Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра ПМиК

Расчетно-графическая работа по дисциплине

«Защита информации»

на тему «Доказательство с нулевым знанием»

«Вариант 2»

Выполнил: студент 4 курса

ИВТ, гр. ИП-013

Копытина Т.А

Проверил: Старший преподаватель кафедры ПМиК

Дьячкова Ирина Сергеевна

Новосибирск 2023г.

Оглавление

Постановка задачи

Необходимо написать программу, реализующую алгоритм доказательства с нулевым знанием «Гамильтонов цикл». Обе рассматриваемые задачи являются NP-полными и не имеют быстрых методов для решения, поэтому для тестирования необходимо будет генерировать правильные решения при помощи дополнительно разработанных программ. 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 n < 1001 и $m \leq n^2$, количество вершин графа и количество рѐбер соответственно;

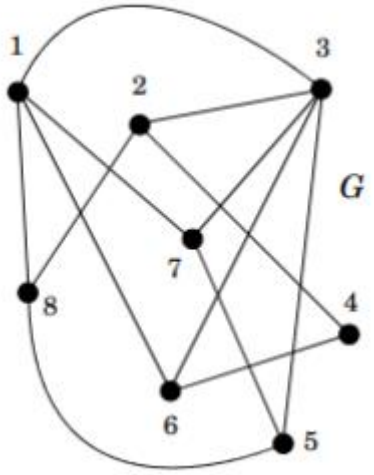
2) в последующих m строках содержится информация о рѐбрах графа, каждое из которых описывается с помощью двух чисел (номера вершин, соединяемых этим ребром);

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).

Теоретический материал

Рассматриваемая в данном разделе задача не просто предоставляет нам возможность описать еще одну схему построения протокола доказательства с нулевым знанием, но и имеет важное теоретическое значение. Блюм (Manuel Blum) показал, что, выражаясь неформально, любое математическое утверждение может быть представлено в виде графа, причем доказательство этого утверждения соответствует гамильтонову циклу в этом графе. Поэтому наличие протокола доказательства с нулевым знанием для гамильтонова цикла означает, что доказательство любого математического утверждения может быть представлено в форме доказательства с нулевым знанием.

Определение 5.1. Гамильтоновым циклом в графе называется непрерывный путь, проходящий через все вершины графа ровно по одному разу.



Путь, проходящий последовательно через вершины 8, 2, 4, 6, 3, 5, 7, 1, представляет собой гамильтонов цикл. Действительно, в этом криптографические протоколы пути содержатся все вершины графа, и каждая вершина посещается только один раз.

Ясно, что если в графе G с n вершинами гамильтонов цикл существует, то при некоторой нумерации вершин он пройдет точно через вершины с последовательными номерами 1, 2, 3, …, n. Поэтому путем перебора всех

возможных нумераций вершин мы обязательно найдем гамильтонов цикл. Но количество возможных нумераций равно n!, и поэтому уже при умеренно больших n, например, при n = 100, такой подход становится практически нереализуемым. Доказано, что задача нахождения гамильтонова цикла в графе является NP-полной. Мы уже говорили кратко о понятии NP-полноты. Неформально, NP-полнота рассматриваемой задачи означает, что для ее решения не существуют (точнее, неизвестны) алгоритмы существенно более быстрые, чем указанный метод перебора.

Нашей задачей будет построение криптографического протокола, с помощью которого Алиса будет доказывать Бобу, что она знает гамильтонов цикл в некотором графе G так, чтобы Боб не получил никаких знаний о самом этом цикле. Иными словами, Алиса будет предоставлять Бобу доказательство с нулевым знанием. Еще раз напомним читателю, что «нулевое знание» означает, что независимо от числа реализаций протокола доказательства Боб будет располагать точно такими же сведениями о гамильтоновом цикле, какие он мог бы получить, просто изучая представленный ему граф G.

Итак, допустим, что Алиса знает гамильтонов цикл в графе G. Теперь она может это доказывать Бобу и всем, кто имеет граф G, с помощью описываемого ниже протокола. Алиса может использовать это доказательство, например, для идентификации своей личности. Прежде чем мы перейдем к описанию протокола, договоримся о некоторых обозначениях.

Мы будем обозначать графы буквами G, H, F, понимая под этим одновременно соответствующие матрицы смежности. Элемент матрицы Hij = 1, если в графе H есть ребро, соединяющее вершины i и j; Hij = 0 в противном случае. Символом k будем обозначать конкатенацию (сцепление) двух чисел, точнее, двоичных слов, им соответствующих. Нам понадобится шифр с открытым ключом. Вообще говоря, это может быть любой шифр, но для определенности будем использовать шифр RSA. Будем считать, что Алиса сформировала систему RSA с открытыми параметрами N и d. Важно, что

зашифрованные в этой системе сообщения может расшифровать только Алиса и больше никто. Протокол доказательства состоит из следующих четырех шагов (пояснения будут даны ниже).

