МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕЛЕКТРИЧНОЇ ІНЖЕНЕРІЇ

ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП’ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ЕЛЕКТРОНІКИ

ЗВІТ  
З ПРАКТИЧНОЇ РОБОТИ №5

Виконала:  
студентка групи КН-24-1  
Лабущак А.В.

Перевірив:  
доцент кафедри КІЕ  
Сидоренко В. М.

Кременчук 2025

# Тема: Графи. Ациклічні графи

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

### Теоретичні відомості

**Орієнтований граф (ОГ)** - це математична структура, що складається з множини вершин та множини ребер, де кожне ребро має напрямок.

Формально ОГ визначається як G = (V, E), де:

* V - множина вершин
* E - множина ребер
* (vᵢ, vⱼ) - ребро від вершини vᵢ до вершини vⱼ

**Орієнтований ациклічний граф (ОАГ)** - це орієнтований граф, який не містить циклів. ОАГ використовується для моделювання систем з частковим порядком, планування проектів та топологічного сортування.

**Топологічне сортування** - це упорядкування вершин ОАГ таким чином, що для кожного ребра (u,v) вершина u йде перед вершиною v в результуючій послідовності.

**Алгоритм Кана**

**Алгоритм Кана** - це один з найпоширеніших алгоритмів для топологічного сортування. Розглянемо алгоритм, запропонований Каном у 1962 р.:

**Крок за кроком:**

1. **L** ← Порожній список, що буде містити відсортовані елементи
2. **S** ← Набір вершин без ребер, що входять (початкові вершини)
3. **Поки S не порожнє, виконати:**
   * Видалити вершину n з S
   * Додати n в L
   * Для кожної вершини m з ребром e з n до m:
     + Видалити ребро e з графа
     + Якщо m не має більше ребер, що входять:
       - Додати m в S
4. **Якщо граф має ребра, то:**
   * Граф має принаймні один цикл
5. **Інакше:**
   * Граф є ациклічним (топологічне сортування: L)

## Індивідуальне завдання (варіант 11)

**Задано ациклічний граф:** {1,2,3,4,5,6,7}|{(1,2),(1,3),(2,4),(3,5),(4,5),(4,6),(6,7)}

**Побудувати граф і розв'язати задачу топологічного сортування за допомогою алгоритму Кана.**

### Розв'язання

**Початкові дані:**

* Вершини: {1, 2, 3, 4, 5, 6, 7}
* Ребра: {(1,2), (1,3), (2,4), (3,5), (4,5), (4,6), (6,7)}

**Аналіз вхідних степенів:**

* Вершина 1: вхідний степінь = 0
* Вершина 2: вхідний степінь = 1 (ребро 1→2)
* Вершина 3: вхідний степінь = 1 (ребро 1→3)
* Вершина 4: вхідний степінь = 1 (ребро 2→4)
* Вершина 5: вхідний степінь = 2 (ребра 3→5, 4→5)
* Вершина 6: вхідний степінь = 1 (ребро 4→6)
* Вершина 7: вхідний степінь = 1 (ребро 6→7)

**Покрокове виконання алгоритму Кана:**

**Крок 1:**

* L = []
* S = {1} (вершини без вхідних ребер)
* Вибираємо вершину 1 з S
* L = [1]
* Видаляємо ребра (1,2) і (1,3)
* Оновлюємо вхідні степені: 2→0, 3→0
* S = {2, 3}

**Крок 2:**

* Вибираємо вершину 2 з S
* L = [1, 2]
* Видаляємо ребро (2,4)
* Оновлюємо вхідний степінь: 4→0
* S = {3, 4}

**Крок 3:**

* Вибираємо вершину 3 з S
* L = [1, 2, 3]
* Видаляємо ребро (3,5)
* Оновлюємо вхідний степінь: 5→1
* S = {4}

**Крок 4:**

* Вибираємо вершину 4 з S
* L = [1, 2, 3, 4]
* Видаляємо ребра (4,5) і (4,6)
* Оновлюємо вхідні степені: 5→0, 6→0
* S = {5, 6}

**Крок 5:**

* Вибираємо вершину 5 з S
* L = [1, 2, 3, 4, 5]
* Вершина 5 не має вихідних ребер
* S = {6}

