МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ ІМЕНІ МИХАЙЛА
ОСТРОГРАДСЬКОГО

Кафедра комп’ютерної інженерії та електроніки

ЗВІТ З ПРАКТИЧНОЇ РОБОТИ №8

з навчальної дисципліни
«Алгоритми та методи обчислень»

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

Студент гр. КІ-24-1, Варакута О.О
Викладач, Сидоренко В. М

Кременчук 2025

Тема. Жадібні алгоритми. Наближене розв’язання екстремальних задач

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

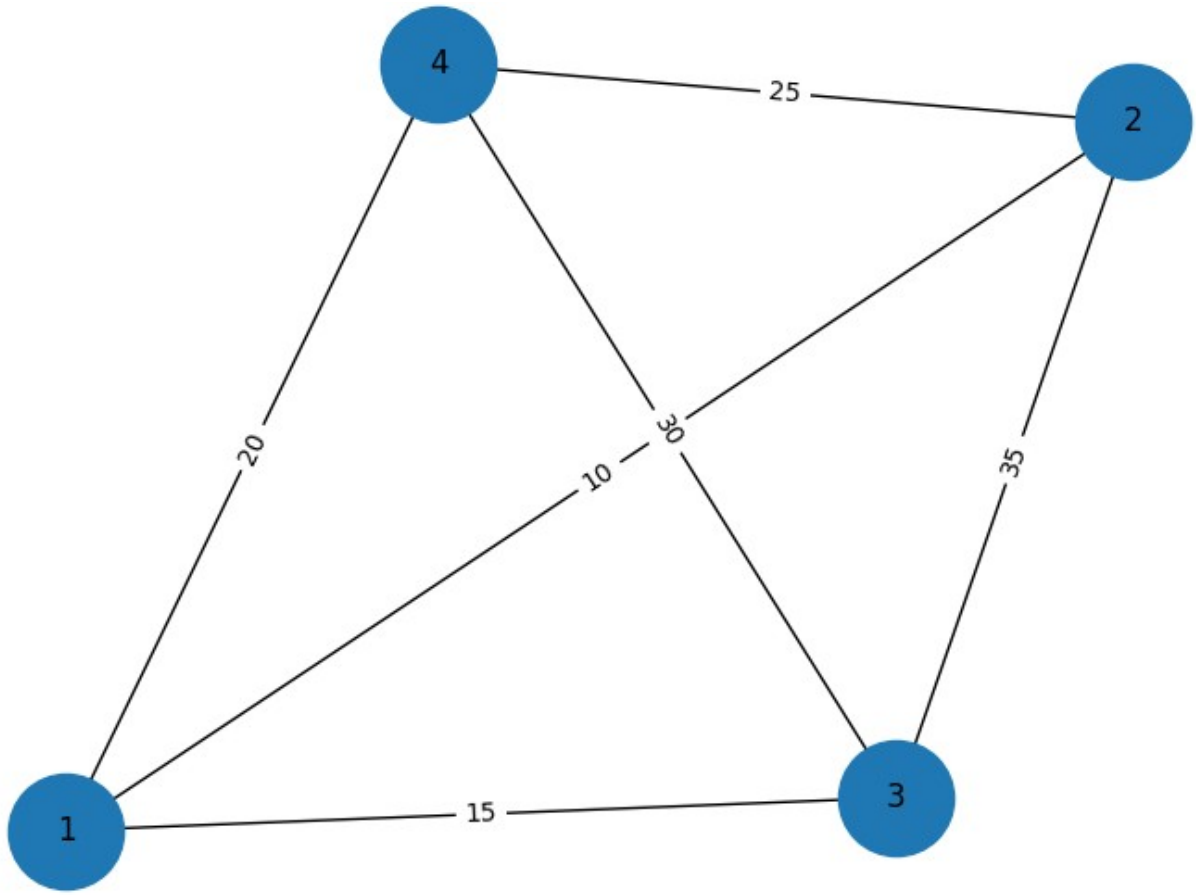
Хід роботи

Завдання:

5. Заданий зважений граф: [(1,2,10), (1,3,15), (1,4,20), (2,3,35), (2,4,25), (3,4,30)]

