Міністерство освіти і науки України
Львівський національний університет Івана Франка
Факультет електроніки та комп'ютерних технологій

Звіт
про виконання лабораторної роботи №2
"Сліпий пошук на графах. Особливості реалізації пошуку в глибину у графах."
з курсу "Системи штучного інтелекту"

Виконала
Пильник С. А.
група ФЕІ-42
Перевірив
ас. Іжик О. Б.

Львів 2025

Мета роботи: Вивчення принципів функціонування алгоритмів сліпого пошуку, зокрема пошуку в ширину (Breadth-First Search, BFS), дослідження його особливостей при застосуванні до різних типів графів та реалізація програмного засобу з візуалізацією процесу пошуку. Додатковою метою є аналіз впливу порядку обходу суміжних вершин, структури графа на результативність алгоритму, порівняння з пошуком у глибину (DFS).

Теоретичні відомості:

Граф у теорії графів є фундаментальною структурою, що моделює зв'язки між об'єктами. Формально, граф визначається як упорядкована пара $G = (V, E)$, де V є скінченною множиною вершин (або вузлів), а E — множиною ребер, що з'єднують пари вершин. Залежно від типу з'єднання, розрізняють неорієнтовані та орієнтовані графи. У **неорієнтованому графі** ребра не мають напрямку, тобто з'єднання між вершинами u та v є двостороннім. На противагу, в **орієнтованому графі** (орграфі) ребра є дугами, що мають чітко визначений напрямок, наприклад, від вершини u до вершини v. Таким чином, шлях може існувати лише в одному напрямку. Особливим випадком графа є **дерево** — це зв'язаний ациклічний неорієнтований граф, в якому існує лише один шлях між будь-якими двома вершинами.

Пошук у ширину, відомий як **BFS (Breadth-First Search)**, є одним із ключових алгоритмів сліпого пошуку, призначений для обходу або пошуку на графах. Його основний принцип полягає у послідовному дослідженні графа "шар за шаром". Алгоритм починає з початкової вершини, відвідує всіх її безпосередніх сусідів (перший шар), потім усіх сусідів цих сусідів (другий шар) і так далі, доки не буде знайдено цільову вершину. Цей підхід забезпечується використанням спеціальної структури даних — **черги (Queue)**. На відміну від пошуку в глибину (DFS), який використовує стек і занурюється якнайглибше в одну гілку, BFS додає всі відкриті, але ще не відвідані вершини до черги. Завдяки принципу роботи черги **FIFO (First-In, First-Out)**, алгоритм гарантовано обробляє вершини в порядку, в якому вони були знайдені, що забезпечує рівномірний обхід.

BFS має кілька важливих властивостей. По-перше, він є **повним** алгоритмом, що означає, що він гарантовано знайде шлях до цільової

вершини, якщо такий шлях існує. По-друге, і це його найважливіша перевага, у незважених графах BFS завжди знаходить **найкоротший шлях** між початковою та цільовою вершинами. Це пов'язано з тим, що алгоритм розглядає всі шляхи з довжиною k перед тим, як перейти до шляхів довжиною $k + 1$. Часова складність BFS становить $O(|V| + |E|)$, де $|V|$ — кількість вершин, а $|E|$ — кількість ребер, оскільки в найгіршому випадку алгоритм відвідає кожну вершину і кожне ребро графа один раз. Проте, важливо зазначити, що BFS може бути **пам'яткоємним**, оскільки в черзі може одночасно зберігатися значна кількість вершин, особливо на широких графах з високим ступенем розгалуження.

Хід роботи:
У процесі виконання лабораторної роботи було створено програму мовою Python із використанням бібліотек *networkx*, *matplotlib* та *tkinter*. Програма реалізує генерацію випадкового розгалуженого графа порядку не менше 30 та розміру не менше 30–40 ребер, забезпечує його графічне відображення та інтерактивне керування параметрами пошуку.

Основні функціональні можливості програми включають: генерацію графа із заданою кількістю вершин і ребер, вибір типу графа (звичайний чи орієнтований), вибір початкової та цільової вершини, визначення порядку обходу суміжних вершин (заданого, зростаючого, спадного або випадкового), а також виконання пошуку в глибину з покроковою візуалізацією процесу. Передбачено як покрокове виконання, так і автоматичний режим з регульованою затримкою між кроками. Результати пошуку — знайдений шлях, його довжина та кількість розкритих вершин — відображаються у вікні інтерфейсу.

