Министерство цифрового развития, связи и
массовых коммуникаций Российской Федерации

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

Отчет
по расчетно-графической работе
по дисциплине «**Защита информации**»

Реализация протокола доказательства с нулевым знанием для задачи "Гамильтонов цикл"

Выполнил:
студент гр. ИС-142 _____ /Наумов А.А./

Проверил:
преподаватель _____ /Истомина А.С./

Новосибирск 2024

ВВЕДЕНИЕ

В данной работе рассматривается протокол доказательства с нулевым знанием (Zero-Knowledge Proof) на примере задачи поиска гамильтонова цикла. Данный протокол позволяет одной стороне (доказывающему) доказать другой стороне (проверяющему) наличие решения задачи (гамильтонова цикла) без раскрытия самого решения. Задача нахождения гамильтонова цикла является NP-полной, что подчёркивает её сложность и актуальность.

Цель работы — разработка программы, реализующей протокол доказательства с нулевым знанием для задачи поиска гамильтонова цикла, и её тестирование с использованием сгенерированных графов.

ПОСТАНОВКА ЗАДАЧИ

В рамках работы необходимо:

1. Реализовать алгоритм для работы с графами, включающий:
    - Чтение графа и гамильтонова цикла из файлов.
    - Генерацию гамильтонова цикла при отсутствии информации в файле.
    - Перестановку вершин графа для создания изоморфного графа.
2. Реализовать протокол доказательства с нулевым знанием для проверки наличия гамильтонова цикла.
3. Тестировать корректность протокола на сгенерированных данных.
4. Реализовать текстовую визуализацию графа и гамильтонова цикла.
5. Оформить протокол и вывод результатов в удобочитаемом виде.

Формат входных данных:

1. Файл с графом содержит:
    - Количество вершин и рёбер в первой строке.
    - Перечень рёбер в формате "u v" (две вершины, соединённые ребром) в следующих строках.
2. Файл с гамильтоновым циклом (при наличии) содержит номера вершин цикла в одной строке.

ВЫПОЛНЕНИЕ РАБОТЫ

В рамках данной работы реализована программа, демонстрирующая протокол доказательства с нулевым знанием для задачи нахождения гамильтонова цикла. Программа состоит из набора функций, каждая из которых выполняет определенную задачу:

1. read_graph_from_file

Эта функция выполняет чтение графа из текстового файла.

- Аргументы:
  - **filename**: путь к файлу, содержащему описание графа.
  - **cycle_filename** (опционально): путь к файлу с информацией о гамильтоновом цикле.
- Возвращаемые значения:
  - **graph**: граф, представленный в виде словаря списков смежности.
  - **n**: количество вершин в графе.
  - **m:** количество рёбер в графе.
  - **hamiltonian_cycle**: гамильтонов цикл (если указан файл с циклом).

- Логика работы:

1. Читает количество вершин и рёбер из первой строки файла.
2. Заполняет словарь графа на основе списка рёбер, указанных в файле.
3. Если указан файл с гамильтоновым циклом, считывает его.

2. shuffle_graph

Эта функция создает изоморфный граф путём случайной перестановки вершин.

- Аргументы:
  - **graph**: исходный граф.
  - **n:** количество вершин.
- Возвращаемые значения:
  - **new_graph**: изоморфный граф после перестановки вершин.
  - **permutation:** список перестановок (каждому индексу соответствует новый номер вершины).
- Логика работы:

1. Генерирует случайную перестановку вершин.
2. Строит новый граф, используя перестановку.

3. apply_permutation

Эта функция преобразует гамильтонов цикл в соответствии с перестановкой вершин.

- Аргументы:
  - **cycle**: исходный гамильтонов цикл.
  - **permutation**: перестановка вершин.
- Возвращаемые значения:
  - Новый гамильтонов цикл, соответствующий перестановке.

- Логика работы:

1. Создаёт обратную перестановку для соответствия оригинальных индексов.
2. Применяет эту перестановку к вершинам цикла.