Візалізуємо граф за допомогою Python:

```
import networkx as nx
import matplotlib.pyplot as plt

def draw_graph(nodes, edges):
    G = nx.Graph()
    G.add_nodes_from(nodes)

    for u, v, w in edges:
        G.add_edge(u, v, weight=w)

    pos = nx.spring_layout(G)

    plt.figure(figsize=(6, 5))
    nx.draw(G, pos, with_labels=True, node_size=2000, font_size=12)
    edge_labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
    plt.show()


nodes = [1, 2, 3, 4]
edges = [
    (1, 2, 10),
    (1, 3, 15),
    (1, 4, 20),
    (2, 3, 35),
    (2, 4, 25),
    (3, 4, 30),
]

draw_graph(nodes, edges)
```



Вихідні дані (зважений повний граф)

Вершини: 1, 2, 3, 4

Ребра з вагами:

(1,2)=10

(1,3)=15

(1,4)=20

(2,3)=35

(2,4)=25

(3,4)=30

Граф повний, неорієнтований, зважений.

Розв’язання 1. Метод грубої сили

Асимптотична складність:

O(n!)

Суть методу: Метод грубої сили полягає у переборі всіх можливих маршрутів комівояжера та виборі маршруту з мінімальною сумарною вагою.Для n вершин кількість можливих маршрутів дорівнює (n−1)!.

Перебір маршрутів (старт з вершини 1)

1 → 2 → 3 → 4 → 1

10 + 35 + 30 + 20 = 95

1 → 2 → 4 → 3 → 1

10 + 25 + 30 + 15 = 80

1 → 3 → 2 → 4 → 1

15 + 35 + 25 + 20 = 95

1 → 3 → 4 → 2 → 1

15 + 30 + 25 + 10 = 80

1 → 4 → 2 → 3 → 1

20 + 25 + 35 + 15 = 95

1 → 4 → 3 → 2 → 1

20 + 30 + 35 + 10 = 95

Результат методу грубої сили

Мінімальна довжина маршруту: 80

Оптимальні маршрути:

1 → 2 → 4 → 3 → 1

1 → 3 → 4 → 2 → 1

Метод гарантує точне оптимальне рішення, але має експоненційну складність.

Розв’язання 2. Метод найближчого сусіда:

Асимптотична складність:

$O(n^2 \cdot \log n)$

Суть методу: Алгоритм починає з фіксованої вершини і на кожному кроці переходить до найближчої ще не відвіданої вершини. Метод є жадібним і не гарантує оптимального результату, але працює значно швидше.

Виконання алгоритму (старт з вершини 1):

Крок 1

Поточна вершина: 1

Найближчі вершини:

до 2: 10

до 3: 15

до 4: 20

Обираємо вершину 2.

Маршрут: 1 → 2

Сума: 10

Крок 2

Поточна вершина: 2

Невідвідані вершини: 3, 4

Відстані:

до 3: 35

до 4: 25

Обираємо вершину 4.

Маршрут: 1 → 2 → 4

Сума: 10 + 25 = 35

Крок 3

Поточна вершина: 4

Залишилась вершина 3.

Маршрут: 1 → 2 → 4 → 3

Сума: 35 + 30 = 65

Крок 4

Повертаємось у початкову вершину 1.

Маршрут: 1 → 2 → 4 → 3 → 1

Сума: 65 + 15 = 80

Результат методу найближчого сусіда

Отриманий маршрут:

1 → 2 → 4 → 3 → 1

Сумарна вага: 80

У даному випадку метод найближчого сусіда знайшов оптимальне рішення, але це не гарантується для довільних графів.

Порівняння методів:

Алгоритм: Груба сила:

Складність: O(n!)

Точність: завжди оптимальне рішення

Недолік: непридатний для великих n

Алгоритм: Найближчий сусід:

Складність: $O(n^2 \cdot \log n)$

Точність: наближене рішення

Перевага: висока швидкість роботи

Реалізація алгоритмів за допомогою Python:

1.Метод грубої сили (Brute Force TSP)

Асимптотична складність: O(n!)

```
import itertools

def tsp_bruteforce(nodes, edges, start=1):
    # словник ваг
    weight = {}
    for u, v, w in edges:
        weight[(u, v)] = w
        weight[(v, u)] = w

    best_cost = float("inf")
    best_path = None

    other_nodes = [v for v in nodes if v != start]

    for perm in itertools.permutations(other_nodes):
        path = [start] + list(perm) + [start]
        cost = 0

        for i in range(len(path) - 1):
            cost += weight[(path[i], path[i + 1])]

        if cost < best_cost:
```

```
            best_cost = cost
            best_path = path

    return best_path, best_cost


nodes = [1, 2, 3, 4]
edges = [
    (1, 2, 10),
    (1, 3, 15),
    (1, 4, 20),
    (2, 3, 35),
    (2, 4, 25),
    (3, 4, 30),
]

path, cost = tsp_bruteforce(nodes, edges)
print("Метод грубої сили")
print("Маршрут:", path)
print("Довжина:", cost)
```