Інструкція з використання програми передбачає наступні дії. Користувач задає кількість вершин та ребер, тип графа та параметри обходу. Після натискання кнопки «Generate Graph» створюється випадковий граф. Далі можна обрати початкову та цільову вершини. Пошук виконується натисканням кнопки «Step» (для покрокового режиму) або «Run (Auto)» (для автоматичного режиму). Після завершення роботи алгоритму у полі результатів з’являється знайдений шлях та статистичні дані.

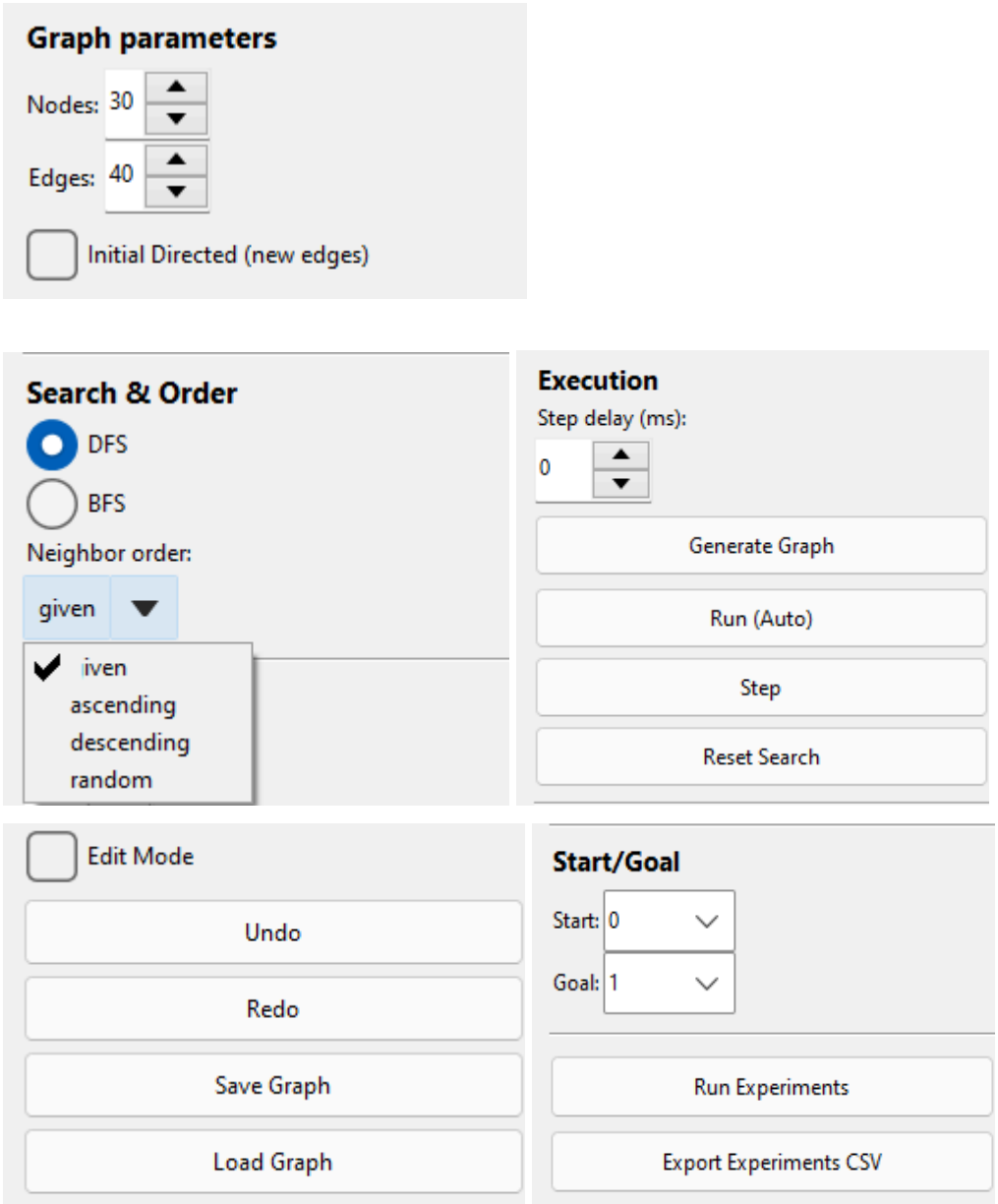


рис.1. Панель керування

Опис кожного елемента панелі керування:

- Graph parameters (Параметри графа)
- Nodes (Вершини): Спінбокс, що дозволяє користувачу обрати кількість вершин для нового графа. Дозволений діапазон від 5 до 300.
- Edges (Ребра): Спінбокс, що дозволяє користувачу обрати кількість ребер для нового графа. Дозволений діапазон від 4 до 2000.

