Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет
телекоммуникаций и информатики»

Расчетно графическая работа

по дисциплине «Защита информации»

Выполнил: студент 4 курса

ИВТ, гр. ИП-012

Хартов А.Е.

Проверил: ассистент кафедры ПМиК

Меркулов Игорь Александрович

Новосибирск, 2021 г.

Задание

Вариант: 1

Необходимо написать программу, реализующую протокол доказательства с нулевым знанием для задачи «Раскраска графа».

Вне зависимости от варианта задания, необходимо информацию о графах считывать из файла. В файле описание графа будет определяться следующим образом: 1) в первой строке файла содержатся два числа и , количество вершин графа и количество рѐбер соответственно; 2) в последующих m строках содержится информация о рѐбрах 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; во втором варианте указывается последовательность вершин, задающая гамильтонов цикл (этот пункт можно вынести в отдельный файл).

Листинг программы.

#rgr.py:

```
import random
import secrets

def gcd(a, b):
while b != 0:
r = a % b
a = b
b = r
return a

def is_prime(p):
if p <= 1:
return False
elif p == 2:
return True
a = random.randint(2, p - 1)
# print(p, "-", a)
if module(a, (p - 1), p) != 1 or gcd(p, a) > 1:
return False
return True

#возвращает рандомное простое число
def rand_prime():
while True:
tmp = secrets.randbits(32)
if is_prime(tmp):
return tmp
```

```
# Обобщённый Алгоритм Евклида, для нахождения
наибольшего общего делителя и двух неизвестных уравнения
def gcd_modified(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
# генерируем взаимно-простое число
def rand_prime_not_p(p):
while (True):
Cb = rand_prime()
if (Cb != p):
break
return Cb

#быстрое возведение в степень по модулю
def module(a, x, p):
y = 1
s = a

while x > 0:
if (x % 2 == 1):
y = (y * s) % p
s = (s * s) % p
x = x//2

return y
```

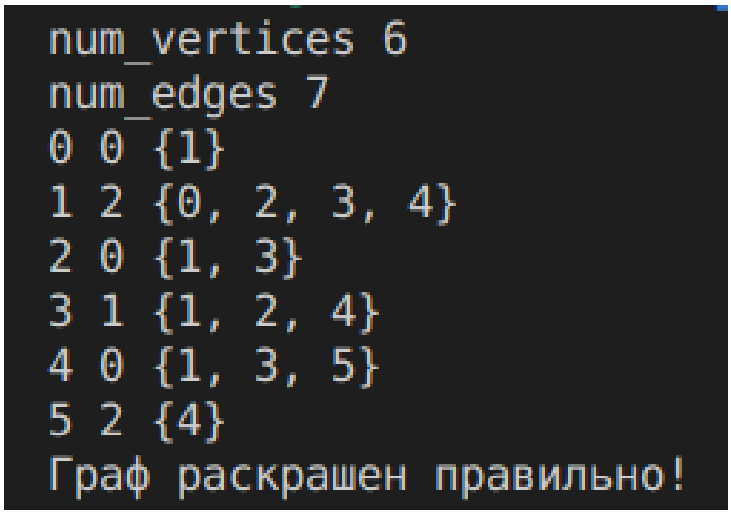
```
class Node:
def __init__(self) -> None:
self.color = -1
self.neighbors = set()

class Graph:
def __init__(self) -> None:
self.nodes = []

def read_graph(graph_name: str):
with open(graph_name, 'r') as file:
all_lines = file.readlines()
num_vertices = int(all_lines[0][0])
num_edges = int(all_lines[0][2])
print("num_vertices " + str(num_vertices))
print("num_edges " + str(num_edges))

graph = Graph()
edges = all_lines[2:int(num_edges)+2]
colors = all_lines[int(num_edges)+3:]
for ind_vert in range(0, num_vertices):
print(ind_vert, end=" ")
tmp_node = Node()
for ind_line in range(0, num_edges):
if ind_vert == int(edges[ind_line][0]):
tmp_node.neighbors.add(int(edges[ind_line][2]))
elif ind_vert == int(edges[ind_line][2]):
tmp_node.neighbors.add(int(edges[ind_line][0]))
tmp_node.color = int(colors[ind_vert][2])
graph.nodes.append(tmp_node)
```

```
print(tmp_node.color, end=" ")
print(tmp_node.neighbors)
return [graph, num_edges, edges]

def RSA_data():
P = rand_prime()
Q = rand_prime()
N = P * Q
Phi = (P - 1) * (Q - 1)
d = rand_prime_not_p(Phi)
c = gcd_modified(d, Phi)[1]
if c < 0:
c += Phi
return [P, Q, N, c, d]

def check_graph(graph, num_edges, edges):
aE = 10 * num_edges
for _ in range(0, aE):
colors = [0,1,2]
#выбираем случайную перестановку цветов
random.shuffle(colors)
#перенумеровываем все вершины согласно перестановке
for tmp_node in graph.nodes:
if tmp_node.color == 0:
tmp_node.color = colors[0]
elif tmp_node.color == 1:
tmp_node.color = colors[1]
elif tmp_node.color == 2:
tmp_node.color = colors[2]


r = list()
for node in graph.nodes:
```

```
tmp_rnd = secrets.randbits(32)
tmp_rnd = tmp_rnd >> 2 << 2 #обнуляем два последних бита
#заменяем два последних бита
if node.color == 1:
tmp_rnd = tmp_rnd | int("01", 2)
elif node.color == 2:
tmp_rnd = tmp_rnd | int("10", 2)
r.append(tmp_rnd)

#для каждой вершины формируем данные используемые в RSA
[P, Q, N, c, d]
rsa_list = list()
for node in graph.nodes:
rsa_list.append(RSA_data())

#Алиса вычисляет Z_v
Z = list()
for i in range(0, len(graph.nodes)):
Z.append(module(r[i], rsa_list[i][4], rsa_list[i][2]))

#Боб выбирает случайное ребро
ind_rand_edge = random.randint(0, num_edges - 1)

#Алиса высылает c_v1 и c_v2
rand_edge = (int(edges[ind_rand_edge][0]), int(edges[ind_rand_edge][2]))
v1 = rand_edge[0]
v2 = rand_edge[1]
c_v1 = rsa_list[v1][3]
c_v2 = rsa_list[v2][3]
N_v1 = rsa_list[v1][2]
N_v2 = rsa_list[v2][2]
```

```
#Боб вычисляет два числа и сравнивает два младших бита
Zv1_ = module(Z[v1], c_v1, N_v1)
Zv2_ = module(Z[v2], c_v2, N_v2)

if (bin(Zv1_ & 0b11) == bin(Zv2_ & 0b11)):
return("Граф раскрашен неправильно!")
return("Граф раскрашен правильно!")

data = read_graph("/home/aleksandr/Zash_inform/rgr/gr.txt")
graph = data[0]
num_edges = data[1]
edges = data[2]
print(check_graph(graph, num_edges, edges))

#gr.txt:
6 7

0 1
1 2
1 3
1 4
2 3
3 4
4 5

0 0
1 2
2 0
3 1
4 0
5 2
```

Результат работы.



Вывод

В ходе выполнения лабораторной работы мы изучили и применили на практике протокол доказательства с нулевым знанием для задачи «Раскраска графа»