**Крок 6:**

* Вибираємо вершину 6 з S
* L = [1, 2, 3, 4, 5, 6]
* Видаляємо ребро (6,7)
* Оновлюємо вхідний степінь: 7→0
* S = {7}

**Крок 7:**

* Вибираємо вершину 7 з S
* L = [1, 2, 3, 4, 5, 6, 7]
* S = {} (порожня множина)

**Результат топологічного сортування:** L = [1, 2, 3, 4, 5, 6, 7]

## Практична реалізація

from collections import import defaultdict

def topological\_sort(graph):

# Ініціалізуємо порожній список для зберігання результату сортування

result = []

# Ініціалізуємо словник для зберігання кількості ребер, що входять у кожну вершину

in\_degree = defaultdict(int)

# Ініціалізуємо множину для зберігання вершин без вхідних ребер

without\_incoming\_edges = set()

# Заповнюємо словник in\_degree за множиною without\_incoming\_edges

for vertex in graph:

for neighbor in graph[vertex]:

in\_degree[neighbor] += 1

if vertex not in in\_degree:

without\_incoming\_edges.add(vertex)

# Продовжуємо, доки множина without\_incoming\_edges не порожня

while without\_incoming\_edges:

# Вибираємо вершину без вхідних ребер

vertex = without\_incoming\_edges.pop()

# Додаємо її до результату сортування

result.append(vertex)

# Перебираємо всі вершини, до яких ведуть ребра з поточної вершини

for neighbor in graph[vertex]:

# Зменшуємо кількість вхідних ребер для сусідніх вершин

in\_degree[neighbor] -= 1

# Якщо для вершини не залишилося вхідних ребер, додаємо її до множини without\_incoming\_edges

if in\_degree[neighbor] == 0:

without\_incoming\_edges.add(neighbor)

# Перевіряємо, чи є цикли у графі

if len(result) != len(graph):

# Граф має принаймні один цикл

print("Граф має принаймні один цикл")

return None

else:

# Граф ациклічний, повертаємо результат сортування

return result

# Приклад графа для тестування (варіант 11)

graph = {

1: [2, 3],

2: [4],

3: [5],

4: [5, 6],

5: [],

6: [7],

7: []

}

# Викликаємо функцію для здійснення топологічного сортування

topological\_order = topological\_sort(graph)

# Виводимо результат

if topological\_order:

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

Асимптотична складність алгоритмів

**Алгоритм Кана** та **алгоритм DFS** зазвичай є найефективнішими алгоритмами топологічного сортування.

Асимптотична складність алгоритму може варіюватися залежно від характеристик графа та конкретної реалізації алгоритму.

## Контрольні запитання

· **Що таке орієнтований граф і як він відрізняється від неорієнтованого?** Орієнтований граф - це граф, в якому ребра мають напрямок, тобто кожне ребро веде від однієї вершини до іншої. У неорієнтованому графі ребра не мають напрямку.

· **Що таке топологічне сортування і для чого воно використовується?** Топологічне сортування - це упорядкування вершин ориентованого ациклічного графа таким чином, що для кожного ребра (u,v) вершина u йде перед вершиною v в упорядкованій послідовності.

· **Опишіть алгоритм Кана для топологічного сортування.** Алгоритм Кана працює шляхом поступового видалення вершин без вхідних ребер та оновлення степенів сусідніх вершин до тих пір, поки граф не стане порожнім.

· **Яка асимптотична складність алгоритму Кана?** Асимптотична складність алгоритму Кана становить O(|V| + |E|), де |V| - кількість вершин, |E| - кількість ребер.

· **Чи можна виконати топологічне сортування для графа з циклами?** Ні, топологічне сортування можливе лише для ациклічних орієнтованих графів (ОАГ).

## Висновки

У ході виконання практичної роботи було вивчено основні поняття теорії графів, зокрема орієнтованих ациклічних графів та алгоритмів топологічного сортування. Було розглянуто алгоритм Кана та його практичну реалізацію на мові Python.

Отримано навички:

* Роботи з орієнтованими графами
* Визначення ациклічності графів
* Застосування алгоритму Кана для топологічного сортування
* Аналізу асимптотичної складності алгоритмів
* Практичної реалізації алгоритмів на мові програмування

Топологічне сортування є важливим алгоритмом, який знаходить застосування в багатьох областях, включаючи планування проектів, компіляцію програм, розв'язання задач з обмеженнями та багато інших.