- Initial Directed (new edges) (Початкові орієнтовані (нові ребра)): Чекбокс, який, якщо відмічений, робить новостворені ребра орієнтованими.

Search & Order (Пошук та Порядок)

- DFS: Радіокнопка для вибору алгоритму пошуку в глибину (Depth-First Search).
- BFS: Радіокнопка для вибору алгоритму пошуку в ширину (Breadth-First Search).
- Neighbor order (**Порядок сусідів**): Випадаючий список, який визначає порядок обходу сусідів при пошуку. Доступні опції: *given* (Обхід сусідів у тому порядку, в якому вони зберігаються у структурі даних графа. Тобто алгоритм не змінює порядок, у якому вершини додаються в чергу.), *ascending* (Сусіди вершини впорядковуються у **зростаючому порядку їхніх номерів**.), *descending* (Сусіди вершини впорядковуються у **спадному порядку їхніх номерів**.) та *random* (Сусіди вершини випадково перемішуються (рандомізуються) перед тим, як додати їх у чергу.).

Execution (Виконання)

- Step delay (ms) (Затримка кроку (мс)): Спінбокс для встановлення затримки між кожним кроком виконання алгоритму. Вказується в мілісекундах, що впливає на швидкість анімації. Дозволений діапазон від 10 до 3000 мс.
- Generate Graph (Створити граф): Кнопка, яка генерує новий граф згідно з встановленими параметрами.
- Run (Auto) (Запустити (автоматично)): Кнопка, яка запускає автоматичне виконання обраного алгоритму пошуку з візуалізацією, використовуючи встановлену затримку.
- Step (Крок): Кнопка, яка виконує один крок алгоритму пошуку, візуалізуючи його результат.
- Reset Search (Скинути пошук): Кнопка, яка скидає стан пошуку, дозволяючи почати новий пошук з тієї ж конфігурації графа.

Edit Mode (Режим редагування)

- Edit Mode (Режим редагування): Чекбокс, який активує функції редагування графа, такі як додавання, видалення та переміщення вершин і ребер.
- Undo (Скасувати): Кнопка для скасування останньої дії.
- Redo (Повторити): Кнопка для повторення останньої скасованої дії.
- Save Graph (Зберегти граф): Кнопка для збереження поточного графа у файл.
- Load Graph (Завантажити граф): Кнопка для завантаження графа з файлу.

Start/Goal (Початок/Мета)

- Start (Початок): Випадаючий список для вибору початкової вершини для пошуку.
- Goal (Мета): Випадаючий список для вибору цільової вершини для пошуку.

Results (Результати)

- Run Experiments (Запустити експерименти): Кнопка, яка виконує серію автоматичних експериментів на декількох типах графів (дерево, неорієнтований, орієнтований), збираючи статистику.
- Export Experiments CSV (Експортувати експерименти в CSV): Кнопка для експорту результатів експериментів у файл формату CSV.
- Текстове поле: Відображає лог виконання програми, включаючи статус пошуку, відвідані вершини та інші повідомлення.

Режим редагування (чекбокс — Edit Mode) дозволяє виконувати такі дії:

- ЛКМ по канві — додається нова вершина.
- Подвійний клік по вершині, потім одинарний по іншій вершині — створюється ребро.
- Одинарний клік по ребру — ребро стає дугою.
- Одинарний клік по дузі — змінює напрям.
- Затискання і перетягування — зміна положення вершини і зв'язаних з нею ребер/дуг.

