МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ   
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕЛЕКТРИЧНОЇ ІНЖЕНЕРІЇ   
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА АВТОМАТИЗАЦІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

НАВЧАЛЬНА ДИСЦИПЛІНА  
Алгоритми та методи обчислень

ЗВІТ

З ПРАКТИЧНИХ РОБІТ

Виконав:

студент групи КН-23-1

Сидоренко А.Ю.

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.

**Хід роботи**

**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.

18. Заданий зважений граф: [(1,2,6), (1,3,7), (1,4,9), (1,5,10), (2,3,5), (2,4,6), (2,5,8), (3,4,4), (3,5,6), (4,5,3)]

import numpy as np #Код з лабораторної з використанням бібліотеки виводу

import networkx as nx

import matplotlib.pyplot as plt

# Заданий граф

edges = [(1, 2, 6), (1, 3, 7), (1, 4, 9), (1, 5, 10),

(2, 3, 5), (2, 4, 6), (2, 5, 8),

(3, 4, 4), (3, 5, 6), (4, 5, 3)]

G = nx.Graph()

G.add\_weighted\_edges\_from(edges)

# Створимо випадкові координати для точок

np.random.seed(0)

points = np.random.rand(5, 2) \* 10 # 5 точок у 2D-просторі

# Функція для знаходження найкоротшого шляху за алгоритмом найближчого сусіда

def nearest\_neighbor\_algorithm(points):

N = len(points)

visited = [False] \* N

tour = []

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]:

distance = np.linalg.norm(points[previous\_point] - points[i])

if distance < min\_distance:

min\_distance = distance

current\_point = i

tour.append(current\_point)

visited[current\_point] = True

tour.append(tour[0])

return tour

# Знайдемо шлях за алгоритмом найближчого сусіда

tour = nearest\_neighbor\_algorithm(points)

tour\_length = sum(np.linalg.norm(points[tour[i]] - points[tour[i+1]]) for i in range(len(tour)-1))

# Виведемо результати

print("Шлях за алгоритмом найближчого сусіда:", tour)

print("Довжина шляху:", tour\_length)

# Візуалізація графа

pos = {i+1: points[i] for i in range(len(points))}

labels = nx.get\_edge\_attributes(G, 'weight')

nx.draw(G, pos, with\_labels=True, node\_color='skyblue', node\_size=700, font\_size=10)

nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=labels)

plt.show()

# Візуалізація шляху

plt.figure()

for i in range(len(tour)-1):

p1, p2 = points[tour[i]], points[tour[i+1]]

plt.plot([p1[0], p2[0]], [p1[1], p2[1]], 'bo-')

for i, point in enumerate(points):

plt.text(point[0], point[1], f'{i+1}', fontsize=12, ha='right')

plt.xlabel('X')

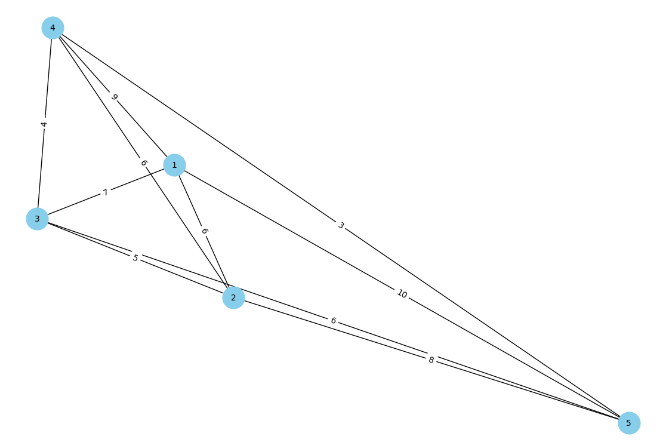
plt.ylabel('Y')

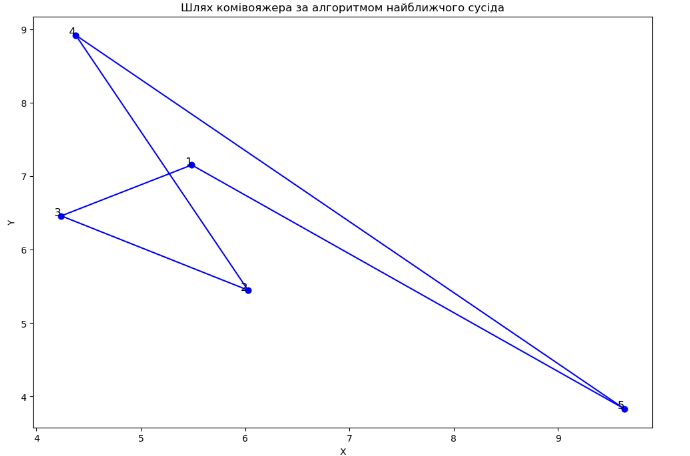
plt.title('Шлях комівояжера за алгоритмом найближчого сусіда')

plt.show()

Шлях за алгоритмом найближчого сусіда: [0, 2, 1, 3, 4, 0]

Довжина шляху: 19.956256435123457





Для графа з n вершинами кількість можливих маршрутів складає ( n − 1 ) ! (оскільки маршрут є циклом), тому алгоритм грубої сили і визначається кількістю перестановок. Генерація всіх перестановок вимагає (n-1)!, а обхід операцій вимагає O(n).

Отжее O(n\*(n-1)!)= O(n!)

В найближчому сусіді обирається початкова точка О(1). Далі ми шукаємо найближчу невідвідану вершину включаючи перевірку всіх n-1 вершин і виконуємо цю перевірку n разів.

Отже O(n\*(n-1))=O(n^2)

Однак, якщо використовувати ефективні структури даних для зберігання та пошуку мінімальних відстаней (наприклад, пріоритетну чергу), цей пошук можна оптимізувати. В такому випадку складність може включати логарифмічний множник:

O(n^2\*logn)

**Контрольні питання**

1. **Що таке жадібний алгоритм?**

Жадібні алгоритми — це клас алгоритмів, які приймають локально оптимальні рішення на кожному кроці з надією, що вони призведуть до глобально оптимального розв'язку задачі.

1. **Які головні принципи роботи жадібних алгоритмів?**

Жадібний вибір

Оптимальна підструктура

Локальна оптимальність

Незмінний вибір

Принцип залишку

1. **Яка головна відмінність між жадібними алгоритмами та динамічним програмуванням?**

Головна відмінність це використання. Жадібні алгоритми частіше застосовуються в ситуаціях, коли можна гарантувати, що локально оптимальні вибори приведуть до глобально оптимального рішення. Динамічне програмування застосовується там, де задачу можна розбити на підзадачі, які пересікаються, і можна скористатися мемоізацією для оптимізації. Та

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

Задача про здачу решти, задача про рюкзак, алгоритм Прима та Крускала, задача про оптимальний вибір активностей

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

Жадібні алгоритми є потужним інструментом для багатьох задач, але їхнє використання вимагає обережності та аналізу конкретної задачі. Перед застосуванням жадібного підходу слід ретельно вивчити особливості задачі та перевірити, чи відповідають вони обмеженням та умовам жадібного алгоритму.

Головні обмеження це наявність циклів, великі обчислювальні витрати, вразливість до змін та неоптимальність локальних виборів

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

Їх використовують через такі причини, як:

* Простота і швидкість
* Завжди отримання близького результату, коли немає точного оптимуму
* Ефективність для певних задач
* Застосування в реальному часі