для решения, поэтому для тестирования необходимо будет генерировать правильные решения при помощи дополнительно разработанных программ. Необходимо информацию о графах считывать из файла. В файле описание графа будет определяться следующим образом:

- 1) в первой строке файла содержатся два числа n и m , количество вершин графа и количество ребер соответственно;
- 2) в последующих m строках содержится информация о ребрах графа, каждое из которых описывается с помощью двух чисел (номера вершин, соединяемых этим ребром);
- 3) указывается необходимая дополнительная: перечисляются цвета вершин графа.

2. Теоретические сведения

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), зеленой (G) и синей (B)). Правильная раскраска — это такая, когда любые две вершины, соединенные одним ребром, окрашены разными цветами.

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 случайную перестановку Π из трех букв (R, G, B) и перекрашивает все вершины графа согласно этой перестановке. Очевидно, что раскраска графа останется верной.

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

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

Шаг 4. Алиса вычисляет для каждой вершины v :

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

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

Шаг 5. Боб выбирает случайно одно ребро и сообщает Алисе, какой именно ребро он выбрал. В ответ Алиса высылает числа c_{v1} и c_{v2} , соответствующие вершинам этого ребра. После этого Боб вычисляет:

$$\hat{Z}_{v1} = Z_{v1}^{c_{v1}} \bmod N_{v1} == r_{v1}; \hat{Z}_{v2} = Z_{v2}^{c_{v2}} \bmod N_{v2} == r_{v2}$$

и сравнивает два младших бита у этих чисел. Если раскраска правильная – два младших бита различаются. Если значения совпали – Алиса обманывает Боба и на этом все заканчивается. Если не совпали, то весь описанный процесс повторяется $a|E|$ раз, где $a > 0$ – параметр.

3. Реализация

Были составлены два тестовых, правильно раскрашенных графа. Далее была реализована программа, которая реализует поток доказательства с нулевым знанием для задачи «Раскраска графа» с использованием языка программирования Python версии 3.8 без сторонних библиотек. В программе были реализованы следующие функции:

is_prime – функция выполняет быструю проверку числа на простоту;

get_prime – функция генерирует простое число в заданном диапазоне;

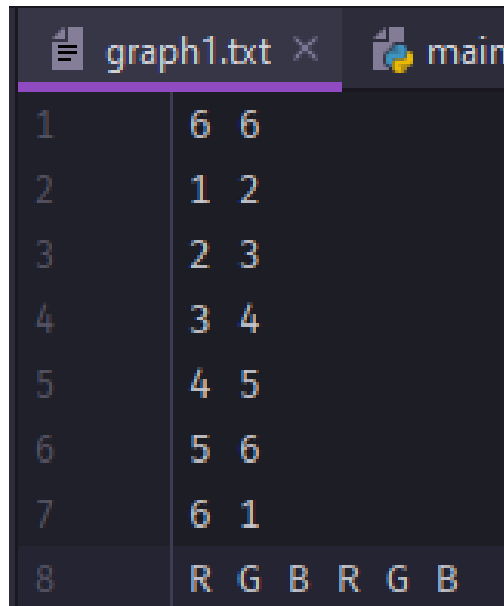
get_coprime – функция находит взаимно простое число данному;

extended_Euclidean_algorithm – реализация расширенного алгоритма Евклида;

load_graph – загружает заранее созданный граф из файла и информацию о его раскраске. Полученный список ребер добавляется в словарь значений с двумя ключами «from» - из какой вершины, «to» - в какую вершину, а информация о цветах заносится в список colors;

main – основная функция, в которой происходят все основные шаги программы: перемешивание цветов Алисой, перекраска графа, генерация больших чисел r для каждой вершины и замена последних двух битов в них, генерация данных, как в RSA, и проверка Бобом данных, что дала ему Алиса.

4. Скриншоты работы программы



1	6 6
2	1 2
3	2 3
4	3 4
5	4 5
6	5 6
7	6 1
8	R G B R G B

Рис.1 – пример файла с готовым графом.

```
"C:\Users\Lolimpo\Google Drive\SibSUTIS\Labs\4 course\Data Protection\Labs\v
Граф содержит 6 вершин и 6 ребер:
1 2
2 3
3 4
4 5
5 6
6 1
Раскраска: ['R', 'G', 'B', 'R', 'G', 'B']
Перекрашенный граф Алисой: ['G', 'B', 'R', 'G', 'B', 'R']
r = [1637330473, 1721763602, 4260643788, 773970917, 2013357314, 1145560964]
Для ребра 1 два младших бита различны.
Для ребра 2 два младших бита различны.
Для ребра 3 два младших бита различны.
Для ребра 4 два младших бита различны.
Для ребра 5 два младших бита различны.
Для ребра 6 два младших бита различны.

Process finished with exit code 0
```

Рис.2 – результат работы программы: вывод заданного графа, его раскраска, результат перемешивания цветов Алисой, числа r для всех вершин и проверка каждой вершины Бобом.

```
Граф содержит 7 вершин и 7 ребер:
1 2
2 3
3 4
4 5
5 6
6 7
7 1
Раскраска: ['R', 'G', 'B', 'R', 'G', 'B', 'R']
Перекрашенный граф Алисой: ['R', 'B', 'G', 'R', 'B', 'G', 'R']
r = [1036431948, 3310880742, 112944913, 1742093580, 2779382466, 3124697709, 2888625228]
Для ребра 1 два младших бита различны.
Для ребра 2 два младших бита различны.
Для ребра 3 два младших бита различны.
Для ребра 4 два младших бита различны.
Для ребра 5 два младших бита различны.
Для ребра 6 два младших бита различны.
Алиса обманула Боба! Два последних бита совпадают у ребра 7 с вершинами:
10101100001011001110110001001100 | 111101110001101011001001001100

Process finished with exit code 0
```

Рис.3 – попытка Алисы обмануть Боба с раскраской.