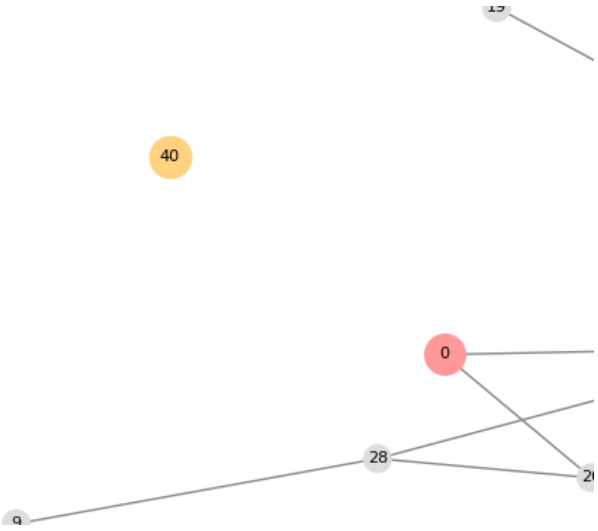


рис.2. Створення і виділення вершини

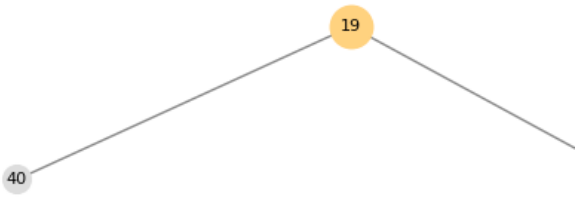


рис.3. Зєднання створеної вершини з виділеною вже існуючою вершиною.

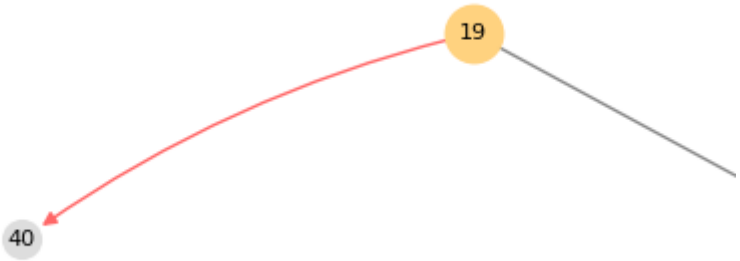


рис.4. Зміна ребра на дугу

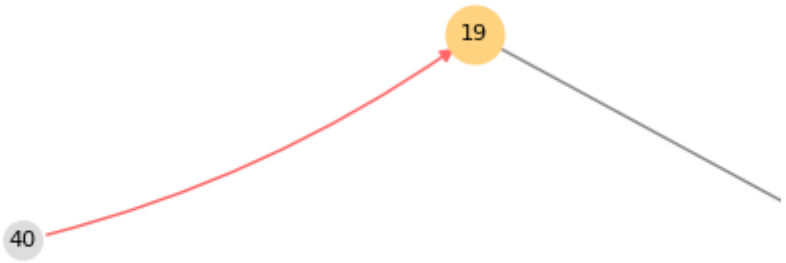


рис.5. Зміна напрямку дуги

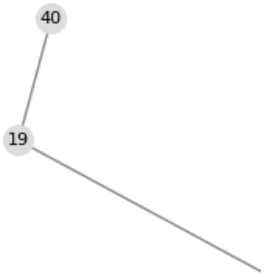
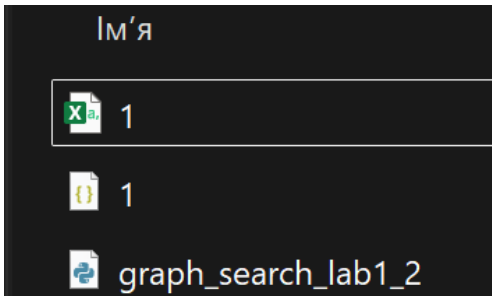


рис.6. Зміна положення вершини і суміжного ребра.

```
Saved C:/projects/sem_7/AIS/lab_1_2/1.json
Loaded
C:/projects/sem_7/AIS/lab_1_2/1.json
```

рис.7. Експорт та імпорт графу з .json файлу.