Метод грубої сили

Маршрут: [1, 2, 4, 3, 1]

Довжина: 80

2. Метод найближчого сусіда (Nearest Neighbor)

Асимптотична складність: $O(n^2 \cdot \log n)$

```
def tsp_nearest_neighbor(nodes, edges, start=1):
    # словник ваг
    weight = {}
    for u, v, w in edges:
        weight[(u, v)] = w
        weight[(v, u)] = w

    visited = {start}
    path = [start]
    total_cost = 0
    current = start

    while len(visited) < len(nodes):
        nearest = None
        nearest_cost = float("inf")

        for v in nodes:
            if v not in visited:
                if weight[(current, v)] < nearest_cost:
                    nearest_cost = weight[(current, v)]
                    nearest = v
```

```
            path.append(nearest)
            visited.add(nearest)
            total_cost += nearest_cost
            current = nearest

        total_cost += weight[(current, start)]
        path.append(start)

        return path, total_cost

def main():
        nodes = [1, 2, 3, 4]
        edges = [
            (1, 2, 10),
            (1, 3, 15),
            (1, 4, 20),
            (2, 3, 35),
            (2, 4, 25),
            (3, 4, 30),
        ]

        path, cost = tsp_nearest_neighbor(nodes, edges)

        print("Метод найближчого сусіда")
        print("Маршрут:", path)
        print("Довжина:", cost)

main()
```

Метод найближчого сусіда

Маршрут: [1, 2, 4, 3, 1]

Довжина: 80

Висновок: Для заданого зваженого графа задача комівояжера була розв’язана методом грубої сили та методом найближчого сусіда. Обидва алгоритми дали однаковий маршрут довжиною 80. Метод грубої сили гарантує оптимальне рішення, але має експоненційну складність, тоді як метод найближчого сусіда є швидшим, але наближеним.

Контрольні питання

1.Що таке жадібний алгоритм?

Жадібний алгоритм — це алгоритм, який на кожному кроці приймає локально оптимальне рішення, тобто вибирає найкращий варіант з доступних у поточний момент, не враховуючи можливі наслідки цього вибору у майбутньому. Передбачається, що послідовність таких локально

оптимальних рішень приведе до оптимального або близького до оптимального глобального результату.

2.Які головні принципи роботи жадібних алгоритмів?

Основними принципами роботи жадібних алгоритмів є вибір локально оптимального рішення на кожному кроці, незворотність прийнятих рішень та поступове побудування розв’язку. Алгоритм не переглядає раніше зроблені вибори та не повертається до попередніх станів, що дозволяє суттєво зменшити час обчислень.

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

Головна відмінність полягає у підході до прийняття рішень. Жадібні алгоритми приймають рішення на основі локальної оптимальності без аналізу майбутніх наслідків, тоді як динамічне програмування розглядає всі можливі підзадачі, зберігає проміжні результати та гарантує знаходження глобально оптимального розв’язку. Динамічне програмування потребує більше пам’яті та часу, але забезпечує точний результат.

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

Жадібні алгоритми застосовуються для розв’язання таких задач, як побудова мінімального остовного дерева графа (алгоритми Крускала і Прима), знаходження найкоротших шляхів у графах без від’ємних ваг (алгоритм Дейкстри), кодування Хаффмана, задача вибору інтервалів, задача розміну монет у певних системах номіналів, а також наближене розв’язання задачі комівояжера.

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

Основним обмеженням жадібних алгоритмів є відсутність гарантії знаходження глобально оптимального розв’язку для всіх задач. У багатьох екстремальних задачах локально оптимальні рішення не приводять до глобального оптимуму. Крім того, жадібні алгоритми залежать від структури задачі та можуть давати неправильні або неоптимальні результати, якщо задача не задовольняє властивість жадібного вибору.

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

Жадібні алгоритми часто використовуються для наближеного розв’язання екстремальних задач через їхню простоту реалізації, високу швидкість роботи та низькі вимоги до пам’яті. Для задач з великою кількістю можливих рішень, де точні алгоритми мають надто високу обчислювальну складність, жадібні алгоритми дозволяють отримати достатньо якісний результат за прийнятний час, що робить їх практично корисними.