Шаг 1. Алиса строит граф H, являющийся копией исходного графа G, где у всех вершин новые, случайно выбранные номера. На языке теории графов говорят, что H изоморфен G. Иными словами, H получается путем некоторой перестановки вершин в графе G (с сохранением связей между вершинами). Алиса кодирует матрицу H, приписывая к первоначально содержащимся в ней нулям и единицам случайные числа rij по схеме Затем она шифрует элементы матрицы H˜, получая зашифрованную матрицу F. Матрицу F Алиса передает Бобу.

Шаг 2. Боб, получив зашифрованный граф F, задает Алисе один из двух вопросов.

1. Каков гамильтонов цикл для графа H?

2. Действительно ли граф H изоморфен G?

Шаг 3. Алиса отвечает на соответствующий вопрос Боба.

1. Она расшифровывает в F ребра, образующие гамильтонов цикл.

2. Она расшифровывает F полностью (фактически передает Бобу граф H˜) и предъявляет перестановки, с помощью которых граф H был получен из графа G.

Шаг 4. Получив ответ, Боб проверяет правильность расшифровки путем повторного шифрования и сравнения с F и убеждается либо в том, что показанные ребра действительно образуют гамильтонов цикл, либо в том, что предъявленные перестановки действительно переводят граф G в граф H. Весь протокол повторяется t раз. Обсудим вначале кратко несколько вопросов по построению протокола.

1. Зачем Алиса строит изоморфный граф? Если бы она этого не делала, то Боб, получив ответ на свой вопрос номер один, узнал бы гамильтонов цикл в графе G.

2. Зачем Алиса кодирует матрицу H? С этим приемом мы уже встречались при шифровании цветов вершин графа. Дело в том, что невозможно зашифровать непосредственно нули и единицы (с помощью шифра RSA они вообще не шифруются). Даже если заменить их на какие-то произвольные числа a и b, то мы получим всего два различных шифротекста, и Бобу не составит труда понять, какой из них какому числу соответствует. Т.е. структура графа не будет скрыта.

Здесь мы сталкиваемся с типичной ситуацией, когда требуется использовать так называемый рандомизированный шифр. И такой шифр строится путем добавления случайных чисел в матрицу H перед шифрованием. Закодированная матрица H˜ точно также задает граф (нечетность числа означает наличие ребра, четность — его отсутствие), но после шифрования H˜ структура графа полностью скрывается (мы используем известное свойство шифра RSA — он полностью скрывает четность числа).

3. Зачем Боб задает два вопроса? Если бы он задавал только вопрос номер один, который по смыслу протокола является основным, то Алиса, не зная в действительности гамильтонова цикла в графе G, могла бы предъявить Бобу совсем другой граф с таким же количеством вершин и искусственно заложенным в него гамильтоновым циклом. Поэтому Боб иногда просит Алису доказать изоморфизм графов H и G. Важно, что Алиса не знает заранее, какой из двух вопросов задаст Боб.

4. Почему Боб не может задать сразу двух вопросов? В этом случае он узнал бы гамильтонов цикл в G, так как ему был бы показан гамильтонов цикл в H и правило перехода от H к G.

5. Зачем Боб проверяет правильность расшифровки? Если бы он этого не делал, то Алиса на четвертом шаге могла бы предоставить ему «выгодную»

для себя информацию, а не ту, которую она посылала ему на втором шаге.

Ход работы

Реализованные методы в программе rgr1.py

def check_prime(p):

Функция проверяет число на простоту

def generate_prime(left, right):

Функция возвращает случайное простой число из передаваемой границы

def generate_coprime(p):

Функция возвращает случайное число взаимно-простое с передаваемым

def pow_module(a, x, p):

Функция возводит в степень по модулю

def gcd(a, b):

Функция - Алгоритм Евклида

def gcd_modified(a, b):

Функция - Модифицированный Алгоритм Евклида

def read_graph_from_file(file_path='Path.txt'):

Функция, создающая сам граф в виде смежной матрицы. Сначала создается матрица NxN заполненная нулями. Затем прочитывается файл с информацией о путях графа и эти пути заполняют матрицу: выставляются единицы(ребра) соединяющие наши вершины. Функция возвращает количество вершин N, ребер M, сам граф Graph и его копию, для проверки в последующих функциях.

def visualize_graph(graph, header, graph_type):

Функция создающая визуальное представление двух графов, исходного графа G и изоморфного графа H.

def all_path(graph)

Функция создает словарь из путей вершин. В функцию поступает граф в виде смежной матрицы. Проходя по двум циклам, алгоритм создает и записывает в словарь вершину (она же служит ключом словаря) и ее пути до других вершин.

