МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ   
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕЛЕКТРИЧНОЇ ІНЖЕНЕРІЇ   
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА АВТОМАТИЗАЦІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

НАВЧАЛЬНА ДИСЦИПЛІНА  
Алгоритми та методи обчислень

ЗВІТ

З ПРАКТИЧНИХ РОБІТ

Виконав:

студент групи КН-23-1

Сидоренко А.Ю.

Кременчук 2024

**Практична робота № 5**

**Тема.** Графи. Ациклічні графи

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

**Хід роботи**

**Завдання:**

**18. Задано ациклічний граф: (я не розумію де він ациклічний, якщо циклічний, але ок)**

**{1,2,3,4,5,6,7,8}{(1,2),(1,3),(2,4),(3,5),(4,5),(4,6),(5,7),(6,7),(6,8),(7,8)}. 16 Побудувати граф і розв’язати задачу топологічного сортування за допомогою алгоритму DFS.**

**import** matplotlib.pyplot **as** plt*##Взято з лаби*

**import** networkx **as** nx

*# Створюємо граф*

G **=** nx**.**Graph()

*# Додаємо ребра*

edges **=** [(1,2),(1,3),(2,4),(3,5),(4,5),(4,6),(5,7),(6,7),(6,8),(7,8)]*##ПЗ5 варіант 18*

*# Додаємо ребра до графа*

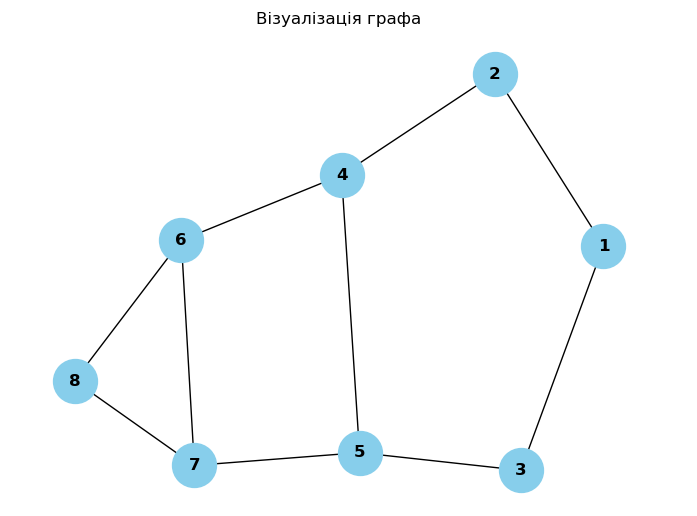
G**.**add\_edges\_from(edges)

*# Візуалізуємо граф*

nx**.**draw(G, with\_labels**=True**, node\_size**=**1000, node\_color**=**"skyblue", font\_size**=**12, font\_weight**=**"bold")

plt**.**title("Візуалізація графа")

plt**.**show()



**%matplotlib** inline

*# Імпорт необхідних бібліотек*

**import** networkx **as** nx

**import** matplotlib.pyplot **as** plt

*# Створення пустого графа*

G **=** nx**.**Graph()

*# Додавання вершин та ребер*

G**.**add\_nodes\_from(['1', '2', '3', '4', '5', '6', '7', '8'])*#Мій граф з ПЗ*

G**.**add\_edges\_from([('1', '2'), ('1', '3'), ('2', '4'), ('3', '5'),

('4', '5'), ('4', '6'), ('5', '7'), ('6', '7'), ('6', '8'), ('7', '8')])

*# Визначення позицій вершин для відображення графа*

pos **=** nx**.**spring\_layout(G)

*# Візуалізація вершин та їх міток*

nx**.**draw\_networkx\_nodes(G, pos, node\_size**=**700)

nx**.**draw\_networkx\_labels(G, pos, font\_size**=**20, font\_family**=**'sans-serif')

*# Візуалізація ребер*

nx**.**draw\_networkx\_edges(G, pos, edgelist**=**G**.**edges())

*# Показ графіка*

plt**.**show()

