МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

**Доказательство с нулевым разглашением изоморфизма графов.**

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»

студента 5 курса 531 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Мызникова Сергея Анатольевича

|  |  |  |
| --- | --- | --- |
| Преподаватель, доцент | \_\_\_\_\_\_\_\_\_\_ | В.Е. Новиков |
|  | подпись, дата |  |

Саратов 2024

**1 Постановка задачи**

Цель работы: изучить доказательство с нулевым разглашением изоморфизма графов и привести его программную реализацию

**2 Теоретические сведения**

В криптографии Доказательство с нулевым разглашением (информации) (Zero-Knowledge Proof) — это интерактивный протокол, позволяющий одной из сторон (проверяющему) убедиться в достоверности какого-либо утверждения (обычно математического), не получив при этом никакой другой информации от второй стороны (доказывающего).

Доказательство с нулевым разглашением должно обладать тремя свойствами:

1. Полнота: если утверждение действительно верно, то доказывающий убедит в этом проверяющего.
2. Корректность: если утверждение неверно, то даже нечестный доказывающий не сможет убедить проверяющего за исключением пренебрежимо малой вероятности.
3. Нулевое разглашение: если утверждение верно, то даже нечестный проверяющий не узнает ничего кроме самого факта, что утверждение верно.

**Алгоритм генерации открытого и закрытого ключей**

Пусть Боб знает изоморфизм графов G0 и G1. Боб хочет доказать Алисе свое знание, не показывая сам изоморфизм.

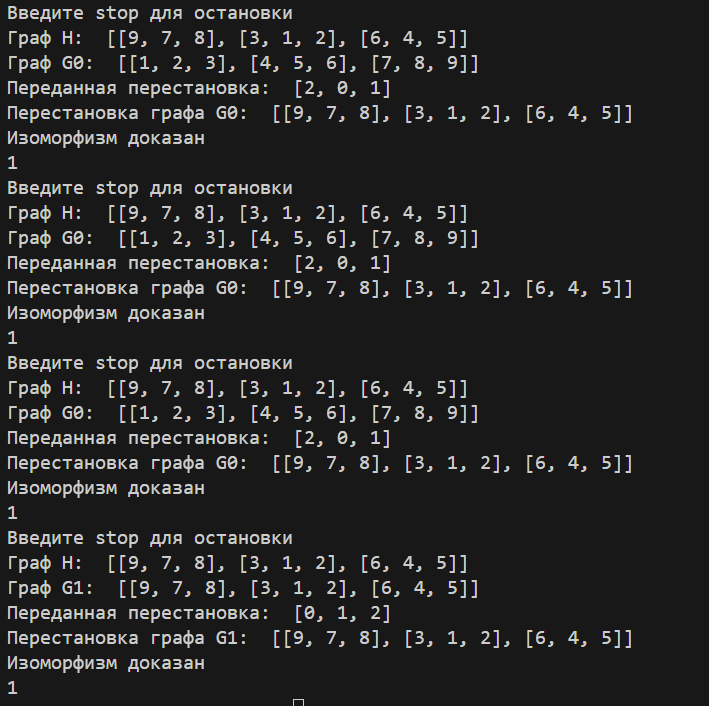
Общий вход пара графов G0 = (V0, E0) и G1 = (V1, E1). Пусть – изоморфизм между G0 и G1, т.е. биективное отображение множества вершин V0 на множество вершин V1 такое, что (u, v) E0 тогда и только тогда, когда ((u), (v)) E1

**Алгоритм**:

1. Боб выбирает случайную изоморфную копию графа G1 (обозначим как H) и отправляет ее проверяющей Алисе. Иначе говоря, выбирается случайная перестановка из множества перестановок.
2. Алиса выбирает случайный бит {0, 1} и отправляет его Бобу. Боб должен показать изоморфизм между .
3. Если сообщение, полученное от проверяющего равно 1, то Боб отправляет Алисе. В противном случае отправляется композиция .
4. Если изоморфизм графов доказан, то проверяющий выдает 1, иначе 0.

Если Алиса хочет провести несколько проверок, то для этого возвращаемся к шагу 2.

**3 Тестирование программы**



**ПРИЛОЖЕНИЕ А**

import random

def read\_matrix\_from\_file(filename):

with open(filename, 'r') as file:

matrix = [list(map(int, line.split())) for line in file]

return matrix

def write\_matrix\_to\_file(matrix, filename):

with open(filename, 'w') as file:

for row in matrix:

file.write(' '.join(map(str, row)) + '\n')

def generate\_permutation(matrix\_size, p):

permutation = list(range(matrix\_size))

random.shuffle(permutation)

while permutation == p:

random.shuffle(permutation)

return permutation

def write\_permutation\_to\_file(permutation, filename):

with open(filename, 'w') as file:

file.write(' '.join(map(str, permutation)) + '\n')

def read\_permutation\_from\_file(filename):

with open(filename, 'r') as file:

permutation = list(map(int, file.readline().split()))

return permutation

def apply\_permutation(matrix, permutation):

permuted\_matrix = [[matrix[i][j] for j in permutation] for i in permutation]

return permuted\_matrix

def mix\_permutation(permutation\_1, permutation\_2):

new\_permutation = [permutation\_2[i] for i in permutation\_1]

return new\_permutation

def main():

G0 = read\_matrix\_from\_file("matrix\_G0.txt")

p = read\_permutation\_from\_file("secret\_permutation.txt")

G1 = apply\_permutation(G0, p)

write\_matrix\_to\_file(G1, "matrix\_G1.txt")

h = generate\_permutation(len(G0), p)

write\_permutation\_to\_file(h, "permutation.txt")