Словарь вида:

1: [2, 3, 4, 5]

2: [1, 3, 5, 6]

3: [1, 2, 4, 6]

4: [1, 3, 5, 6]

5: [1, 2, 4, 6]

6: [2, 3, 4, 5]

def search_hamilton_cycle(graph, size, pt, path=[]):

Данная функция ищет Гамильтонов цикл по графу. На вход поступают:

1) Словарь связей вершин

2) Количество вершин, которое необходимо обойти

3) Выбор стартовой вершины

4) Пустой список, в который в будущем будет записан цикл прохождения по графу.

Программа начинает с заданной вершины, записывает ее в список, затем проверяет связи этой вершины с другими вершинами. Если связи есть, то начинает шагать вперед, записывая вершины в список, до тех пор, пока элементов в списке не станет равно количеству вершин N нашего графа. Запись осуществляется без повтора вершин. В случае, если цикл зашел в тупиковую вершину и ему некуда шагнуть, кроме как назад, он вернет список, в котором будет видно, что цикл не был пройден и вновь попытается найти цикл.

def rsa_matrix_encode(new_graph, x, p, N):

Функция кодирует элементы матрицы при помощи алгоритма RSA и возвращает результат.

def hamilton_cycle():

Основная часть программы. Алгоритм программы эмулирует общение двух людей. Сначала функция получает значения количества вершин N, ребер M, граф G и его копию граф H. Вторым шагом, по основному графу находится Гамильтонов цикл. Затем столбцы смежной матрицы переставляются и порядковые перестановки записываются в список, получая Изоморфный граф. Затем к каждому элементу матрицы дописывается число от 1 до N (случайное число на каждый элемент столбца матрицы одно и то же, но для каждого столбца уникально), чтобы каждый элемент можно было закодировать по алгоритму RSA. После кодирования, матрица готова к отправке Бобу.

Боб, получив матрицу, задает один из двух вопросов Алисе:

1) Каков Гамильтонов цикл для графа?

2) Действительно ли граф H изоморфен G

Выбрав первый вопрос, Алиса отправляет Бобу Гамильтонов цикл и матрицу H. Боб преобразует матрицу H, выбирая элемент по порядку из Гамильтонова цикла и проходя по алгоритму RSA. Полученную матрицу он сравнивает с той, что получил изначально, она же матрица F. Если они идентичны, то Боб проходит по матрице при помощи Гамильтонова цикла. Если это возможно, то алгоритм отработал верно.

Выбрав второй вопрос, Алиса отправляет Бобу матрицу H, которая еще не закодирована. Боб кодирует матрицу. Если эта матрица равна той, что он получил изначально, то Боб продолжает преобразовывать матрицу. Следующим шагом он отбрасывает от каждого элемента матрицы все разряды, оставляя разряд единиц в качестве элемента матрицы. Следом Алиса отправляет ему список переставленных порядков матрицы и Боб переставляет

столбцы матрицы в соответствии с этим списком. Тем самым Боб сможет убедиться, что матрицы идентичны.

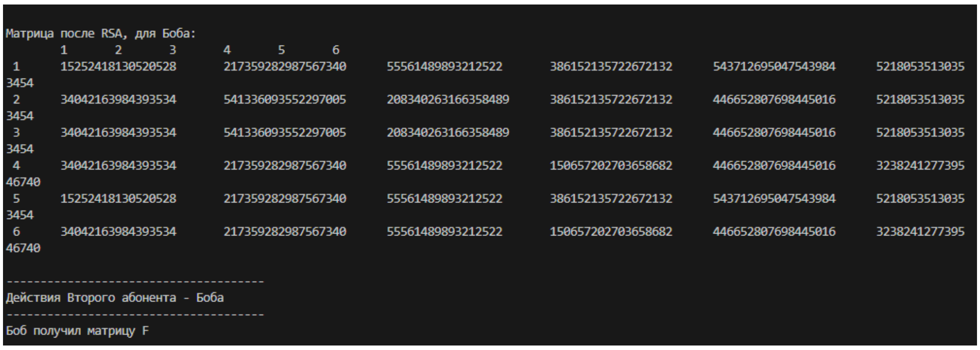
Результат работы программы

```
PS C:\ZI> & C:/Users/Татьяна/AppData/Local/Programs/Python/Python310/python.exe c:/ZI/rgr1.py
Стартовый граф:
       1      2      3      4      5      6
 1     0      0      0      0      0      0
 2     0      0      0      0      0      0
 3     0      0      0      0      0      0
 4     0      0      0      0      0      0
 5     0      0      0      0      0      0
 6     0      0      0      0      0      0

Граф после заполнения Алисой:
       1      2      3      4      5      6
 1     0      1      1      1      1      0
 2     1      0      1      0      1      1
 3     1      1      0      1      0      1
 4     1      0      1      0      1      1
 5     1      1      0      1      0      1
 6     0      1      1      1      1      0

N = 6, M = 12

Переходы:
1: [2, 3, 4, 5]
2: [1, 3, 5, 6]
3: [1, 2, 4, 6]
4: [1, 3, 5, 6]
5: [1, 2, 4, 6]
6: [2, 3, 4, 5]

Гамильтонов цикл: [1, 2, 3, 4, 5, 6]
```

