МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ ІМЕНІ МИХАЙЛА
ОСТРОГРАДСЬКОГО

Кафедра комп’ютерної інженерії та електроніки

# ЗВІТ З ПРАКТИЧНОЇ РОБОТИ №5

з навчальної дисципліни
«Алгоритми та методи обчислень»

Тема «Графи. Ациклічні графи»

Студент гр. КІ-24-1, Варакута О.О
Викладач, Сидоренко В. М

Кременчук 2025

Тема: Графи. Ациклічні графи

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

**Завдання**

Задано ациклічний граф: {2, 4, 7, 10, 12}, {(2, 7), (4, 10), (7, 10), (4, 12)}. Побудувати граф і розв’язати задачу топологічного сортування за допомогою алгоритму Кана.

Множина вершин:

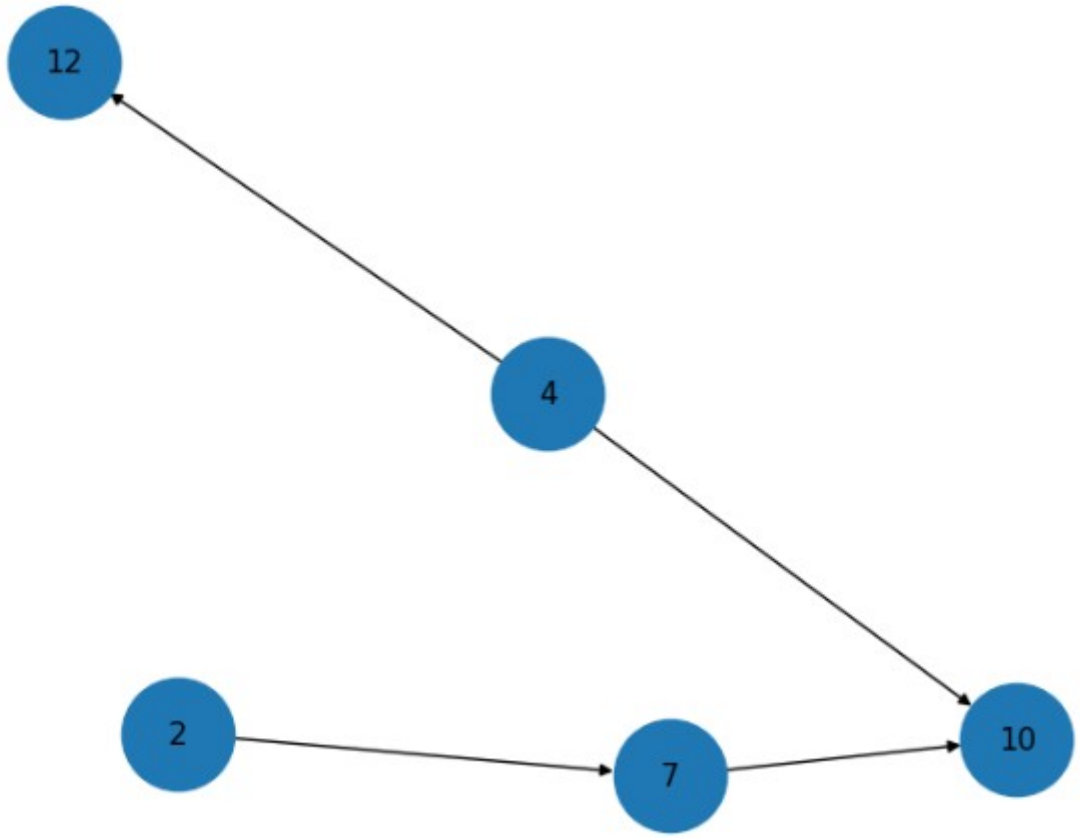
V = {2, 4, 7, 10, 12}

Множина ребер:

E = {(2,7), (4,10), (7,10), (4,12)}

Потрібно: Побудувати орієнтований ациклічний граф. Виконати топологічне сортування графа за допомогою алгоритму Кана. Отримати один з можливих топологічних порядків вершин.

Побудова графа:



Граф є орієнтованим і ациклічним, оскільки не містить жодного циклу.