**def** dfs(graph, start, visited**=None**):*#Взято з лаби*

**if** visited **is** **None**:

visited **=** set() *# Створюємо пустий набір для збереження відвіданих вершин*

visited**.**add(start) *# Додаємо поточну вершину до відвіданих*

*# Перебираємо всі сусідні вершини поточної вершини*

**for** next **in** set(graph**.**adj[start]**.**keys()) **-** visited:

*# Рекурсивно викликаємо функцію dfs для наступної вершини*

dfs(graph, next, visited)

**return** visited

*# Викликаємо функцію dfs з графом G та початковою вершиною 'C'*

dfs(G, '1')

*# Функція dfs\_paths також використовує алгоритм пошуку в глибину (DFS),*

*# але вона знаходить всі можливі шляхи з початкової вершини до цільової вершини.*

*# Параметр path використовується для зберігання поточного шляху.*

**def** dfs\_paths(graph, start, goal, path**=None**):

**if** path **is** **None**:

path **=** [start] *# Починаємо з початкової вершини*

**if** start **==** goal:

**yield** path *# Якщо досягли цільової вершини, повертаємо шлях*

*# Перебираємо всі сусідні вершини поточної вершини, які ще не відвідані*

**for** next **in** set(graph**.**adj[start]**.**keys()) **-** set(path):

*# Рекурсивно викликаємо функцію dfs\_paths для наступної вершини*

**yield from** dfs\_paths(graph, next, goal, path **+** [next])

*# Викликаємо функцію dfs\_paths з графом G, початковою вершиною 'C' та цільовою вершиною 'F'*

list(dfs\_paths(G, '1', '5'))

[['1', '2', '4', '5'],

['1', '2', '4', '6', '7', '5'],

['1', '2', '4', '6', '8', '7', '5'],

['1', '3', '5']]

**Контрольні питання**

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

Алгоритм Кана має більшу ефективність для орієнтованих ациклічних графів, простий в реалізації та може використовуватись в паралельних алгоритмах.

До недоліків я віднесу неможливість роботи з графами в яких є цикли та відсутність ефективності для графів з великою кількістю вершин і ребер.

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

Найкращий і найгірший випадок алгортму Кана за часом і пам’яті= O(V + E), де V - кількість вершин, а E - кількість ребер.

Алгоритм Кана потребує пройти крізь всі вершини і ребра графа, тому і складність для часу однакова.

Найкращий і найгірший випадок DFS за часом= O(V + E), де V - кількість вершин, а E - кількість ребер. ( також завжди має пройти крізь всі вершини та ребра графа)

Складність для пам’яті= O(V) (завжди має пройти крізь всі вершини та ребра графа)

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

Так, алгоритм Кана можна застосовувати до графів з вагами на ребрах, проте він розроблений для роботи з орієнтованими ациклічними графами, тому ваги на ребрах не враховуються при сортуванні.

В той час коли DFS працює з урахуванням ваг на ребрах.

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

Структура графа сильно впливає на роботу і швидкість алгоритму. Наприклад велика кількість вершин і ребер в алгоритмі Кана буде потребувати більше часу на обробку а також збільшить кількість операцій додавання і видалення з черги. Ациклічність збільшує швидкодію алгоритму Кана. Також якщо структура графа має багато гілок, то на обробку піде більше часу

Для DFS важливими параметрами є глибина і ширина графа, бо з їх великим значенням алгоритм буде витрачати більше часу. Як і у кана, цикли також впливають на швидкодію, бо алгоритм може зациклитися і робити повторні відвідування

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

Для Кана важливо: відсутність циклів і проста структура

Для DFS: необхідність уважного контролю за відвідуванням вершин, велика глибина рекурсії, можливість використання для пошуку в глибину і топологічного сортування

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

Для Кана: паралельне виконання, використання оптимізованої структури даних для черги та використання топологічного сортування зі зворотнім реберним списком.

Для DFS: використання ітераційного підходу замість рекурсії, уникнення дублювання вершин та використання оптимізованих структур даних