Рис 1. Заполнение первоначального графа Алисой

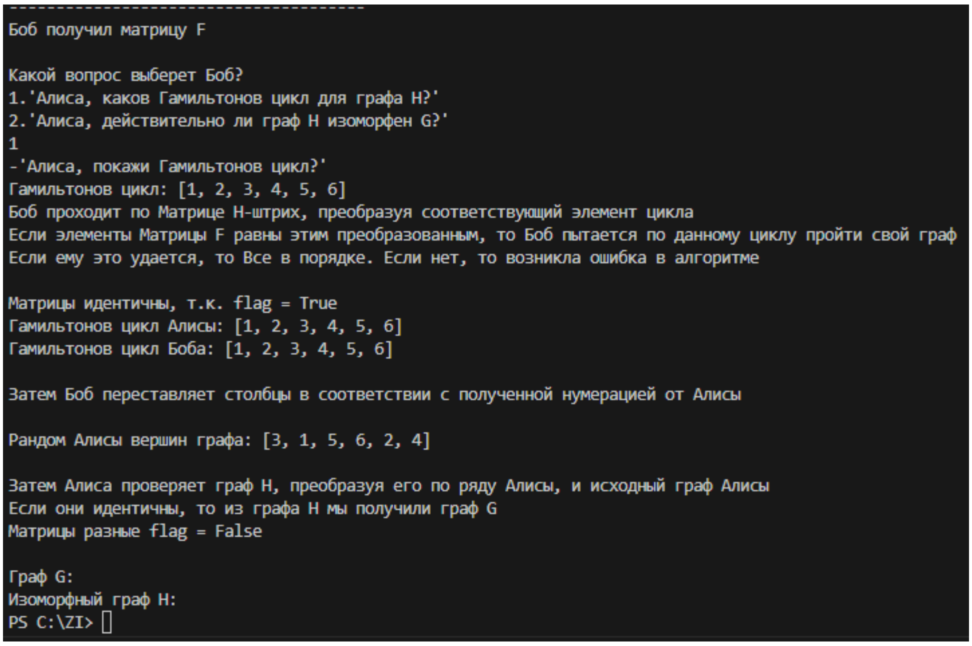
```
------------------------------------
Действия первого абонента - Алисы
------------------------------------
Алиса рандомит вершины графа: [2, 1, 6, 3, 4, 5]

Изоморфный граф:
       1      2      3      4      5      6
 1     0      1      1      1      0      1
 2     1      0      0      1      1      1
 3     1      0      0      1      1      1
 4     1      1      1      0      1      0
 5     0      1      1      1      0      1
 6     1      1      1      0      1      0

Перед кодировкой алгоритмом RSA
Припишем рандомное число из списка [3, 6, 4, 1, 2, 5]

Подготовленная матрица до RSA:
       1      2      3      4      5      6
 1     30     61     41     11     20     51
 2     31     60     40     11     21     51
 3     31     60     40     11     21     51
 4     31     61     41     10     21     50
 5     30     61     41     11     20     51
 6     31     61     41     10     21     50

Алиса генерирует ключи:
Ключ N:  626254885660641593
Ключ c:  584684543365727411
```