Розв’язання задачі (алгоритм Кана):

Крок 1. Визначення вхідних степенів вершин

| Вершина | Вхідний степінь |
|---|---|
| 2 | 0 |
| 4 | 0 |
| 7 | 1 (від 2) |
| 10 | 2 (від 4 і 7) |
| 12 | 1 (від 4) |

Крок 2. Формування початкової множини S

Множина S — це вершини без вхідних ребер:

S = {2, 4}

Список результату L спочатку порожній: L = [ ]

Крок 3. Виконання алгоритму

**1.** Вибираємо вершину **2** з множини S

S = {4} L = [2]

Видаляємо ребро (2,7)

Вхідний степінь вершини 7 стає 0 → додаємо 7 до S

S = {4, 7}

**2.** Вибираємо вершину 4

S = {7} L = [2, 4]

Видаляємо ребра (4,10) і (4,12)

Вхідний степінь 12 стає 0 → додаємо 12 до S

S = {7, 12}

**3.** Вибираємо вершину **7**

S = {12} L = [2, 4, 7]

Видаляємо ребро (7,10)

Вхідний степінь 10 стає 0 → додаємо 10 до S

S = {12, 10}

**4.** Вибираємо вершину 12

S = {10} L = [2, 4, 7, 12]

**5.** Вибираємо вершину 10

S = ∅ L = [2, 4, 7, 12, 10]

Крок 4. Перевірка на цикли:

Оскільки всі вершини були додані до списку L, граф не містить циклів.

**Результат:**

Топологічне сортування графа (один з можливих варіантів):

L = [2, 4, 7, 12, 10]

Асимптотична складність алгоритму Кана

Часова складність: O(|V| + |E|)

Просторова складність:O(|V|)

V - кількість вершин, E - кількість ребер графа

Контрольні питання

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

Алгоритм Кана і алгоритм, заснований на пошуку в глибину (DFS), є двома основними методами топологічного сортування орієнтованих ациклічних графів.

Перевагою алгоритму Кана - є те, що він явно виявляє наявність циклів у графі: якщо після завершення алгоритму залишаються необроблені ребра, це означає, що граф містить цикл. Алгоритм працює ітеративно та не використовує рекурсію, що виключає ризик переповнення стеку викликів. Також алгоритм Кана є зручним для задач планування та керування залежностями між об’єктами.

Недоліком алгоритму Кана - є необхідність зберігати та оновлювати вхідні степені вершин, що потребує додаткової пам’яті. Крім того, порядок топологічного сортування може змінюватися залежно від вибору вершини з множини вершин без вхідних ребер.

Алгоритм DFS - простіший у реалізації та не потребує обчислення вхідних степенів вершин. Він добре підходить для розріджених графів. Однак DFS складніше використовувати для явного виявлення циклів, а рекурсивна реалізація може призводити до переповнення стеку при великій глибині графа.

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

Для алгоритму Кана - часова складність у найкращому і найгіршому випадках дорівнює O(|V| + |E|), оскільки кожна вершина і кожне ребро обробляються лише один раз. Просторова складність алгоритму становить O(|V|), оскільки необхідно зберігати вхідні степені вершин та допоміжні структури даних.

Для алгоритму DFS - часова складність у найкращому і найгіршому випадках також дорівнює O(|V| + |E|), оскільки кожна вершина і кожне ребро відвідуються під час обходу графа. Просторова складність становить O(|V|) через використання стеку викликів і масиву відвіданих вершин.

Таким чином, обидва алгоритми мають однакову асимптотичну складність.

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

Алгоритм Кана можна застосовувати до графів з вагами на ребрах, однак ваги при цьому не використовуються і не впливають на результат. Топологічне сортування базується лише на напрямках ребер і залежностях між вершинами.

Аналогічно, алгоритм DFS також ігнорує ваги ребер і використовує лише структуру графа. Отже, наявність ваг не впливає на коректність роботи жодного з алгоритмів, але самі ваги не беруть участі у процесі сортування.

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

Хоча асимптотична складність алгоритмів Кана і DFS однакова, структура графа може впливати на фактичний час виконання.

Для алгоритму Кана швидкість роботи залежить від кількості вершин з нульовим вхідним степенем. У графах із великою кількістю таких вершин алгоритм працює ефективніше. У густих графах з великою кількістю ребер збільшується кількість операцій оновлення вхідних степенів.

Для алгоритму DFS структура графа впливає на глибину рекурсії. У графах з довгими ланцюжками залежностей може зростати глибина стеку викликів,

що негативно впливає на продуктивність. У розріджених графах DFS зазвичай працює швидше.

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

Обидва алгоритми можуть застосовуватися лише до орієнтованих ациклічних графів. Якщо граф містить цикл, топологічне сортування виконати неможливо.

Алгоритм Кана - більш придатний для задач планування, компіляції та керування залежностями, але потребує додаткової пам’яті для зберігання вхідних степенів вершин.

Алгоритм DFS - має обмеження, пов’язані з використанням рекурсії, що може бути проблемою для дуже великих або глибоких графів. Водночас він добре підходить для теоретичного аналізу та обходу графів.

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

Для алгоритму Кана можливими оптимізаціями є використання черги замість множини для зберігання вершин без вхідних ребер, застосування ефективних структур даних та попереднє обчислення вхідних степенів.

Для алгоритму DFS можна використовувати ітеративну реалізацію замість рекурсивної, оптимізувати зберігання списків суміжності та застосовувати маркери станів вершин для швидкого виявлення циклів.