МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Кафедра комп’ютерної інженерії та електроніки

Виконала: Групи

Пугачова Д. В. КІ-23-1

Кременчук 2024

Практична робота №5

Тема. Графи. Ациклічні графи

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

Завдання

**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 Кана.**

-побудова графа

6

4

2

1

5

3

**-Вихідні данні:**

Вершини: {1,2,3,4,5,6}\{1, 2, 3, 4, 5, 6\}{1,2,3,4,5,6}

Ребра: {(1,2),(1,3),(2,4),(3,5),(4,5),(5,6)}\{(1,2), (1,3), (2,4), (3,5), (4,5), (5,6)\}{(1,2),(1,3),(2,4),(3,5),(4,5),(5,6)}

-обчислення степеня входу для кожної вершини:

* Степінь входу для вершини 1: 0 (немає входящих ребер)
* Степінь входу для вершини 2: 1 (від 1)
* Степінь входу для вершини 3: 1 (від 1)
* Степінь входу для вершини 4: 1 (від 2)
* Степінь входу для вершини 5: 2 (від 3 і 4)
* Степінь входу для вершини 6: 1 (від 5)

**-Код:**

from collections import deque, defaultdict

def topological\_sort\_kahn(graph, vertices\_count):

# Степінь входу для кожної вершини

in\_degree = {i: 0 for i in range(1, vertices\_count + 1)}

# Обчислюємо степінь входу для кожної вершини

for u in graph:

for v in graph[u]:

in\_degree[v] += 1

# Черга для зберігання вершин зі степенем входу 0

queue = deque([v for v in in\_degree if in\_degree[v] == 0])

# Результат топологічного сортування

top\_order = []

while queue:

# Витягуємо вершину з черги

node = queue.popleft()

top\_order.append(node)

# Зменшуємо степінь входу для сусідів

for neighbor in graph[node]:

in\_degree[neighbor] -= 1

if in\_degree[neighbor] == 0:

queue.append(neighbor)

# Перевірка на наявність циклів (якщо топологічне сортування неможливе)

if len(top\_order) == vertices\_count:

return top\_order

else:

return "Граф має цикл, топологічне сортування неможливе"

# Опис графа

graph = {

1: [2, 3],

2: [4],

3: [5],

4: [5],

5: [6],

6: [] # Вершина 6 не має вихідних ребер

}

# Кількість вершин

vertices\_count = 6

# Викликаємо функцію для топологічного сортування

result = topological\_sort\_kahn(graph, vertices\_count)

print("Топологічний порядок:", result)

Топологічний порядок: [1, 2, 3, 4, 5, 6]

**КОНТРОЛЬНІ ПИТАННЯ**

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

-Алгоритм Кана:

Переваги:

* **Ітеративний підхід**: Алгоритм Кана працює за допомогою черги і поступово видаляє вершини з нульовим степенем входу. Це може бути корисно для великих графів, оскільки він не використовує рекурсію, як DFS.
* **Менша ймовірність стекового переповнення**: Оскільки алгоритм Кана ітеративний, він уникатиме проблеми переповнення стека, яка може виникнути в DFS при глибоких рекурсивних викликах.
* **Легше розпаралелити**: Алгоритм Кана більш підходить для паралельних обчислень, оскільки вершини з нульовим степенем входу можуть оброблятися одночасно.

**Недоліки:**

* **Потрібно зберігати степінь входу**: Алгоритм Кана вимагає зберігання і оновлення степеня входу для всіх вершин, що може потребувати додаткової пам'яті.
* **Не завжди ефективний для графів з високою щільністю**: Для графів з великою кількістю ребер алгоритм Кана може бути менш ефективним через необхідність постійно оновлювати степінь входу.

DFS:

Переваги

* **Простота**: Алгоритм DFS дуже простий для реалізації і не потребує додаткових структур даних, як у алгоритмі Кана.
* **Менше пам'яті в деяких випадках**: Якщо граф має низьку щільність, DFS може використовувати менше пам'яті, оскільки не потрібно зберігати степінь входу для кожної вершини.

**Недоліки:**

* **Рекурсія**: У випадку великої глибини графа може виникнути переповнення стека.
* **Складніше паралелити**: DFS важче розпаралелювати, оскільки обробка вершин залежить від глибини рекурсії.

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

***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 призначений для графів без циклів, але він працює тільки з топологічним порядком, а не з вагами ребер. Це означає, що ваги ребер не враховуються в алгоритмі Кана. Якщо потрібно врахувати ваги при сортуванні, то слід використовувати інші алгоритми, наприклад, Алгоритм Дейкстри або Алгоритм Беллмана-Форда для знаходження найкоротших шляхів.*

*У DFS також можна додавати ваги до ребер під час обходу, але це не дасть вам топологічного порядку — DFS просто виконує обхід, і додавання ваги змінює лише спосіб пошуку мінімального шляху, якщо це необхідно. Таким чином, для топологічного сортування DFS також не підходить, якщо є ваги ребер.*

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

** ***Алгоритм Кана****: Якщо граф має багато ребер, то алгоритм Кана може стати менш ефективним, оскільки для кожної вершини потрібно оновлювати степінь входу кожного її сусіда. Однак, у випадку розріджених графів, його ефективність залишається високою.*

** ***DFS****: Для графів з великою кількістю вершин DFS може мати високу складність через рекурсивний характер алгоритму, особливо якщо граф має велику глибину або складну структуру зв'язності.*

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

** ***Алгоритм Кана*** *працює тільки для ациклічних орієнтованих графів (DAG). Якщо в графі є цикл, алгоритм не зможе виконати топологічне сортування.*

** ***DFS*** *також можна використовувати для ациклічних орієнтованих графів, але він потребує додаткових перевірок для виявлення циклів під час обходу, якщо ми хочемо впевнитись, що граф є DAG.*

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

***Алгоритм Кана:***

* ***Зменшення часу на оновлення степеня входу****: Використання структур даних, таких як списки суміжності з оптимізованим доступом до сусідів, може допомогти пришвидшити оновлення степеня входу.*
* ***Паралельне виконання****: Якщо є можливість, можна паралелити обробку вершин, оскільки вершин з нульовим степенем входу можна обробляти незалежно одна від одної.*

***Алгоритм DFS:***

* ***Ітеративний DFS****: Для уникнення проблем зі стеком при глибоких рекурсіях можна використовувати ітеративний підхід з власним стеком.*
* ***Оптимізація з використанням збереження результатів****: Для деяких завдань можна використовувати кешування або мемоізацію для уникнення повторних обчислень під час обходу.*