	A	B	C	D	E	F
1	variant	algo	order	found	pathlen	opened
2	Tree	DFS	given	TRUE	7	20
3	Tree	DFS	ascending	TRUE	7	29
4	Tree	DFS	descending	TRUE	7	8
5	Tree	DFS	random	TRUE	7	11
6	Tree	BFS	given	TRUE	7	30
7	Tree	BFS	ascending	TRUE	7	30
8	Tree	BFS	descending	TRUE	7	30
9	Tree	BFS	random	TRUE	7	30
10	Undirected	DFS	given	TRUE	7	29

рис.8-9. Збережиниі граф 1.json та таблиця з результатами експеременту 1.csv

Дослідження

Було проведено низку експериментів для дослідження поведінки алгоритмів сліпого пошуку в різних умовах. Програма дозволяла змінювати ключові параметри, такі як тип графа (дерево, неорієнтований, орієнтований), алгоритм пошуку (BFS або DFS), а також порядок обходу суміжних вершин (*given, ascending, descending, random*). Для об'єктивного порівняння, початкова та кінцева вершини були зафіксовані. Результати експериментів представлені в таб. 1.

variant	algo	order	found	pathlen	opened
Tree	DFS	given	TRUE	7	20
Tree	DFS	ascending	TRUE	7	29
Tree	DFS	descending	TRUE	7	8
Tree	DFS	random	TRUE	7	11
Tree	BFS	given	TRUE	7	30
Tree	BFS	ascending	TRUE	7	30
Tree	BFS	descending	TRUE	7	30
Tree	BFS	random	TRUE	7	30
Undirected	DFS	given	TRUE	7	29
Undirected	DFS	ascending	TRUE	15	16
Undirected	DFS	descending	TRUE	17	23
Undirected	DFS	random	TRUE	17	23
Undirected	BFS	given	TRUE	7	30

Undirected	BFS	ascending	TRUE	7	30
Undirected	BFS	descending	TRUE	7	30
Undirected	BFS	random	TRUE	7	30
Directed	DFS	given	TRUE	6	6
Directed	DFS	ascending	TRUE	11	13
Directed	DFS	descending	TRUE	8	16
Directed	DFS	random	TRUE	6	6
Directed	BFS	given	TRUE	6	26
Directed	BFS	ascending	TRUE	6	26
Directed	BFS	descending	TRUE	6	27
Directed	BFS	random	TRUE	6	26

таб. 1. Зведені результати експериментів.

Вплив порядку обходу суміжних вершин

Пошук у ширину (BFS): Результати для BFS демонструють, що зміна порядку обходу суміжних вершин не впливає на довжину знайденого шляху. У всіх випадках для одного й того ж графа BFS знаходив шлях однакової довжини: 7 для дерева та неорієнтованого графа і 6 для орієнтованого. Це підтверджує ключову перевагу BFS — **гарантію знаходження найкоротшого шляху** у незваженому графі. Кількість розкритих вершин також залишалася майже незмінною (наприклад, для Directed графа вона коливалася від 26 до 27), що свідчить про стабільність алгоритму.

Пошук у глибину (DFS): Для DFS вплив порядку обходу є критичним. З таблиці видно, що для неорієнтованого графа *pathlen* змінюється від 7 (для *given*) до 17 (для *descending і random*). Кількість відкритих вершин також значно варіюється — від 16 до 29. Це яскраво ілюструє, що **DFS не**

гарантує знаходження найкоротшого шляху, оскільки він сліпо занурюється в одну з гілок, і якщо ця гілка є довгою, то знайдений шлях буде неоптимальним. Те саме спостерігається і для орієнтованого графа, де *pathlen* коливається від 6 до 11, а *opened* від 6 до 16.

Вплив типу графа та порівняння BFS і DFS

На дереві: DFS виявився більш ефективним з точки зору кількості розкритих вершин, ніж BFS. Наприклад, для порядку *descending* DFS розкрив лише 8 вершин, що значно менше, ніж 30 вершин, які розкрив BFS. Це пояснюється тим, що на дереві немає циклів, і якщо цільова вершина знаходиться на "правильній" гілці, DFS швидко її знаходить, не відвідуючи зайвих вузлів.

На неорієнтованому графі: BFS, як і очікувалося, стабільно знаходив найкоротший шлях довжиною 7, розкриваючи 30 вершин. DFS, залежно від порядку обходу, знаходив шлях від 7 до 17 вузлів, розкриваючи від 16 до 29 вершин. У випадку *given* порядку, DFS знайшов шлях такої ж довжини (7) як і BFS, з аналогічною кількістю розкритих вершин (29 vs 30). Це вказує на те, що DFS може бути ефективним, якщо порядок обходу сприяє пошуку.