5. Список литературы

1. Рябко Б. Я., Фионов А. Н. Криптографические методы защиты информации: Учебное пособие для вузов. – 2-е издание, стереотип. – М.: Горячая линия–Телеком, 2012. – 229 с.
2. Визинг В.Г., Раскраска вершин графа в предписанные цвета // Методы дискретного анализа в теории кодов и схем: сборник научных трудов. Т. 29. Институт математики СО АН СССР: Новосибирск, 1976. С. 3–10.
3. Любанович Б., Простой Python. Современный стиль программирования. – СПб.: Питер, 2017 – 480 с.: ил. – (Серия «Бестселлеры O'Reilly»).

6. Листинг

Основной модуль программы main.py:

```
from Lab3.lab3 import *

def read_graph(filename: str):
    """
    Функция получает на вход название файла, в котором лежит граф и возвращает
    считанный из него
    список ребер в dict, список цветов в list, и количество вершин int.
    """

    vertex_array = {'from': [], 'to': []}
    with open(filename, 'r') as f:
        vertex_num, edge_num = [int(x) for x in next(f).split()]
        for i, line in enumerate(f):
            if i == edge_num:
                colors = [x for x in line.split()]
                break
            fr, to = [int(x) for x in line.split()]
            vertex_array['from'].append(fr)
            vertex_array['to'].append(to)
    return vertex_array, colors, vertex_num

def main() -> int:
    """
    Основная функция программы.
    """

    graph, colors, vertex_num = read_graph('graph_wrong.txt')
    print(f'Граф содержит {vertex_num} вершин и {len(graph["from"])}
    ребер:')
    for i in range(len(graph["from"])):
        print(f'{graph["from"][i]} {graph["to"][i]}')
    print(f'Раскраска: {colors}')

    # Часть Алисы
    color_name = ['R', 'G', 'B']
    color_name_shuffle = color_name.copy()
    while color_name_shuffle == color_name:
        random.shuffle(color_name_shuffle)
    colors_shuffle = ['' for _ in range(len(colors))]
    for i in range(vertex_num):
        if colors[i] == 'R':
            colors_shuffle[i] = color_name_shuffle[0]
        elif colors[i] == 'G':
            colors_shuffle[i] = color_name_shuffle[1]
        elif colors[i] == 'B':
            colors_shuffle[i] = color_name_shuffle[2]
    print(f'Перекрашенный граф Алисой: {colors_shuffle}')

    # Генерация больших чисел r Алисой
    r = list()
    for i in colors_shuffle:
        if i == 'R':
            r.append(random.getrandbits(32) >> 2 << 2)
```



```

        elif i == 'G':
            r.append((random.getrandbits(32) >> 2 << 2) | 1)
        elif i == 'B':
            r.append((random.getrandbits(32) >> 2 << 2) | 2)
    print(f'r = {r}')

    # Генерация данных, как в RSA
    p = [get_prime(0, 10 ** 9) for _ in range(vertex_num)]
    q = [get_prime(0, 10 ** 9) for _ in range(vertex_num)]
    n = [p[i] * q[i] for i in range(vertex_num)]
    phi = [(p[i] - 1) * (q[i] - 1) for i in range(vertex_num)]
    d = [get_coprime(phi[i]) for i in range(vertex_num)]
    c = [extended_euclidean_algorithm(d[i], phi[i])[1] for i in
range(vertex_num)]
    for i in range(vertex_num):
        while c[i] < 0:
            c[i] += phi[i]

    Z = [pow_mod(r[i], d[i], n[i]) for i in range(vertex_num)]

    # Проверка ребер Бобом
    for i in range(len(graph['from'])):
        _Z1 = pow(Z[graph['from'][i] - 1], c[graph['from'][i] - 1],
n[graph['from'][i] - 1])
        _Z2 = pow(Z[graph['to'][i] - 1], c[graph['to'][i] - 1],
n[graph['to'][i] - 1])
        print(f'Для ребра {i + 1} два младших бита различны.'
            if bin(_Z1)[-2:] != bin(_Z2)[-2:]
            else f'Алиса обманула Боба! Два последних бита совпадают у
ребра {i + 1} с вершинами:\n{_Z1:b} | {_Z2:b}')
    return 0

# Точка входа в программу.
if __name__ == '__main__':
    exit(main())

```

Листинг вспомогательных функций:

```

def is_prime(n):
    for x in range(1, 5):
        if pow(random.randrange(n - 1) + 1, (n - 1), n) != 1:
            return False
    return True

def get_prime(left, right):
    while True:
        p = random.randint(left, right)
        if is_prime(p):
            return p

def pow_mod(base, exp, module):
    result = 1
    base %= module

    if not base:
        return 0

```

```

while exp > 0:
    if exp & 1 == 1:
        result = (result * base) % module
    exp >>= 1
    base = (base ** 2) % module

return result

def extended_euclidean_algorithm(a, b):
    u = [a, 1, 0]
    v = [b, 0, 1]
    while v[0] != 0:
        q = u[0] // v[0]
        t = [u[0] % v[0], u[1] - q * v[1], u[2] - q * v[2]]
        u, v = v, t
    return u

```