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 структури даних»

Тема «Графи. Ациклічні графи»

Студентка гр. КН-24-1 Бояринцова П. С.
Викладач Сидоренко В. М.

Кременчук 2025

Тема роботи: Графи. Ациклічні графи

1.1 Постановка завдання

Мета роботи: набути практичних навичок розв’язання задач топографічного сортування та оцінювання їх асимптотичної складності.
Завдання: Реалізація алгоритму топологічного сортування та оцінка його складності.

1.2 Розв’язання задачі

Завдання

Задано ациклічний граф: $\{1,2,3,4,5,6\}\{(1,2),(1,3),(2,4),(3,5),(4,5),(4,6)\}$.

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

Розв’язання. Опишемо процес розв’язання покроково.

1. Початковий стан графа:

Вершини: $\{1,2,3,4,5,6\}$

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

2. Початковий стан списку L: Порожній.

Початковий стан множини S: $\{1\}$

3. Виберіть вершину, які не мають вхідних ребер. Початково це вершини з множини S.

4. Виберемо вершину 1 і видалимо її з множини S.

Потім додамо вершину 1 у список L. Список L = [1]

5. Видалимо ребра, що виходять з вершини 1, тобто (1, 2), (1, 3).

6. Перевіримо всі вершини, до яких ведуть ребра з вершини 1. У нашому випадку це вершини 2 та 3. Перевіримо, чи вершини 2 та 3 не має більше вхідних ребер.

Вони не мають. Отже, можемо додати вершину 2 та 3 до множини S.

7. Тепер $S = \{2, 3\}$. Виберемо вершину 2 і повторимо ті самі кроки.

8. Додаємо вершину 2 до списку L і видаляємо ребра, які виходять з неї.

9. Перевіримо всі вершини, до яких ведуть ребра з вершини 2. У нашому випадку це вершина 4. Перевіримо, чи вершина 4 не має більше жодного ребра, що входить. Вона не має ребер, тому можемо додати її до множини S.

10. Тепер $S = \{3, 4\}$. Список $L = [1, 2]$

Виберемо наступну вершину без ребер, що входять, із множини S. Це вершина 3.

11. Повторимо дії для вершини 3 Ребра: $\{(3, 5)\}$

12. Після цього маємо: Множина $S = \{4\}$, список $L = [1, 2, 3]$

13. Виберемо вершину 4. Ребра: $\{(4, 5), (4, 6)\}$

14. Видалимо вершину 4 з множини S і додамо її до списку L. Після цього маємо: Множина $S = \{5, 6\}$, список $L = [1, 2, 3, 4]$

15. Виберемо вершину 5 Ребра: $\{\ \}$

16. Видалимо вершину 5 з множини S і додамо її до списку L. Після цього маємо: Множина $S = \{6\}$, список $L = [1, 2, 3, 4, 5]$

16. Видалимо вершину 6 з множини S і додамо її до списку L.

15. Множина S порожня, тому завершуємо процес.

19. Отже, результуюче топологічне сортування: $L = [1, 2, 3, 4, 5, 6]$.

1.3 Відповіді на контрольні питання

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

Переваги алгоритму Кана:

- Простий у реалізації через роботу з чергою вершин з нульовим входом.
- Легко виявляє цикли: якщо не вдалося обійти всі вершини — граф циклічний.
- Підходить для задач, де важливо відразу знати залежності (наприклад, розклади, компіляції).

Недоліки Кана:

- Потребує додаткових структур (вектор вхідних степенів).
- Не підходить для рекурсивних підходів.

Переваги DFS:

- Рекурсивний підхід дозволяє легко реалізувати обхід в глибину.
- Підходить для роботи зі стеком і більш природно вписується в рекурсивні алгоритми.

Недоліки DFS:

- Складніше обробляти виявлення циклів (потрібен додатковий механізм — кольори вершин).
- Менш очевидний для початківців у порівнянні з Каном.

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

Позначимо:

V — кількість вершин

E — кількість ребер

- **Алгоритм Кана:**

Часова складність: O(V+E) — і в найкращому, і в найгіршому випадках

Пам'ять: O(V+E) — зберігаються степені вершин та список суміжності

- **Алгоритм DFS:**

Часова складність: O(V+E) — і в найкращому, і в найгіршому випадках

Пам'ять: O(V+E) — список суміжності + стек глибини

Обидва алгоритми мають однакову асимптотику, але DFS зазвичай потребує менше оперативної пам'яті через відсутність додаткових структур, таких як черги.

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

Так, можна використовувати алгоритм Кана, якщо ваги не впливають на порядок сортування — тобто коли нам потрібно тільки топологічне впорядкування.

Але ані Кана, ані DFS не враховують ваги — вони просто будують порядок на основі залежностей.

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

У загальному випадку, алгоритм Кана є ефективнішим для густо зв'язаних графів, тоді як DFS може бути кращим варіантом для розріджених графів з меншою кількістю вершин.

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

Обидва алгоритми працюють тільки з ациклічними орієнтованими графами (DAG).

Якщо граф має цикл — топологічне сортування неможливе:

- Кан легко виявляє цикл (залишилися вершини без обробки).
- DFS потребує додаткового коду для виявлення циклу.

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

Алгоритм Кана:

- Зберігати граф у списках суміжності для зменшення витрат пам’яті.
- Використовувати heap/priority queue, якщо потрібно певний порядок сортування (наприклад, лексикографічний).

Алгоритм DFS:

- Ітеративна реалізація (замість рекурсивної) з використанням власного стеку — дозволяє уникнути переповнення стека.
- Застосування кольорової маркировки (0 — не відвідано, 1 — у процесі, 2 — завершено) дозволяє ефективно знаходити цикли.