Рис 2. Преобразования графа



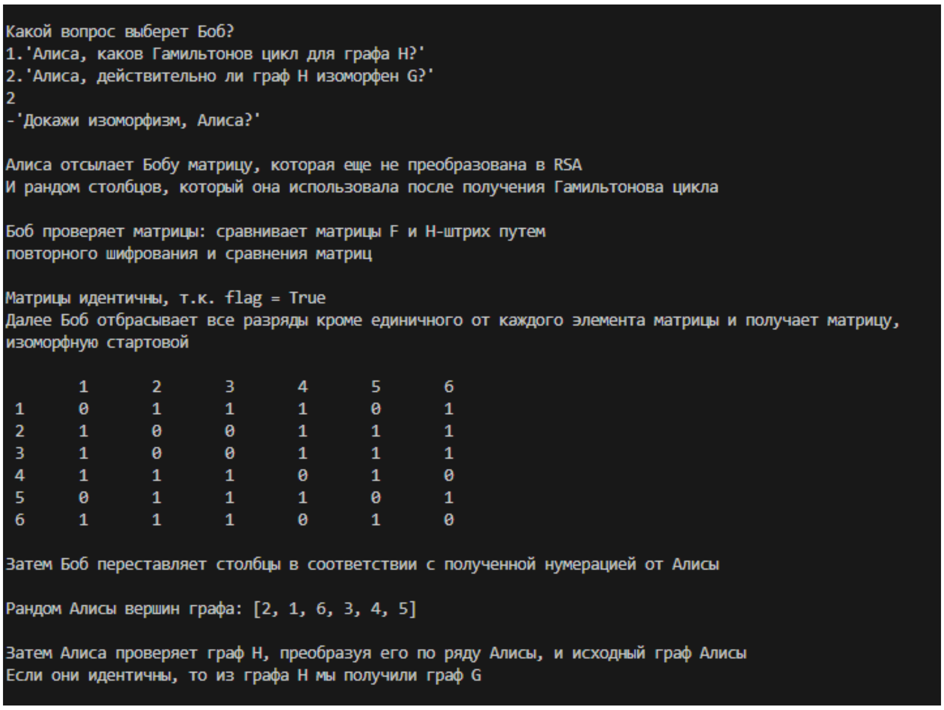
```
Матрица после RSA, для Боба:
        1       2       3       4       5       6
 1      15252418130520528       217359282987567340      55561489893212522       386152135722672132      543712695047543984      5218053513035
3454
 2      34042163984393534       541336093552297005      208340263166358489      386152135722672132      446652807698445016      5218053513035
3454
 3      34042163984393534       541336093552297005      208340263166358489      386152135722672132      446652807698445016      5218053513035
3454
 4      34042163984393534       217359282987567340      55561489893212522       150657202703658682      446652807698445016      3238241277395
46740
 5      15252418130520528       217359282987567340      55561489893212522       386152135722672132      543712695047543984      5218053513035
3454
 6      34042163984393534       217359282987567340      55561489893212522       150657202703658682      446652807698445016      3238241277395
46740

-------------------------------------
Действия Второго абонента - Боба
-------------------------------------
Боб получил матрицу F
```

Рис 3. Шифрование матрицы графа Алисой.



```
-------------------------------------
Боб получил матрицу F

Какой вопрос выберет Боб?
1.'Алиса, каков Гамильтонов цикл для графа H?'
2.'Алиса, действительно ли граф H изоморфен G?'
1
-'Алиса, покажи Гамильтонов цикл?'
Гамильтонов цикл: [1, 2, 3, 4, 5, 6]
Боб проходит по Матрице H-штрих, преобразуя соответствующий элемент цикла
Если элементы Матрицы F равны этим преобразованным, то Боб пытается по данному циклу пройти свой граф
Если ему это удается, то Все в порядке. Если нет, то возникла ошибка в алгоритме

Матрицы идентичны, т.к. flag = True
Гамильтонов цикл Алисы: [1, 2, 3, 4, 5, 6]
Гамильтонов цикл Боба: [1, 2, 3, 4, 5, 6]

Затем Боб переставляет столбцы в соответствии с полученной нумерацией от Алисы

Рандом Алисы вершин графа: [3, 1, 5, 6, 2, 4]

Затем Алиса проверяет граф H, преобразуя его по ряду Алисы, и исходный граф Алисы
Если они идентичны, то из графа H мы получили граф G
Матрицы разные flag = False

Граф G:
Изоморфный граф H:
PS C:\ZI>
```

Рис 4. Боб задает первый вопрос Алисе.



```
Какой вопрос выберет Боб?
1.'Алиса, каков Гамильтонов цикл для графа H?'
2.'Алиса, действительно ли граф H изоморфен G?'
2
-'Докажи изоморфизм, Алиса?'

Алиса отсылает Бобу матрицу, которая еще не преобразована в RSA
И рандом столбцов, который она использовала после получения Гамильтонова цикла

Боб проверяет матрицы: сравнивает матрицы F и H-штрих путем
повторного шифрования и сравнения матриц

Матрицы идентичны, т.к. flag = True
Далее Боб отбрасывает все разряды кроме единичного от каждого элемента матрицы и получает матрицу,
изоморфную стартовой

        1       2       3       4       5       6
 1      0       1       1       1       0       1
 2      1       0       0       1       1       1
 3      1       0       0       1       1       1
 4      1       1       1       0       1       0
 5      0       1       1       1       0       1
 6      1       1       1       0       1       0

Затем Боб переставляет столбцы в соответствии с полученной нумерацией от Алисы

Рандом Алисы вершин графа: [2, 1, 6, 3, 4, 5]

Затем Алиса проверяет граф H, преобразуя его по ряду Алисы, и исходный граф Алисы
Если они идентичны, то из графа H мы получили граф G
```

Рис 5. Боб задает второй вопрос Алисе.

