результат роботи програми та Excel аналогічно, отже програма працює коректно та може облегшити розв’язання такого роду задач. [3]

Список літератури

1. *Волонтир Л.О., Зелінська О.В., Потапова Н.А., Чіков І.А. Чисельні методи. Навчальний посібник. Вінниця: ВНАУ. 2020. 322 с.*
2. *Jaan K., “Numerical methods in engineering with Matlab”, 2005. 435 с.*
3. *JavaFX Official Documentation. URL: https://fxdocs.github.io/docs/html5/*

УДК 004.6

Лаптєва М. А.., студентка 1 курсу
спеціальності 122 «Комп’ютерні науки»
Науковий керівник:
Гончар В. М., асистент
кафедри інформаційних технологій

АЛГОРИТМИ ЗНАХОДЖЕННЯ КІЛЬКОСТІ ШЛЯХІВ МІЖ ВЕРШИНАМИ В ГРАФАХ

Донецький національний університет імені Василя Стуса, м. Вінниця

Існує декілька алгоритмів для знаходження кількості шляхів між вершинами графа. Вибір алгоритму залежить від конкретних характеристик графа і бажаної ефективності обчислень. Нижче наведено три найпоширеніші алгоритми, які використовуються для цієї мети:

1. **Пошук в глибину (DFS):**

Пошук у глибину (DFS) - це алгоритм, який використовується для дослідження або обходу графа або деревовидної структури даних. Він починається з заданої вихідної вершини і систематично досліджує якомога далі вздовж кожної гілки, перш ніж повернутися назад. DFS можна використовувати для різних завдань, таких як пошук зв'язаних компонентів, виявлення циклів, топологічне сортування і, як згадувалося раніше, знаходження кількості шляхів між вершинами.

DFS використовує рекурсивний підхід для обходу графа. Алгоритм підтримує стек для відстеження відвіданих вершин та їхніх сусідів, які ще не були досліджені. При зворотному обході алгоритм витягує останню вершину зі стеку і продовжує досліджувати її сусідів, що залишилися невідвіданими.

DFS часто реалізується за допомогою рекурсивної функції, що робить код лаконічним та інтуїтивно зрозумілим.

Під час виконання DFS алгоритм може виконувати різні дії на різних етапах, такі як ініціалізація обходу, обробка вершини або завершення обходу. Модифікуючи реалізацію, ви можете адаптувати DFS до конкретних вимог.

Важливо зазначити, що на графі з циклами DFS може застрягти у нескінченному циклі, якщо немає механізму для виявлення та обробки циклів. Щоб уникнути цього, ви можете використовувати додаткові методи обліку, такі як позначення вершин як "у процесі" під час обходу та перевірка наявності зворотних ребер [1].

DFS має часову складність O(V + E), де V - кількість вершин, а E - кількість ребер у графі. Просторова складність дорівнює O(V), оскільки алгоритм вимагає пам'яті для зберігання відвіданих вершин у найгіршому випадку.

DFS є фундаментальним графовим алгоритмом і широко використовується в різних додатках. Його простота і універсальність роблять його цінним інструментом для розв'язання задач, пов'язаних з графами.

2. **Пошук в ширину (BFS):**

Пошук в ширину (BFS) - це ще один алгоритм обходу графа, який досліджує граф у напрямку вшир, тобто відвідує всі вершини на одному рівні, перш ніж перейти на наступний рівень. BFS зазвичай використовується для пошуку найкоротшого шляху між двома вершинами у незваженому графі або для дослідження всіх вершин у зв'язному компоненті.

BFS досліджує граф рівень за рівнем, переконуючись, що всі вершини на певному рівні відвідані, перш ніж перейти на наступний рівень. Таким чином, він гарантує, що буде знайдено найкоротший шлях від початкової вершини до будь-якої іншої вершини, якщо припустити, що граф незважений.

BFS можна реалізувати за допомогою структури даних у вигляді черги, яка працює за принципом First-In-First-Out (FIFO). У черзі зберігаються вершини, які потрібно відвідати, і алгоритм обробляє їх по черзі, одночасно опитуючи їхніх невідвіданих сусідів.

BFS гарантує, що всі вершини, доступні з вихідної вершини, будуть відвідані, і знаходить найкоротший шлях до кожної доступної вершини. Якщо вам потрібно знайти найкоротший шлях між двома конкретними вершинами, ви можете доповнити алгоритм BFS, щоб зберігати інформацію про батьків для кожної вершини, що дозволить вам реконструювати шлях від вершини призначення назад до джерела.

BFS має часову складність O(V + E), де V - кількість вершин, а E - кількість ребер у графі. Просторова складність також дорівнює O(V), оскільки алгоритм потребує пам'яті для зберігання відвіданих вершин та черги.

BFS особливо корисний для розв'язання задач, де потрібен найкоротший шлях або дослідження вшир. Його здатність знаходити найкоротший шлях у незважених графах робить його цінним у багатьох додатках, таких як планування маршрутів, аналіз мереж та головоломки на основі графів.

3. **Динамічне програмування:**

Динамічне програмування (ДП) - це метод, який використовується для вирішення складних проблем шляхом розбиття їх на підпроблеми, що перетинаються, і вирішення кожної підпроблеми лише один раз, зберігаючи результати для подальшого використання. Це особливо корисно, коли проблема має оптимальну підструктуру та підпроблеми, що перетинаються.

Основна ідея динамічного програмування полягає у вирішенні проблеми шляхом розв'язання менших підпроблем та об'єднання їхніх рішень. Метод часто передбачає побудову таблиці або масиву для зберігання проміжних результатів, що дозволяє ефективно шукати і повторно використовувати раніше обчислені рішення [2].

Динамічне програмування зазвичай використовується для вирішення оптимізаційних задач, таких як пошук максимального або мінімального значення або підрахунок кількості способів досягнення певної мети. Воно може бути застосоване до різних областей, включаючи комбінаторні задачі, вирівнювання послідовностей, графові алгоритми, розподіл ресурсів тощо.

Запам'ятовуючи проміжні результати, динамічне програмування дозволяє уникнути надлишкових обчислень, що призводить до значного підвищення швидкості порівняно з наївними підходами, які багаторазово розв'язують одні й ті ж підзадачі. Однак важливо зазначити, що динамічне програмування можна застосовувати не до всіх задач, оскільки воно залежить від наявності оптимальної підструктури та підзадач, що перетинаються.

Класичним прикладом динамічного програмування є послідовність Фібоначчі, де кожне число є сумою двох попередніх чисел. Використовуючи динамічне програмування для зберігання та повторного використання раніше обчислених чисел Фібоначчі, часову складність можна зменшити з експоненціальної до лінійної.

Загалом, динамічне програмування є потужною технікою, яка дозволяє ефективно та елегантно розв'язувати складні задачі, використовуючи повторне використання результатів обчислень. Воно має широке застосування в різних галузях і продовжує залишатися важливим інструментом у вирішенні алгоритмічних проблем.

Варто зазначити, що складність цих алгоритмів залежить від характеристик графа, таких як його розмір та зв'язність. Деякі графи можуть мати експоненціальну кількість шляхів, що робить обчислення неможливими для великих графів. У таких випадках може знадобитися використання методів апроксимації або спеціалізованих алгоритмів, пристосованих до конкретних структур графів [3].

Список літератури

1. *"Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein*
2. *"Algorithms" by Robert Sedgewick and Kevin Wayne*
3. *"The Algorithm Design Manual" by Steven S. Skiena*