H = apply\_permutation(G1, h)

write\_matrix\_to\_file(H, "matrix\_H.txt")

word = "-1"

while word != "stop":

word = input("Введите stop для остановки")

k = random.randint(0, 1)

if k == 1:

H = read\_matrix\_from\_file("matrix\_H.txt")

print("Граф H: ", H)

G1 = read\_matrix\_from\_file("matrix\_G1.txt")

print("Граф G1: ", G1)

h = read\_permutation\_from\_file("permutation.txt")

print("Переданная перестановка: ", h)

new\_G1 = apply\_permutation(G1, h)

print("Перестановка графа G1: ", new\_G1)

if H != new\_G1:

print("Изоморфизм не доказан")

print(0)

else:

print("Изоморфизм доказан")

print(1)

else:

p = read\_permutation\_from\_file("secret\_permutation.txt")

h = read\_permutation\_from\_file("permutation.txt")

G0 = read\_matrix\_from\_file("matrix\_G0.txt")

H = read\_matrix\_from\_file("matrix\_H.txt")

print("Граф H: ", H)

print("Граф G0: ", G0)

l = mix\_permutation(h, p)

print("Переданная перестановка: ", l)

new\_G0 = apply\_permutation(G0, l)

print("Перестановка графа G0: ", new\_G0)

if H != new\_G0:

print("Изоморфизм не доказан")

print(0)

else:

print("Изоморфизм доказан")

print(1)

return

main()import random

def generate\_regular\_graph(n, degree):

if (n \* degree) % 2 != 0:

raise ValueError("Для регулярного графа необходимо, чтобы число вершин \* степень вершин было четным")

matrix = [[0] \* n for \_ in range(n)]

for i in range(n):

for j in range(1, degree // 2 + 1):

matrix[i][(i + j) % n] = 1

matrix[(i + j) % n][i] = 1

return matrix

def read\_matrix\_from\_file(filename):

with open(filename, 'r') as f:

matrix = [list(map(int, line.split())) for line in f]

return matrix

def write\_matrix\_to\_file(matrix, filename):

with open(filename, 'w') as f:

for row in matrix:

f.write(' '.join(map(str, row)) + '\n')

def generate\_permutation(matrix\_size, p):

permutation = list(range(matrix\_size))

random.shuffle(permutation)

while permutation == p:

random.shuffle(permutation)

return permutation

def write\_permutation\_to\_file(permutation, filename):

with open(filename, 'w') as file:

file.write(' '.join(map(str, permutation)) + '\n')

def read\_permutation\_from\_file(filename):

with open(filename, 'r') as file:

permutation = list(map(int, file.readline().split()))

return permutation

def apply\_permutation(matrix, permutation):

permuted\_matrix = [[matrix[i][j] for j in permutation] for i in permutation]

return permuted\_matrix

def mix\_permutation(permutation\_1, permutation\_2):

new\_permutation = [permutation\_2[i] for i in permutation\_1]

return new\_permutation

def print\_graph(matrix):

for row in matrix:

print(" | ".join(f"{val:2}" for val in row))

def main():

o = input("Введите 1 для генерации параметров\nВведите 2 для проверки доказательства\n")

if o == "1":

n = int(input("Введите количество вершин графа: "))

degree = int(input("Введите степени вершин у графа: "))

tmp = generate\_regular\_graph(n, degree)

write\_matrix\_to\_file(tmp, "matrix\_G0.txt")

G0 = read\_matrix\_from\_file("matrix\_G0.txt")

mas = list(range(0, len(G0)))

tmp = generate\_permutation(len(G0), mas)

write\_permutation\_to\_file(tmp, "secret\_permutation.txt")

p = read\_permutation\_from\_file("secret\_permutation.txt")

G1 = apply\_permutation(G0, p)

write\_matrix\_to\_file(G1, "matrix\_G1.txt")

h = generate\_permutation(len(G0), p)

write\_permutation\_to\_file(h, "permutation.txt")

H = apply\_permutation(G1, h)

write\_matrix\_to\_file(H, "matrix\_H.txt")

elif o == "2":

word = "-1"

while word != "stop":

word = input("Введите stop для остановки")

k = random.randint(0, 1)

if k == 1:

H = read\_matrix\_from\_file("matrix\_H.txt")

print("Граф H: ")

print\_graph(H)

G1 = read\_matrix\_from\_file("matrix\_G1.txt")

print("Граф G1: ")

print\_graph(G1)

h = read\_permutation\_from\_file("permutation.txt")

print("Переданная перестановка: ", h)

new\_G1 = apply\_permutation(G1, h)

print("Перестановка графа G1: ")

print\_graph(new\_G1)

if H != new\_G1:

print("Изоморфизм не доказан")

print(0)

else:

print("Изоморфизм доказан")

print(1)

else:

p = read\_permutation\_from\_file("secret\_permutation.txt")

h = read\_permutation\_from\_file("permutation.txt")

G0 = read\_matrix\_from\_file("matrix\_G0.txt")

H = read\_matrix\_from\_file("matrix\_H.txt")

print("Граф H: ")

print\_graph(H)

print("Граф G0: ")

print\_graph(G0)

l = mix\_permutation(h, p)

print("Переданная перестановка: ", l)

new\_G0 = apply\_permutation(G0, l)

print("Перестановка графа G0: ")

print\_graph(new\_G0)

if H != new\_G0:

print("Изоморфизм не доказан")

print(0)

else:

print("Изоморфизм доказан")

print(1)

return

main()