МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Кафедра комп'ютерної інженерії та електроніки

ЗВІТ З ПРАКТИЧНИХ РОБІТ

з навчальної дисципліни

«Алгоритми та методи обчислень»

Тема «Жадібні алгоритми. Наближене розв'язання екстремальних задач»

Студент гр. КІ-23-1 ПІБ Кобець О. О.

Кременчук 2024

Практична робота № 8

Тема. Жадібні алгоритми. Наближене розв’язання екстремальних задач

Мета: набути практичних навичок застосування деяких жадібних алгоритмів для розв’язання екстремальних задач.

Задачі для самостійного розв’язання

Виконати індивідуальне завдання. Завдання полягає у розв’язанні єдиного завдання для всіх, вибравши граф згідно з варіантом. Номер варіанта відповідає номеру студента у списку групи. У разі, якщо було досягнуто кінця списку задач, потрібно циклічно повернутися на його початок.

Індивідуальне завдання.

1. Розв’язати задачу комівояжера для графа, заданого варіантом, використовуючи код, наведений вище.

2. Візуалізувати граф.

3. Обґрунтувати асимптотику для обох алгоритмів, неведену в табл. 1.4.

Завдання

8. Заданий зважений граф: [(1,3,5), (1,4,8), (1,5,7), (2,3,4), (2,4,6), (2,5,8), (3,4,6), (3,5,5), (4,5,4)]

Ребро (1, 3) з вагою 5

Ребро (1, 4) з вагою 8

Ребро (1, 5) з вагою 7

Ребро (2, 3) з вагою 4

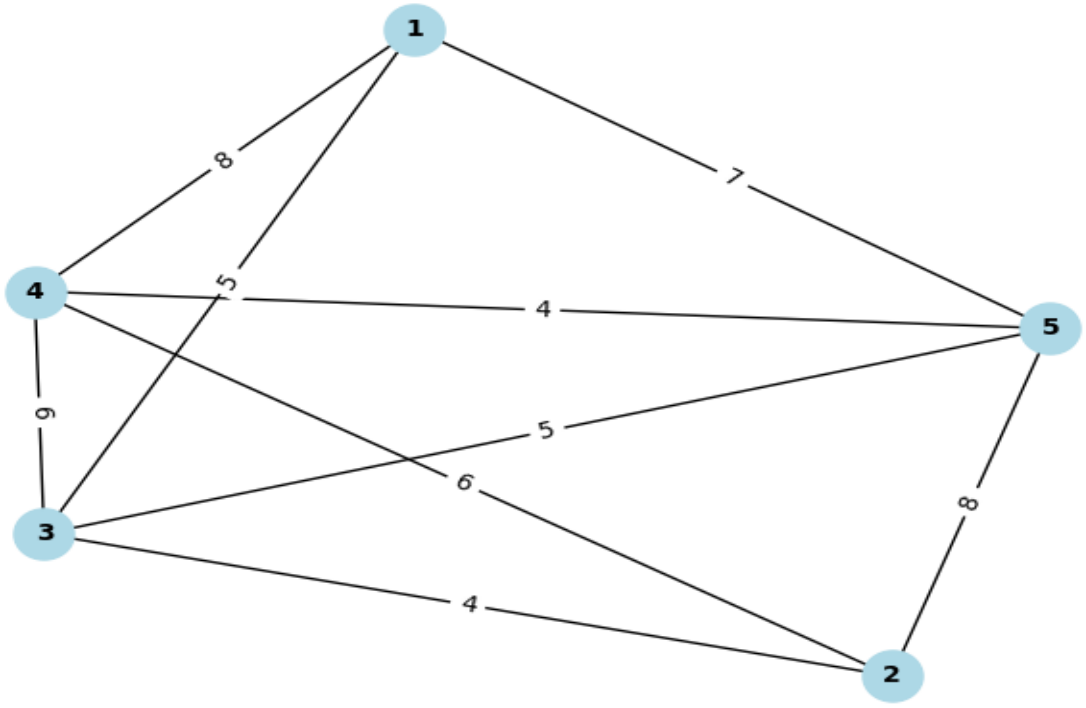
Ребро (2, 4) з вагою 6

Ребро (2, 5) з вагою 8

Ребро (3, 4) з вагою 6

Ребро (3, 5) з вагою 5

Ребро (4, 5) з вагою 4



Оптимальний маршрут: (1, 3, 2, 4, 5, 1)

Оптимальна вартість: 26

1. Груба сила (Brute Force): O(n!)

Алгоритм перебирає всі можливі маршрути, що проходять через усі вершини графа та повертаються у початкову вершину. Для графа з n вершинами:

- **Кількість можливих маршрутів:** (n−1)! (оскільки стартова вершина фіксована, а решта n−1 вершин можуть бути розташовані в довільному порядку). Додаючи повернення до початкової вершини, загальна кількість стає n!.
- **Час виконання:** Для кожного маршруту обчислюється сума ваг усіх ребер, що займає O(n) часу. У загальному випадку: $T(n)=O(n \cdot n!) \approx O(n!)$

(оскільки факторіал n! зростає значно швидше, ніж множник n).