4. generate_hamiltonian_cycle

Эта функция генерирует случайный гамильтонов цикл.

- Аргументы:

- **n**: количество вершин.
- Возвращаемое значение:
  - Список вершин, задающий гамильтонов цикл.

- Логика работы:

1. Создаёт список всех вершин графа.
2. Перемешивает их в случайном порядке.

5. verify_hamiltonian_cycle

Функция проверяет, является ли переданный цикл гамильтоновым для указанного графа.

- Аргументы:
  - **graph**: граф в виде словаря списков смежности.
  - **cycle**: проверяемый цикл.
- Возвращаемое значение:
  - **True**, если цикл корректен; **False** в противном случае.

- Логика работы:

1. Проверяет, соединены ли последовательно все вершины цикла.
2. Убеждается, что цикл замкнут, то есть последняя вершина соединена с первой.

6. zero_knowledge_proof

Главная функция, реализующая протокол доказательства с нулевым знанием.

- Аргументы:
  - **graph**: исходный граф.
  - **hamiltonian_cycle**: гамильтонов цикл.
  - **n**: количество вершин.
  - **rounds**: количество раундов для протокола.
- Возвращаемое значение:
  - **True,** если протокол успешно завершен; **False** при ошибке.

- Логика работы:

1. Для каждого раунда:
   - Перемешивает граф, создавая его изоморф.
   - Проверяющий случайным образом выбирает один из запросов:
     - Показать перестановку вершин (изоморфизм).
     - Показать гамильтонов цикл в новом графе.
   - Протокол возвращает либо перестановку, либо перестроенный цикл.
2. Если в любом раунде проверка завершается с ошибкой, протокол считается неуспешным.

7. visualize_graph

Функция выводит текстовое представление графа и, при наличии, выделяет гамильтонов цикл.

- Аргументы:
  - **graph**: граф в виде словаря списков смежности.
  - **hamiltonian_cycle** (опционально): гамильтонов цикл.

- Логика работы:

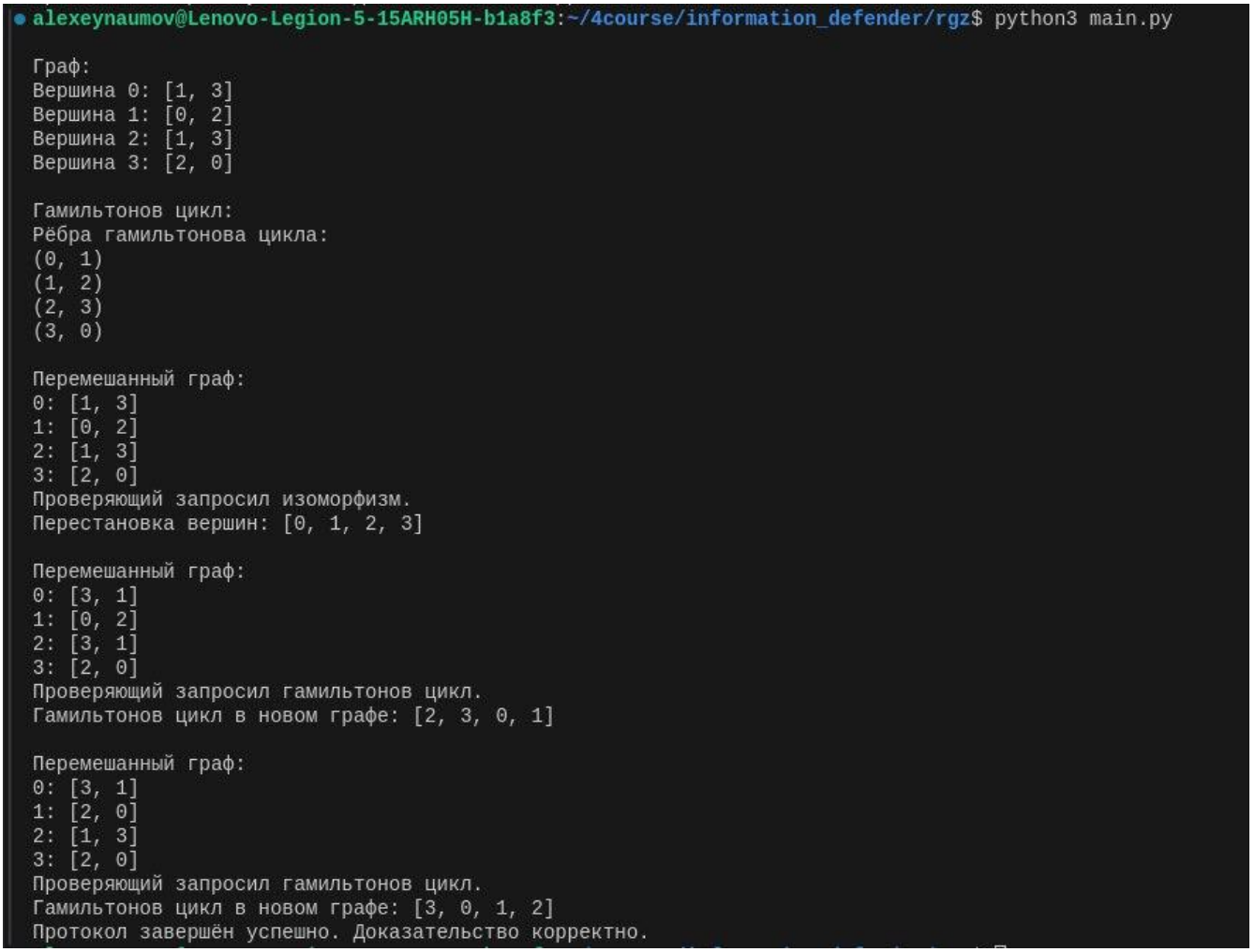
1. Выводит вершины графа и их смежные вершины.
2. Если задан гамильтонов цикл, выводит рёбра, которые входят в цикл.

8. main

Основная функция программы.

- Логика работы:
    1. Загружает граф и, при необходимости, генерирует гамильтонов цикл.
    2. Визуализирует исходный граф.
    3. Запускает протокол доказательства с нулевым знанием.
    4. Выводит результат работы протокола.

ДЕМОНСТРАЦИЯ РАБОТЫ



ЗАКЛЮЧЕНИЕ

В рамках работы была успешно реализована программа для проверки наличия гамильтонова цикла с использованием протокола доказательства с нулевым знанием. Программа считывает данные из файлов, обрабатывает графы, выполняет перестановку вершин для создания изоморфных графов и корректно проверяет наличие гамильтонова цикла.

Протокол был протестирован на различных графах, что подтвердило его функциональность и соответствие требованиям. Реализация протокола демонстрирует возможность безопасной передачи доказательств без раскрытия решения задачи.

ПРИЛОЖЕНИЕ

Исходный код программы

Файл Path.txt:

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

Файл Hamiltonian_cycle.py:

```
0 1 2 3
```

Файл **main.py**:

```
import random
import os
import networkx as nx
import matplotlib.pyplot as plt
from collections import defaultdict

# Чтение графа из файла
def read_graph_from_file(filename, cycle_filename=None):
    with open(filename, 'r') as f:
        lines = f.readlines()
        # Читаем количество вершин и рёбер
        n, m = map(int, lines[0].split())
        graph = defaultdict(list)

        # Заполняем граф рёбрами
        for line in lines[1:m+1]:  # Используем строки из списка lines
            u, v = map(int, line.split())
            graph[u].append(v)
            graph[v].append(u)

        # Дополнительная информация (например, гамильтонов цикл)
        hamiltonian_cycle = None
        if cycle_filename and os.path.exists(cycle_filename):
            with open(cycle_filename, 'r') as cycle_file:
                hamiltonian_cycle = list(map(int, cycle_file.readline().split()))

        return graph, n, m, hamiltonian_cycle

# Перестановка вершин графа для создания изоморфного графа
def shuffle_graph(graph, n):
    permutation = list(range(n))
    random.shuffle(permutation)
    new_graph = {i: [] for i in range(n)}
    for u in range(n):
        for v in graph[u]:
            new_graph[permutation[u]].append(permutation[v])
    return new_graph, permutation

# Применение обратной перестановки
def apply_permutation(cycle, permutation):
    inverse_perm = {v: k for k, v in enumerate(permutation)}
    return [inverse_perm[v] for v in cycle]

# Генерация гамильтонова цикла
def generate_hamiltonian_cycle(n):
    cycle = list(range(n))
    random.shuffle(cycle)
    return cycle
```

```
# Проверка гамильтонова цикла в графе
def verify_hamiltonian_cycle(graph, cycle):
    n = len(graph)
    for i in range(n):
        u, v = cycle[i], cycle[(i + 1) % n]
        if v not in graph[u]:
            return False
    return True

# Протокол доказательства с нулевым знанием
def zero_knowledge_proof(graph, hamiltonian_cycle, n, rounds=3):
    for _ in range(rounds):
        # Шаг 1: Перемешать вершины графа
        shuffled_graph, permutation = shuffle_graph(graph, n)
        print("\nПеремешанный граф:")
        for i in range(n):
            print(f"{i}: {shuffled_graph[i]}")

        # Шаг 2: Проверяющий делает запрос
        challenge = random.choice(["isomorphism", "cycle"])

        if challenge == "isomorphism":
            # Запрос: Показать изоморфизм
            print("Проверяющий запросил изоморфизм.")
            print(f"Перестановка вершин: {permutation}")
        elif challenge == "cycle":
            # Запрос: Показать гамильтонов цикл
            print("Проверяющий запросил гамильтонов цикл.")
            new_cycle = apply_permutation(hamiltonian_cycle, permutation)
            if verify_hamiltonian_cycle(shuffled_graph, new_cycle):
                print(f"Гамильтонов цикл в новом графе: {new_cycle}")
            else:
                print("Ошибка: гамильтонов цикл неверен.")
                return False
        else:
            print("Ошибка протокола.")
            return False

    return True

# Визуализация графа текстовым выводом
def visualize_graph(graph, hamiltonian_cycle=None):
    # Печать рёбер графа
    print("\nГраф:")
    for node, neighbors in graph.items():
        print(f"Вершина {node}: {neighbors}")

    # Если есть гамильтонов цикл, выделяем его
    if hamiltonian_cycle:
        print("\nГамильтонов цикл:")
        cycle_edges = [(hamiltonian_cycle[i], hamiltonian_cycle[i+1]) for i in
range(len(hamiltonian_cycle) - 1)]
        cycle_edges.append((hamiltonian_cycle[-1], hamiltonian_cycle[0]))  # Замкнуть
цикл
        print("Рёбра гамильтонова цикла:")
        for edge in cycle_edges:
            print(edge)


# Тестовый пример
def main():
    filename = 'Path.txt' # Название файла с графом
    graph, n, m, hamiltonian_cycle = read_graph_from_file('Path.txt',
'hamiltonian_cycle.txt')

    if not hamiltonian_cycle:
        hamiltonian_cycle = generate_hamiltonian_cycle(n)
```

```
        print(f"Генерация гамильтонова цикла: {hamiltonian_cycle}")

    # Визуализация графа
    visualize_graph(graph, hamiltonian_cycle)

    # Запуск протокола доказательства
    result = zero_knowledge_proof(graph, hamiltonian_cycle, n)
    if result:
        print("Протокол завершён успешно. Доказательство корректно.")
    else:
        print("Ошибка протокола. Доказательство не удалось.")

if __name__ == "__main__":
    main()
```