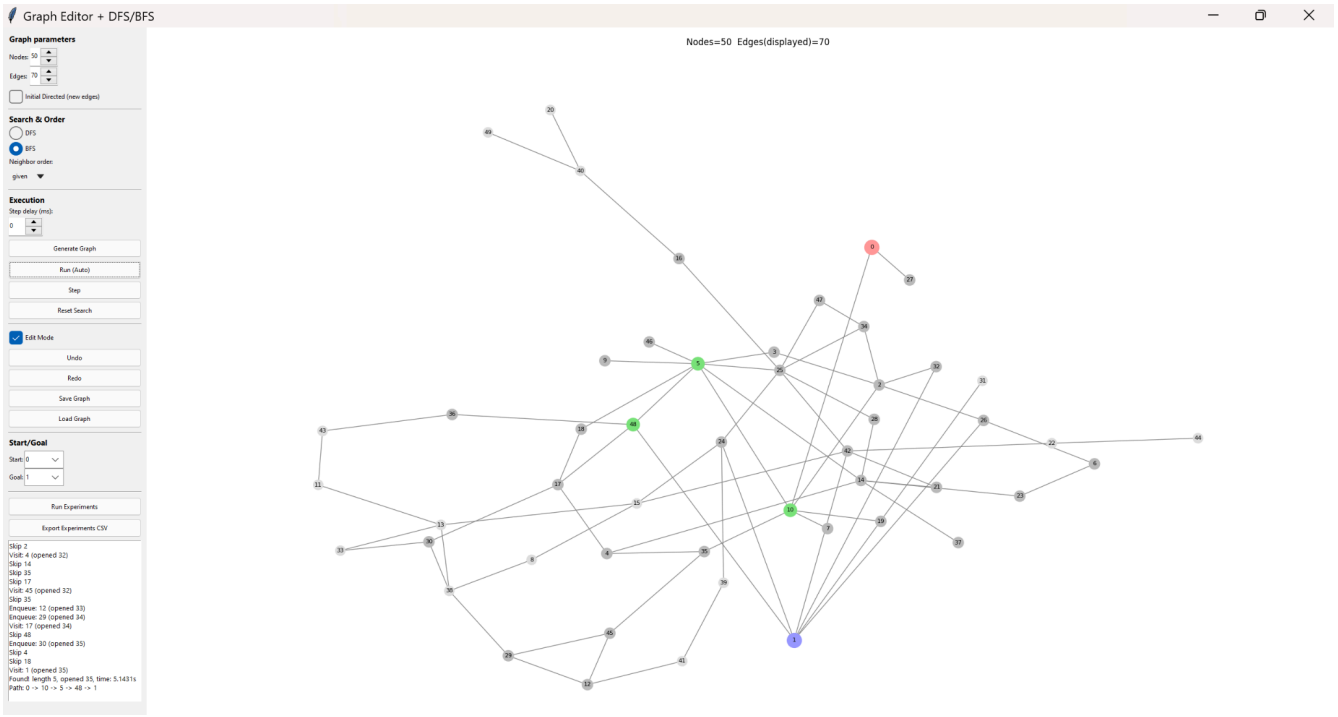


рис. 10. Візуалізація шляху, знайденого BFS, на неорієнтованому графі.

