Звіт

Практична робота №8

Варіант 9

Тема: Жадібні алгоритми. Наближене розв’язання екстремальних задач

Постановка завдань: Заданий зважений граф: [(1,2,3), (1,3,5), (1,4,7), (2,3,6), (2,4,8), (3,4,9)]

Розвязання

Код

import numpy as np

edges = [(1, 2, 3), (1, 3, 5), (1, 4, 7), (2, 3, 6), (2, 4, 8), (3, 4, 9)]

nodes = set()

for edge in edges:

nodes.update(edge[:2])

nodes = list(nodes)

N = len(nodes)

distance\_matrix = np.full((N, N), np.inf)

for u, v, w in edges:

distance\_matrix[u-1][v-1] = w

distance\_matrix[v-1][u-1] = w

# Алгоритм найближчого сусіда

def nearest\_neighbor\_algorithm(dist\_matrix):

N = len(dist\_matrix)

visited = [False] \* N

tour = []

# 1. Вибираємо випадкову стартову точку

current\_point = np.random.randint(N)

tour.append(current\_point)

visited[current\_point] = True

for \_ in range(1, N):

previous\_point = current\_point

min\_distance = float('inf')

for i in range(N):

if not visited[i] and dist\_matrix[previous\_point][i] < min\_distance:

min\_distance = dist\_matrix[previous\_point][i]

current\_point = i

tour.append(current\_point)

visited[current\_point] = True

tour.append(tour[0]) # Повертаємось до стартової точки

return tour

# Розв'язання TSP за допомогою алгоритму найближчого сусіда

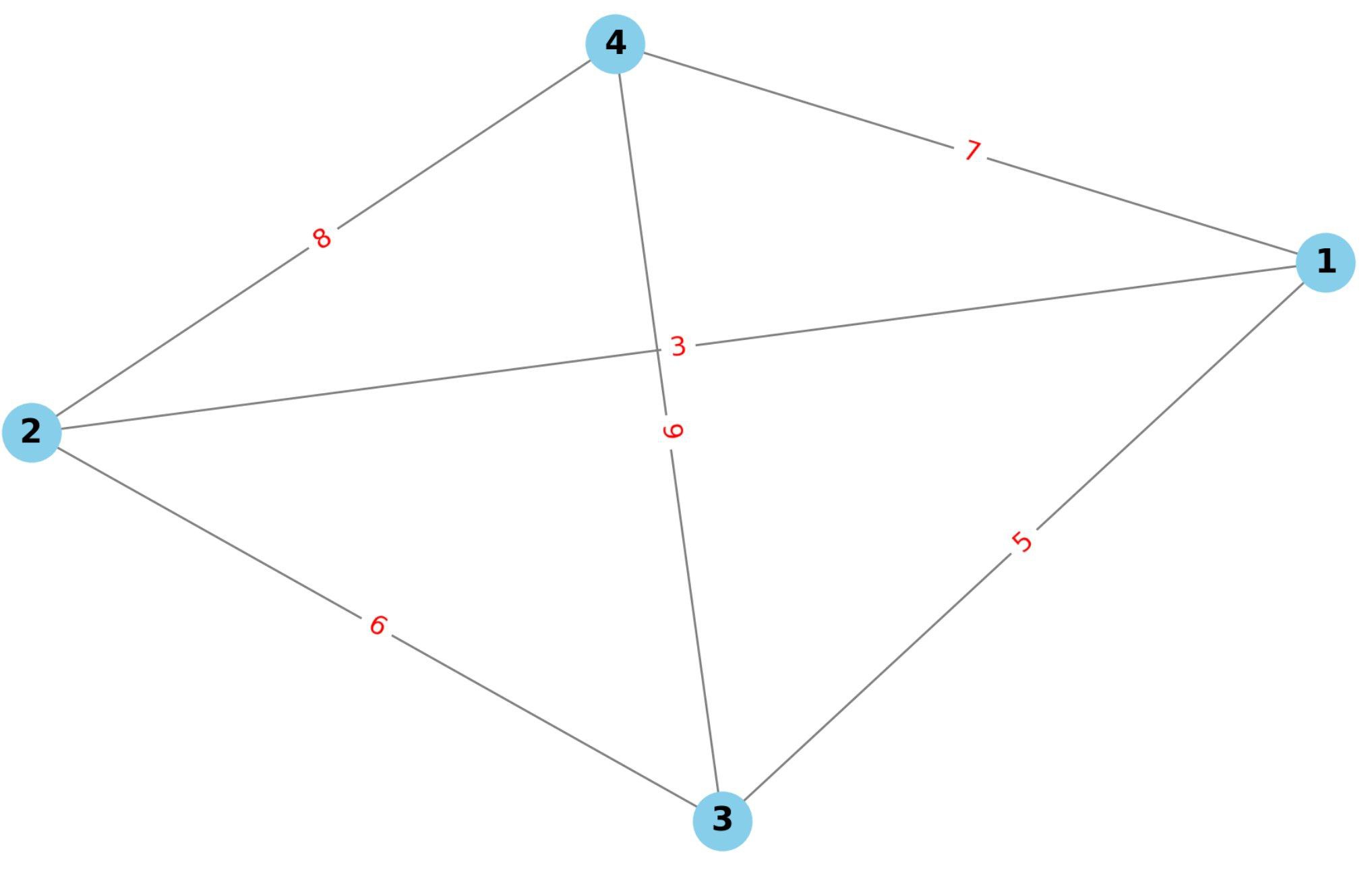
tour = nearest\_neighbor\_algorithm(distance\_matrix)

