переміщення, а також покращити загальний досвід користувачів. Застосування алгоритму Дейкстри може сприяти оптимізації різних процесів, що пов'язані з маршрутизацією, транспортом та доставкою, що робить його незамінним інструментом у сучасному світі.

Важливо також зазначити, що розвиток технологій та збільшення обчислювальної потужності дозволяють застосовувати більш складні алгоритми, які можуть враховувати додаткові фактори та обмеження, такі як трафік, густота населення, обмеження швидкості тощо. Однак, алгоритм Дейкстри залишається основою та одним з найпоширеніших методів для пошуку оптимального маршруту.

У підсумку, застосування алгоритму Дейкстри для пошуку оптимального маршруту має широкі можливості у різних галузях, що потребують ефективної маршрутизації. Його використання дозволяє забезпечити швидкий та ефективний пошук найкоротшого шляху, що впливає на покращення процесів транспортування, маршрутизації даних, навігації та інших систем, які вимагають оптимальних маршрутів. Алгоритм Дейкстри залишається одним з найефективніших та популярних методів для вирішення цих завдань [3].

Список літератури.

1. Labeling Algorithm for Shortest Paths on Road Networks. / [Abraham I., Delling D., Goldberg A., Werneck R.]. - Philadelphia.- Symposium on Experimental Algorithms, 2011. — pp. 230-241.

2. Wikipedia. "Dijkstra's algorithm." Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

3. GeeksforGeeks. "Dijkstra's Algorithm for Finding Shortest Path." [Online]. Available: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

УДК 004.6

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

АЛГОРИТМИ ЗНАХОДЖЕННЯ НАЙБІЛЬШОГО ПІДГРАФА З НЕЗАЛЕЖНОЮ МНОЖИНОЮ РЕБЕР

Донецький національний університет імені Василя Стуса, м. Вінниця

У наш час графи є потужним інструментом для моделювання різноманітних систем. Знаходження найбільшого підграфа з незалежною множиною ребер має багато прикладних застосувань, наприклад, в оптимізації мережі транспорту або при проектуванні системи зв'язку.

Далі розглянемо деякі алгоритми для знаходження найбільшого підграфа з незалежною множиною ребер.

Алгоритм Брона-Кербоша — це рекурсивний підхід знаходження найбільшого підграфа з незалежною множиною ребер у неорієнтованому графі.

Можна його описати таким чином:

1. Починаємо з порожньої множини вершин.
2. Додаємо по одній вершині на кожному кроці, перевіряючи, чи утворюється незалежна множина. Якщо так, то алгоритм викликається рекурсивно для підмножини вершин, яка включає додану вершину та всі вершини, з якими вона спільна.
3. Повертаємось до попереднього рівня рекурсії, коли більше немає вершин для додавання.

Для того, щоб використати даний алгоритм для знаходження найбільшого підграфа з незалежною множиною ребер вихідного графа, необхідно застосувати його до доповнення графа та знайти найбільшу знайдену незалежну множину вершин. Потім, вихідний граф можна перетворити, видаливши всі вершини, які належать до цієї незалежної множини, та всі зв'язані з ними ребра. Залишений граф буде максимальним підграфом з незалежною множиною ребер.

Часова складність становить $O(3^{n/3})$, де n - кількість вершин у графі. Алгоритм Брона-Кербоша може бути ефективним для графів з невеликою кількістю вершин та ребер. Однак, зі збільшенням розміру графа час роботи алгоритму значно зростає [1].

Жадібний алгоритм – прямолінійний евристичний алгоритм, який приймає найкраще рішення, виходячи з наявних на кожному етапі даних.

Можна його описати таким чином:

1. Починаємо з порожнього підграфа
2. Для кожної вершини в графі перевіряємо умову: якщо вершина не є суміжною з жодною вершиною підграфа, додаємо її до підграфа.
3. Повторюємо крок 2, доки до підграфа більше не можна буде додати вершини.
4. Підграф, знайдений на кроці 3, є найбільшим підграфом із незалежним набором ребер.

Часова складність зазвичай становить $O(n \log n)$ або $O(n^2)$, де n - кількість вершин у графі.

Основна перевага жадібного алгоритму – простота та швидкість виконання, однак він має деякі недоліки. Жадібний алгоритм приймає локальні рішення на кожному кроці, не враховуючи глобальний оптимум (деякі ребра можуть бути пропущені в першій ітерації через вибір неправильного ребра на початку). В результаті він може не знайти оптимального рішення проблеми. Також продуктивність жадібного алгоритму може бути чутливою до вхідних даних. Навіть невелика їхня зміна може призвести до того, що алгоритм видасть інший результат.

Незважаючи на ці недоліки, жадібний алгоритм все ще є одним з найбільш популярних підходів для знаходження найбільшого підграфа з незалежною множиною ребер, оскільки він дуже легко реалізовується [2].

Алгоритм локального пошуку – алгоритм, що ітеративно вдосконалює підграф, змінюючи місцями вершини в підграфі та поза ним.

Можна його описати таким чином:

1. Починаємо з довільного підграфа.
2. Повторюємо наступні кроки, доки можливо. Для кожної вершини підграфа виконаємо наступні дії:
   - Якщо вершину можна видалити з підграфа без зменшення розміру незалежної множини, то видаляємо її.
   - Якщо будь-які вершини були видалені на попередньому кроці, додаємо нову вершину до підграфа, яка не є суміжною з жодною з вершин, які були видалені.
3. Остаточним результатом є найбільший підграф із незалежною множиною ребер.

Алгоритм перевіряє, чи нове рішення, отримане шляхом внесення невеликих змін, краще за поточне. Якщо так, то поточне рішення замінюється новим. В іншому випадку алгоритм зберігає наявне. Алгоритм завершується, якщо виконується критерій зупинки (це може бути максимальна кількість ітерацій, максимальний проміжок часу або певний рівень покращення) [3].

У загальному випадку, часова складність алгоритму локального пошуку зазвичай є поліноміальною, тобто $O(n^k)$, де n - розмір вхідних даних, а k - ступінь полінома. Проте, у конкретних випадках часова складність може бути і експоненціальною (кількість можливих шляхів зростає факторіально з кількістю вершин у графі).

Даний алгоритм складніший за жадібний, але часто ефективніший у пошуку оптимального рішення [4].

Отже, алгоритм Брона-Кербоша та жадібний алгоритм досить прості у реалізації, натомість алгоритм локального пошуку є складнішим, проте більш ефективним у пошуку оптимального рішення для знаходження найбільшого підграфа з незалежною множиною ребер. Вагоме значення у цьому питанні має часова складність, що визначається кількістю елементарних операцій, які потрібно виконати для реалізації алгоритму [5].

З огляду на все викладене вище, кожен з цих алгоритмів має свої переваги та недоліки та може бути використаний у різних випадках в залежності від конкретної задачі, властивостей графу та вимог до точності результату.

Список літератури:

1. *Bron–Kerbosch algorithm. URL: https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm*
2. *Greedy algorithm. URL: https://en.wikipedia.org/wiki/Greedy_algorithm*
3. *Local search (optimization). URL: https://en.wikipedia.org/wiki/Local_search_(optimization) (дата звернення: 11.05.2023)*
4. *Local search. URL: https://docs.optaplanner.org/6.0.0.CR5/optaplanner-docs/html/localSearch.html (дата звернення: 11.05.2023)*
5. *Big O: Складність алгоритмів. URL: https://www.the-code.com.ua/skladnist-alghoritmiv/ (дата звернення: 12.05.2023)*