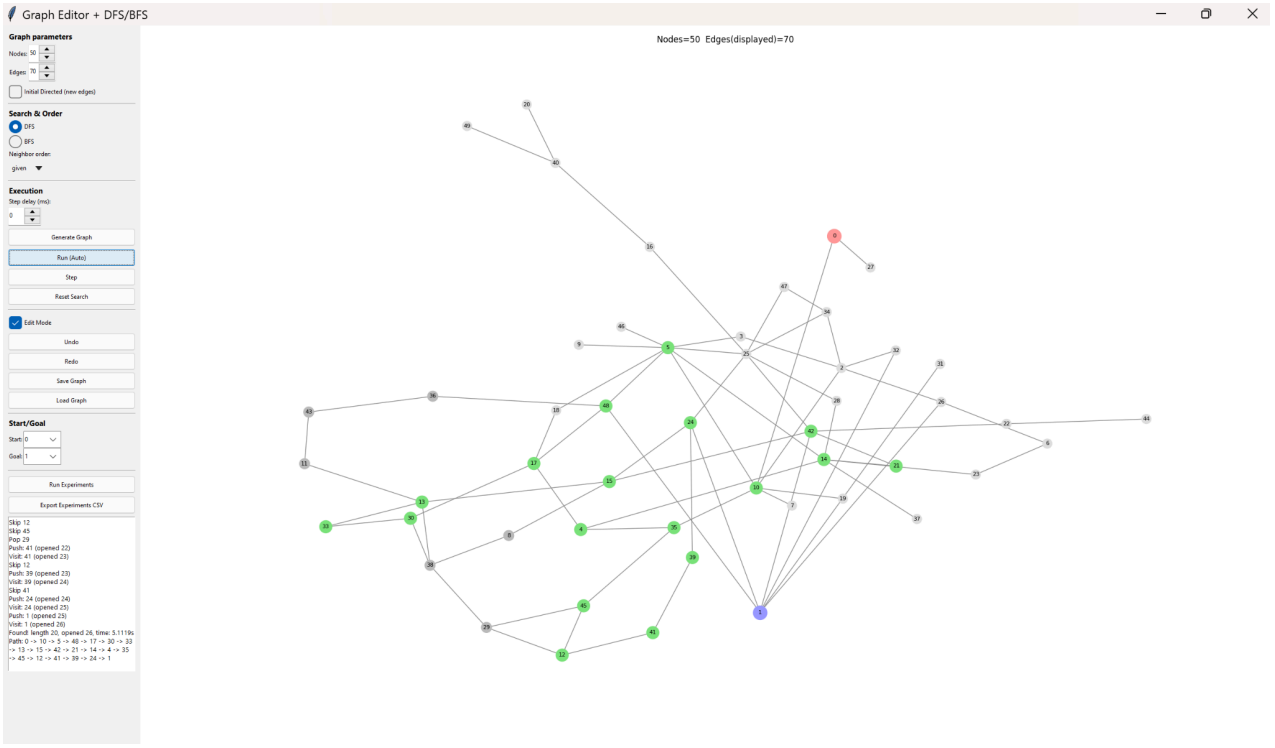


рис. 11. Візуалізація шляху, знайденого DFS, на неорієнтованому графі.

На орієнтованому графі: Ситуація аналогічна. BFS завжди знаходив найкоротший шлях довжиною 6. DFS, знову ж таки, демонстрував залежність від порядку, знаходячи шляхи від 6 до 11 вузлів. Це підкреслює, що наявність однонаправлених дуг не змінює фундаментальних властивостей алгоритмів.

Дзеркальна заміна початкової і цільової вершин призводить до непередбачуваних результатів, особливо на орієнтованих графах. Шлях, який існує від A до B, може не існувати від B до A, що може призвести до того, що пошук не знайде рішення. На неорієнтованих графах шлях завжди двонаправлений, але кількість розкритих вершин може змінюватися через різну структуру графа, що "розгортається" від нової початкової вершини.