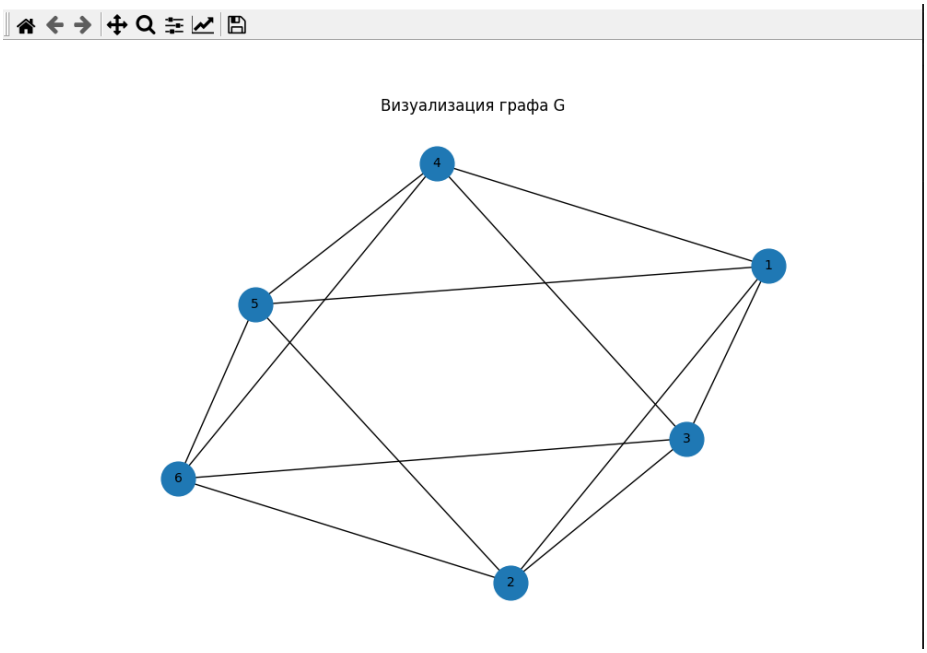


Рис 6. Визуализация графа G

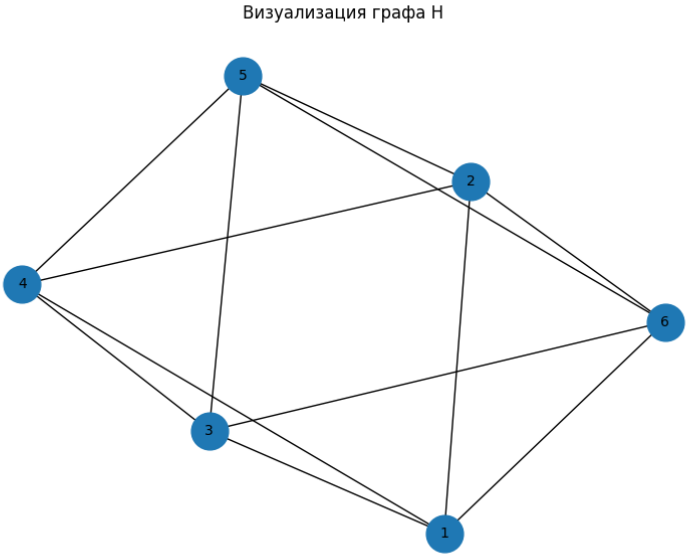


Рис 7. Визуализация графа H

Листинг кода

```
from collections import defaultdict
import random
import matplotlib.pyplot as plt
import networkx as nx
from lab2 import *
def check_prime(p):
    if p <= 1:
        return False
    elif p == 2:
        return True
    a = random.randint(2, p - 1)
    # print(p, "-", a)
    if pow_module(a, (p - 1), p) != 1
or gcd(p, a) > 1:
        return False
    return True
def generate_prime(left, right):
    while True:
        p = random.randint(left, right)
        # print("--", p)
        if check_prime(p):
            return p
def generate_coprime(p):
    result = random.randint(2, p)
    # print(result)
    while gcd(p, result) != 1:
        result = random.randint(2, p)
    # print(result)
    return result
def pow_module(a, x, p):
    result = 1
    a = a % p
    if a == 0:
        return 0;
    while x > 0:
        if x & 1 == 1:
            result = (result * a) % p
        a = (a ** 2) % p
        x >>= 1
def gcd(a, b):
    while b != 0:
        r = a % b
        a = b
        b = r
    return a
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


def
read_graph_from_file(file_path='Path.tx
t'):
    try:
        with open(file_path, 'r') as
file:
            lines =
file.read().splitlines()
    except OSError:
        print("Ошибка открытия файла с
графами")
        return -1


    N = int(lines[0])
    M = int(lines[1])
    node_ids = list(range(1, N + 1))
```

```
    header = "\t" + "\t".join(map(str,
node_ids))


    graph_G = [[0] * N for _ in
range(N)]
    graph_H = [[0] * N for _ in
range(N)]


    print("Стартовый граф:")
    print(header)


    for i in range(N):
        print(f" {node_ids[i]}",
end="\t")
        for j in range(N):
            graph_G[i][j] = 0
            print(graph_G[i][j],
end="\t")
        print()


    print("\nГраф после заполнения
Алисой:")
    print(header)


    for k in range(M):
        line = lines[k + 2]
        i, j = map(int,
line.split(","))
        graph_G[i - 1][j - 1] = 1
        graph_G[j - 1][i - 1] = 1
        graph_H[i - 1][j - 1] = 1
        graph_H[j - 1][i - 1] = 1


    for i in range(N):
        print(f" {node_ids[i]}",
end="\t")
        for j in range(N):
            print(graph_G[i][j],
end="\t")
        print()


    return N, M, graph_G, graph_H,
header


def visualize_graph(graph, header,
graph_type):
    G = nx.Graph()
    N = len(graph)


    for i in range(N):
        G.add_node(i + 1)


    for i in range(N):
        for j in range(N):
            if graph[i][j] == 1:
                G.add_edge(i + 1, j +
1)


    pos = nx.spring_layout(G)


    plt.figure(figsize=(10, 8))
    plt.title(f"Визуализация графа
{graph_type}")
    nx.draw(G, pos, with_labels=True,
node_size=700, font_size=10,
font_color="black")
    plt.show()


def all_paths(graph):
    paths = defaultdict(list)
    for i in range(len(graph)):
        for j in range(len(graph[i])):
            if graph[i][j] == 1:
                paths[i].append(j + 1)
    return paths
```

```
def search_hamilton_cycle(graph, size, pt, path=[]):
    if pt not in set(path):
        path.append(pt)
        if len(path) == size:
            return path
        for pt_next in graph.get(pt - 1, []):
            res_path = path.copy()
            candidate = search_hamilton_cycle(graph, size, pt_next, res_path)
            if candidate is not None:
                return candidate


def rsa_matrix_encode(new_graph, x, p, N):
    temp_graph = [[0] * N for _ in range(N)]
    for i in range(len(new_graph)):
        for j in range(len(new_graph[i])):
            temp_graph[i][j] = pow_module(new_graph[i][j], x, p)
    return temp_graph


def hamilton_cycle():
    N, M, graph_G, graph_H, header = read_graph_from_file()
    list_path = list()
    new_graph_H = [[0] * N for _ in range(N)]
    new_graph_G = [[0] * N for _ in range(N)]
    left_random = list(range(1, N + 1))
    random.shuffle(left_random)


    print(f"\nN = {N}, M = {M}")
    paths = all_paths(graph_G)
    print("\nПереходы:")
    for i in range(len(paths)):
        print(f"{i + 1}: {paths[i]}")


    cycle_path = search_hamilton_cycle(paths, N, 1, list_path)
    print(f"\nГамильтонов цикл: {cycle_path}\n")


    print("-----------------------------------------")
    print("Действия первого абонента - Алисы")
    print("-----------------------------------------")


    new_list = list(range(N))
    random.shuffle(new_list)
    new_list_str = [i + 1 for i in new_list]
    print(f"Алиса рандомит вершины графа: {new_list_str} \n")


    k, z = 0, 0
    for i in new_list:
        for j in new_list:
            new_graph_H[k][z] = graph_G[int(i)][int(j)]
            z += 1
        z = 0
        k += 1


    print("Изоморфный граф: ")
    print(header)
    for i in range(len(new_graph_H)):
        print(" " + str(i + 1), end="\t")
```

```
        for j in
range(len(new_graph_H[i])):
            print(new_graph_H[i][j],
end="\t")
        print("\n", end="")


    print("\nПеред кодировкой
алгоритмом RSA\n" +
          f"Припишем рандомное число из
списка {left_random}")


    k, z = 0, 0
    for i in left_random:
        for j in left_random:
            new_graph_H[k][z] =
int(str(j) + str(new_graph_H[k][z]))
            z += 1
        z = 0
        k += 1


    print("\nПодготовленная матрица до
RSA:")
    print(header)
    for i in range(len(new_graph_H)):
        print(" " + str(i + 1),
end="\t")
        for j in
range(len(new_graph_H[i])):
            print(new_graph_H[i][j],
end="\t")
        print("\n", end="")


    print("\nАлиса генерирует ключи: ")
    P = generate_prime(0, 10 ** 9)
    Q = generate_prime(0, 10 ** 9)
    N_encode = P * Q
    print("Ключ N: ", N_encode)
    Phi = (P - 1) * (Q - 1)
    d = generate_coprime(Phi)
    c = gcd_modified(d, Phi)[1]
    if c < 0:
        c += Phi
    print("Ключ c: ", c)
    graph_F =
rsa_matrix_encode(new_graph_H, d,
N_encode, N)


    print("\nМатрица после RSA, для
Боба:")
    print(header)
    for i in range(len(graph_F)):
        print(" " + str(i + 1),
end="\t")
        for j in
range(len(graph_F[i])):
            print(graph_F[i][j],
end="\t")
        print("\n", end="")


    print()
    print("----------------------------
----------")
    print("Действия Второго абонента -
Боба")
    print("----------------------------
----------")
    print("Боб получил матрицу F\n")
    print("Какой вопрос выберет Боб?")
