МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Навчалньно-науковий інститут електричної інженерії та інформаційних технологій

Кафедра автоматизації та інформаційних систем

**ПРАКТИЧНА РОБОТА**

Виконав: студент групи КІ-23-1

Черниш В’ячеслав Олександрович

Перевірив:

Сидоренко Валерій Миколайович

м. Кременчук

2024 рік

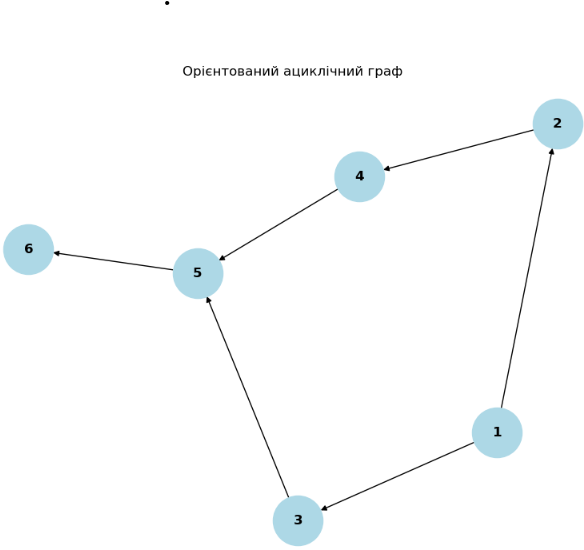
**Практична робота № 5**

**Тема**. **Графи. Ациклічні графи**

**Мета:** набути практичних навичок розв’язання задач топографічного сортування та оцінювання їх асимптотичної складності.

**Хід роботи**

20. Задано ациклічний граф: {1,2,3,4,5}{(1,2),(1,3),(2,4),(3,5),(4,5), (5, 6)}. Побудувати граф і розв’язати задачу топологічного сортування за допомогою алгоритму DFS.



from collections import defaultdict

# Реалізація графа

graph = {

1: [2, 3],

2: [4],

3: [5],

4: [5],

5: [6],

6: []

}

# Алгоритм DFS для топологічного сортування

def dfs\_topological\_sort(graph):

visited = set() # Множина відвіданих вершин

result = [] # Список для зберігання результату

def dfs(node):

if node in visited:

return

visited.add(node)

for neighbor in graph[node]:

dfs(neighbor)

result.append(node)

for vertex in graph:

if vertex not in visited:

dfs(vertex)

return result[::-1] # Зворотній порядок списку

# Викликаємо функцію для сортування

topological\_order = dfs\_topological\_sort(graph)

print("Топологічне сортування:", topological\_order)

Пропоноване топологічне сортування: [1, 3, 2, 4, 5, 6]

**Контрольні питання**

**1.Які переваги і недоліки алгоритму Кана порівняно з алгоритмом DFS для топологічного сортування графа?**

**Переваги алгоритму Кана**:

**Простота реалізації**: Алгоритм Кана простіший для розуміння та реалізації, оскільки використовує лише чергу для обробки вершин без необхідності рекурсії.

**Чітка інтеграція з пріоритетами**: Легше працювати з задачами, де необхідно знаходити мінімальні або максимальні вершини для подальшої обробки.

**Менше ймовірність помилок**: Не вимагає рекурсивного виклику, тому він більш надійний у великих графах, де можуть виникнути проблеми з переповненням стека.

**Недоліки алгоритму Кана**:

**Потребує додаткової пам'яті**: Для підтримки черги потрібна додаткова пам'ять, що може бути недоліком у великих графах.

**Переваги DFS**:

**Менше використання пам'яті**: DFS працює на основі рекурсії, що дозволяє зменшити потребу в додатковій пам'яті (за умови наявності оптимізації стека).

**Гнучкість у застосуванні**: Алгоритм DFS можна модифікувати для використання в різних задачах (наприклад, пошук шляхів або перевірка на наявність циклів).

**Недоліки DFS**:

**Складність реалізації**: Реалізація DFS для топологічного сортування вимагає більше уваги, оскільки потрібно враховувати рекурсію, стек і відвідувані вершини.

**Переповнення стека**: Для дуже великих графів з глибокими деревами рекурсія може призвести до переповнення стека.

**2.Яка складність часу і пам’яті для кожного з алгоритмів у найгіршому і найкращому випадках?**

**Алгоритм Кана**:

**Часова складність**:

**Найгірший випадок**: O(V+E), де V — кількість вершин, E — кількість ребер. У найгіршому випадку потрібно пройти всі вершини та ребра.

**Найкращий випадок**:)O(V+E), оскільки навіть при малих змінах структура графа потребує все одно обробки всіх вершин і ребер.

**Просторова складність**:

O(V) для збереження черги та масиву для відслідковування ступеня вхідних ребер.

**Алгоритм DFS**:

**Часова складність**:

**Найгірший випадок**:O(V+E), оскільки кожну вершину і ребро потрібно відвідати.

**Найкращий випадок**: O(V+E), оскільки кожен елемент буде оброблений у будь-якому випадку.

**Просторова складність**:

O(V) для збереження стану відвіданих вершин і викликів рекурсії (в найгіршому випадку глибина рекурсії може бути до V).

**3.Чи можна застосовувати алгоритм Кана до графів з вагами на ребрах? Як це порівняти з DFS?**

Алгоритм Кана не може бути безпосередньо застосований до графів з вагами на ребрах, оскільки він не враховує ці ваги при виконанні топологічного сортування. Він просто орієнтується на порядок обробки вершин на основі їх ступеня входу.

У випадку DFS, топологічне сортування також не залежить від ваг на ребрах. Однак, якщо задача потребує обробки ваг, можна модифікувати алгоритм для врахування ваг під час сортування (але це вже буде інша задача, наприклад, пошук найкоротшого шляху або критичного шляху).

**4.Як впливає структура графа на швидкість роботи кожного з цих алгоритмів?**

**Алгоритм Кана** буде ефективним для графів з малим числом циклів і коли багато вершин мають ступінь входу рівний нулю. У таких випадках, він буде швидко обробляти граф за допомогою черги.

**DFS** працює ефективно для графів, де є багато глибоких зв'язків і якщо потрібно виконати додаткові операції після проходу графа (наприклад, пошук компонент зв'язності або циклів).

**5.Чи є обмеження використання кожного алгоритму для певних типів графів або завдань?**

**Алгоритм Кана**:

Не працює для графів з циклами, оскільки вони не можуть мати топологічного сортування.

Підходить для орієнтованих ациклічних графів (DAG), де є чітке визначення залежностей.

**DFS**:

Також не працює для графів з циклами, оскільки не зможе виконати топологічне сортування в присутності циклів (можна використовувати DFS для виявлення циклів).

Підходить для загальних орієнтованих ациклічних графів.

**6.Які варіанти оптимізації можна застосувати для кожного алгоритму з метою поліпшення його продуктивності?**

**Алгоритм Кана**:

Використання пріоритетних черг (наприклад, черги з мінімальним елементом) для того, щоб обробляти вершини з найменшими залежностями першими.

Оптимізація вбудованих структур даних для зменшення часу обробки ступенів входу та ребер.

**DFS**:

Використання ітеративної заміни рекурсії (стек замість рекурсії) для уникнення переповнення стека в великих графах.

Виявлення циклів у графах під час обходу та завершення пошуку при їх виявленні.