# Перетворення туру на 1-based індексацію для зручності

tour\_1\_based = [node + 1 for node in tour]

tour\_1\_based

Маршрут, знайдений алгоритмом найближчого сусіда: ([4, 1, 2, 3, 4]).



Алгоритм грубої сили

- Складність: O(n!)

- Обґрунтування: Підхід грубої сили передбачає генерацію всіх можливих перестановок вузлів для знаходження найкоротшого шляху. Для n вузлів існує n! перестановок, що робить складність факторіальною.

Алгоритм найближчого сусіда

- Складність: (O(n^2 log n)

- Обґрунтування: Алгоритм найближчого сусіда включає пошук найближчого невідвіданого вузла на кожному кроці. Для n вузлів це потребує перевірки відстаней від поточного вузла до всіх інших вузлів, що займає O(n) часу. Оскільки це необхідно зробити для кожного з n вузлів, загальна складність складає O(n^2). Додатковий фактор log n зазвичай враховує операції з пріоритетною чергою, якщо вона використовується для підвищення ефективності, що робить загальну складність O(n^2 log n)

Відповідь на контрольні питання

1. Жадібний алгоритм:

- Алгоритм, який приймає локально оптимальне рішення на кожному кроці з надією, що це призведе до глобально оптимального розв’язку.

2. Головні принципи роботи жадібних алгоритмів:

- Вибір жадібного кроку: На кожному етапі обирається найкраще локальне рішення.

- Безповоротність рішень: Прийняті рішення не змінюються у подальшому процесі.

- Локальна оптимальність: Кожний крок має бути найкращим серед усіх можливих на даний момент.

3. Головна відмінність між жадібними алгоритмами та динамічним програмуванням:

- Жадібні алгоритми приймають локально оптимальні рішення без перегляду попередніх рішень, тоді як динамічне програмування використовує попередні обчислення і комбінує їх для досягнення глобального оптимального розв’язку.

4. Приклади задач для жадібних алгоритмів:

- Задача про здачу решти.

- Задача про рюкзак .

- Задача про покриття відрізків

- Задача про оптимальне з'єднання вузлів з алгоритмом Крускала або Пріма.

5. Обмеження у використанні жадібних алгоритмів:

- Не завжди оптимальні: Жадібні алгоритми можуть не знаходити оптимального рішення для всіх задач.

- Потреба у специфічних умовах: Працюють добре лише для задач, які задовольняють умови жадібності (мають властивість "жадібності").

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

- Простота та швидкість: Жадібні алгоритми часто простіші і швидші у реалізації порівняно з іншими методами.

- Наближення до оптимального рішення: Вони можуть надати достатньо близький до оптимального результат за короткий час.