- **Переваги:** Гарантує знаходження оптимального маршруту.
- **Недоліки:** Непрактичний для великих графів через експоненційну складність. Наприклад, для n=10, кількість маршрутів = 10!=3,628,800.

2. Найближчий сусід (Nearest Neighbor): O(n2·logn)

Алгоритм вибирає локально оптимальні рішення: на кожному кроці переходить до найближчої доступної вершини. Хоча цей метод не гарантує оптимального глобального рішення, він ефективніший за перебір.

- **Час виконання:**
  - Алгоритм проходить через n вершин.
  - Для кожної вершини обчислюється відстань до решти $n-1$ вершин. Це вимагає $O(n \cdot \log n)$ операцій для пошуку найближчої вершини (з використанням структур, як-от купа).
  - Загальна складність: $T(n)=O(n \cdot (n \cdot \log n))=O(n2 \cdot \log n)$
- **Переваги:** Значно швидший за метод грубої сили, підходить для великих графів.
- **Недоліки:** Може дати субоптимальний результат (залежить від вибору початкової вершини).

Висновок

1. **Груба сила:** Експоненційна складність $O(n!)$ обумовлена повним перебором усіх можливих маршрутів. Підходить для графів із малою кількістю вершин ($n \leq 10n$).
2. **Найближчий сусід:** Має кращу асимптотику $O(n2 \cdot \log n)$, але може давати неточні результати через жадібний підхід.

Контрольні питання

1. Що таке жадібний алгоритм?

Жадібний алгоритм (Greedy Algorithm) — це алгоритмічний підхід, який на кожному кроці приймає локально оптимальне рішення, сподіваючись, що воно приведе до глобально оптимального результату.

2. Які головні принципи роботи жадібних алгоритмів?

1. **Жадібний вибір:**
   На кожному кроці обирається рішення, яке здається найкращим у поточний момент.
2. **Оптимальність підструктури:**
   Якщо задача може бути розбита на підзадачі, розв’язання кожної підзадачі оптимальним способом гарантує оптимальність загального рішення.

3. **Незалежність вибору:**
   Прийняті на попередніх етапах рішення не впливають на наступні кроки.

3. Яка головна відмінність між жадібними алгоритмами та динамічним програмуванням?

Жадібні алгоритми:

Підхід: Приймають локально оптимальні рішення.

Оптимальність рішення: Не завжди гарантує оптимальне рішення.

Розбиття задачі: Задача розбивається на кроки, кожен із яких вирішується незалежно.

Складність: Як правило, нижча.

Динамічне програмування:

Підхід: Розв’язують задачу шляхом побудови розв’язків для підзадач.

Оптимальність рішення: Гарантує оптимальне рішення.

Розбиття задачі: Задача розбивається на підзадачі, які залежать одна від одної.

Складність: Може бути вищою через необхідність зберігання результатів підзадач.

4. Наведіть приклади задач, які можна розв’язати за допомогою жадібних алгоритмів.

Задача про рюкзак (0/1):
Обрати предмети з максимальною цінністю на одиницю ваги.

Алгоритм Дейкстри:
Пошук найкоротшого шляху в графі.

Оптимізація розкладу:
Наприклад, задача про розклад занять у класах.

Задача покриття множини:
Вибір мінімального підмножини, яке покриває всі елементи.

Призначення задач:
Наприклад, обрати найкращі завдання для виконання за обмежений час.

5. Які можуть бути обмеження у використанні жадібних алгоритмів для розв'язання екстремальних задач?

Невірний вибір на ранніх етапах:
Жадібний підхід може зробити локально оптимальний вибір, який у майбутньому унеможливить отримання глобально оптимального результату.

Відсутність гарантії оптимального результату:
У задачах, де рішення залежить від усієї структури, жадібний алгоритм може давати неточний результат.

Обмеження в задачах із залежностями:
Жадібний підхід працює лише за умов незалежності між етапами розв’язання.

Вимога до специфіки задачі:
Жадібні алгоритми ефективні лише для задач із властивістю *оптимальної підструктури* (наприклад, якщо часткове оптимальне рішення веде до загального).

6. Чому жадібні алгоритми часто використовуються для наближеного розв’язання екстремальних задач?

Швидкість виконання:
Жадібні алгоритми зазвичай мають низьку обчислювальну складність O(n) або O(n log n), що робить їх ефективними для великих задач.

Простота реалізації:
Підхід інтуїтивний і не вимагає складної структури, як у динамічному програмуванні.

Наближене рішення:
У задачах, де оптимальне рішення занадто важке або неможливе для обчислення, жадібні алгоритми забезпечують прийнятне наближення до оптимального результату.

Широке застосування:
У багатьох задачах, як-от маршрутизація мереж, розклад задач або пошук найкоротших шляхів, жадібні алгоритми працюють ефективно й забезпечують рішення достатньої якості.