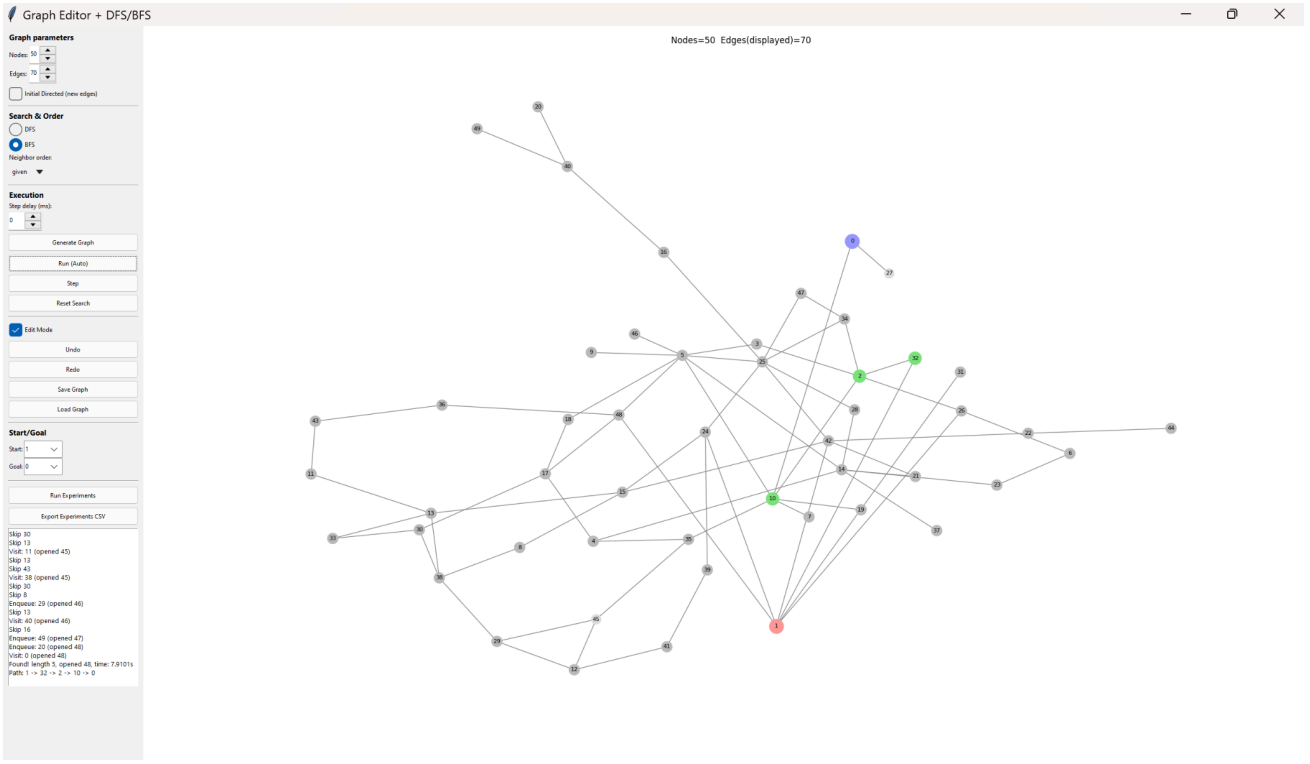


рис. 12. Візуалізація шляху, знайденого DFS, на неорієнтованому графі з дзеркальною заміною вершин (1→0).

Головна відмінність між BFS і DFS полягає у використанні різних структур даних для керування процесом обходу:

BFS використовує **чергу (Queue)**. Завдяки принципу **FIFO (First-In, First-Out)**, він обробляє вершини в порядку їхнього додавання. Це змушує алгоритм досліджувати всі вершини на поточному "шарі" від початкової вершини, перш ніж перейти до наступного шару. Це і є причиною його здатності знаходити найкоротший шлях.

DFS використовує **стек (Stack)**. Завдяки принципу **LIFO (Last-In, First-Out)**, він завжди заглиблюється в першу знайдену гілку до кінця. Це робить його менш передбачуваним щодо довжини шляху, але більш пам'ятко-ефективним у глибоких графах.

Це фундаментальне організаційне розходження є причиною всіх відмінностей у результатах: стабільності BFS проти варіативності DFS, гарантії найкоротшого шляху BFS проти ризику знаходження неоптимального шляху DFS.

Висновки:

У результаті виконання лабораторної роботи було успішно реалізовано програмний комплекс для візуалізації та дослідження алгоритмів сліпого пошуку на графах.

Пошук у ширину (BFS) є надійним і передбачуваним алгоритмом для знаходження найкоротшого шляху в незважених графах. Його головна перевага — це гарантія оптимальності знайденого шляху. Експерименти підтвердили, що довжина шляху, знайденого BFS, залишається незмінною незалежно від порядку обходу суміжних вершин (*given, ascending, descending, random*). Водночас, кількість розкритих вершин для BFS є відносно стабільною, але може бути значною на широких графах.

Пошук у глибину (DFS), навпаки, є менш передбачуваним з точки зору оптимальності шляху. Дослідження показали, що довжина шляху та кількість розкритих вершин для DFS значно залежать від порядку обходу суміжних вершин. Це підтверджує, що DFS не гарантує знаходження найкоротшого шляху, оскільки його стратегія полягає в заглибленні в одну гілку. Проте, DFS може бути більш ефективним з точки зору пам'яті та швидкості, якщо цільова вершина знаходиться на першій же "щасливій" гілці, що є особливо актуальним для глибоких, але вузьких графів, таких як дерева.

Посилання на GitHub репозиторій з кодом проєкту:

https://github.com/sofiapylnyk/ais_2025/blob/main/lab_1_2/graph_search_lab1_2.py