    print("1.'Алиса, каков Гамильтонов
цикл для графа H?'")
    print("2.'Алиса, действительно ли
граф H изоморфен G?'")


    answer = int(input())
    if answer == 1:
        print("-'Алиса, покажи
Гамильтонов цикл?'")
        print(f"Гамильтонов цикл:
{cycle_path}")
```

```
        print("Боб проходит по Матрице H-штрих, преобразуя соответствующий элемент цикла")

        print("Если элементы Матрицы F равны этим преобразованным, то Боб пытается по данному циклу пройти свой граф\n"

              + "Если ему это удается, то Все в порядке. Если нет, то возникла ошибка в алгоритме")

        bob_check = rsa_matrix_encode(new_graph_H, d, N_encode, N)

        flag = all(bob_check[i][j] == graph_F[i][j] for i in range(len(graph_F)) for j in range(len(graph_F[i])))

        if flag:

            print(f"\nМатрицы идентичны, т.к. flag = {flag}")

            print(f"Гамильтонов цикл Алисы: {cycle_path}")

            print(f"Гамильтонов цикл Боба: {cycle_path}")

        else:

            print(f"Матрицы разные flag = {flag}")


    if answer == 2:

        print("-'Докажи изоморфизм, Алиса?'")

        print("\nАлиса отсылает Бобу матрицу, которая еще не преобразована в RSA\n" +

              "И рандом столбцов, который она использовала после получения Гамильтонова цикла")

        print("\nБоб проверяет матрицы: сравнивает матрицы F и H-штрих путем\n" +

              "повторного шифрования и сравнения матриц")

        bob_check = rsa_matrix_encode(new_graph_H, d, N_encode, N)

        flag = all(bob_check[i][j] == graph_F[i][j] for i in range(len(graph_F)) for j in range(len(graph_F[i])))

        if flag:

            print(f"\nМатрицы идентичны, т.к. flag = {flag}")

        else:

            print(f"\nМатрицы разные flag = {flag}")

        print("Далее Боб отбрасывает все разряды кроме единичного от каждого элемента матрицы и получает матрицу,\n" +

              "изоморфную стартовой\n")

        print(header)

        for i in range(len(new_graph_H)):

            print(" " + str(i + 1), end="\t")

            for j in range(len(new_graph_H[i])):

                num_str = str(new_graph_H[i][j])

                new_graph_H[i][j] = int(num_str[-1])


print(new_graph_H[i][j], end="\t")

            print("\n", end="")


    print("\nЗатем Боб переставляет столбцы в соответствии с полученной нумерацией от Алисы")

    print(f"\nРандом Алисы вершин графа: {new_list_str}")


    k, z = 0, 0

    for i in new_list:

        for j in new_list:

            new_graph_G[int(i)][int(j)] = new_graph_H[k][z]

            z += 1
```

```
        z = 0
        k += 1


    print("\nЗатем Алиса проверяет граф
H, преобразуя его по ряду Алисы, и
исходный граф Алисы\n" +
          "Если они идентичны, то из
графа H мы получили граф G")
    flag = all(graph_H[i][j] ==
new_graph_G[i][j] for i in
range(len(graph_H)) for j in
range(len(graph_H[i])))
    if flag:
        print()
        print(f"Матрицы идентичны, т.к.
flag = {flag}")
    else:
        print(f"Матрицы разные flag =
{flag}")


    print("\nГраф G:")
    visualize_graph(graph_G, header,
"G")


    print("Изоморфный граф H:")
    visualize_graph(new_graph_H,
header, "H")


if __name__ == '__main__':
    hamilton_cycle()
```