МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

ІНСТИТУТ ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ФУНДАМЕНТАЛЬНИХ НАУК

Кафедра прикладної математики

**ЗВІТ**

про виконання лабораторної роботи №4 із дисципліни:

“**Математичні основи штучного інтелекту”**

Виконав:

студент групи ПМ-31

Матолін Марко

Прийняв:

доцент кафедри ПМ

Пабирівський В.В.

Львів ‒ 2023

**Тема:** «Мурашиний алгоритм»

# Завдання

Розв’язати задачу комівояжера із застосуванням мурашиного алгоритму.

Етапи виконання завдання

1. Згенерувати карту маршрутів, які відвідує комівояжер:

1. кількість міст N обрати випадковим чином з діапазону 25...35;

припускати, що усі міста з’єднані між собою дорогами;

1. довжини доріг між містами вибрати випадковим чином як ціле Примітка:

число у діапазоні 10..100.

Для зручності повторного використання передбачити запис даних щодо згенерованої карти у файл та можливість завантажити раніше збережену карту з файлу.

1. Реалізувати мурашиний алгоритм для розв’язання задачі комівояжера (див. зразок реалізації у [1]).
2. Провести послідовність з 10 симуляцій на однаковій карті та провести аналіз отриманих результатів із урахуванням таких факторів:
3. Кількістьмураху«мурашнику».
4. Константа випаровування ферменту .
5. Співвідношення констант–константаваги/ кількостіферменту, константа видимості (довжини шляху)

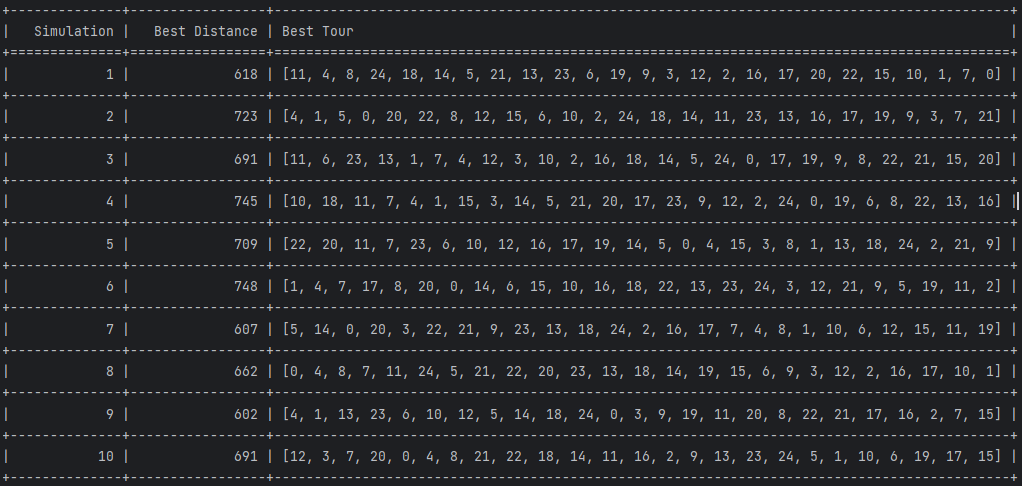
7. Порівняйте кількість кластерів та якість кластеризації (можна просто оцінити середньо-зважене розмірів утворених кластерів згідно заданої міри віддалі для кожного із методів).

# Варіант-3

Код програми:

import random  
import math  
from tabulate import tabulate  
  
def generate\_route\_map(file\_path, num\_cities\_range, road\_distance\_range):  
 num\_cities = random.randint(\*num\_cities\_range)  
 route\_map = [[0] \* num\_cities for \_ in range(num\_cities)]  
  
 for i in range(num\_cities):  
 for j in range(i+1, num\_cities):  
 distance = random.randint(\*road\_distance\_range)  
 route\_map[i][j] = distance  
 route\_map[j][i] = distance  
  
 with open(file\_path, 'w') as file:  
 file.write(f'{num\_cities}\n')  
 for row in route\_map:  
 file.write(' '.join(map(str, row)) + '\n')  
  
def load\_route\_map(file\_path):  
 route\_map = []  
  
 with open(file\_path, 'r') as file:  
 num\_cities = int(file.readline().strip())  
 for \_ in range(num\_cities):  
 row = list(map(int, file.readline().strip().split()))  
 route\_map.append(row)  
  
 return route\_map  
  
def ant\_colony\_tsp(route\_map, num\_ants, evaporation\_rate, alpha, beta):  
 num\_cities = len(route\_map)  
 pheromone = [[1.0] \* num\_cities for \_ in range(num\_cities)]  
 best\_distance = float('inf')  
 best\_tour = []  
  
 for \_ in range(num\_ants):  
 start\_city = random.randint(0, num\_cities-1)  
 ant = Ant(start\_city, num\_cities, alpha, beta)  
  
 while not ant.is\_completed():  
 ant.select\_next\_city(route\_map, pheromone)  
  
 distance = ant.get\_total\_distance()  
 if distance < best\_distance:  
 best\_distance = distance  
 best\_tour = ant.get\_tour()  
  
 ant.update\_pheromone(pheromone, distance)  
  
 return best\_distance, best\_tour  
  
class Ant:  
 def \_\_init\_\_(self, start\_city, num\_cities, alpha, beta):  
 self.alpha = alpha  
 self.beta = beta  
 self.num\_cities = num\_cities  
 self.visited = [False] \* num\_cities  
 self.visited[start\_city] = True  
 self.tour = [start\_city]  
 self.total\_distance = 0.0  
  
 def select\_next\_city(self, route\_map, pheromone):  
 current\_city = self.tour[-1]  
 probabilities = [0.0] \* self.num\_cities  
 total = 0.0  
  
 for i in range(self.num\_cities):  
 if not self.visited[i]:  
 probabilities[i] = (  
 math.pow(pheromone[current\_city][i], self.alpha)  
 \* math.pow(1.0 / route\_map[current\_city][i], self.beta)  
 )  
 total += probabilities[i]  
  
 rand = random.uniform(0, total)  
 current = 0.0  
  
 for i in range(self.num\_cities):  
 if not self.visited[i]:  
 current += probabilities[i]  
 if current >= rand:  
 next\_city = i  
 break  
  
 self.visited[next\_city] = True  
 self.tour.append(next\_city)  
 self.total\_distance += route\_map[current\_city][next\_city]  
  
 def is\_completed(self):  
 return all(self.visited)  
  
 def get\_total\_distance(self):  
 return self.total\_distance  
  
 def get\_tour(self):  
 return self.tour  
  
 def update\_pheromone(self, pheromone, distance):  
 evaporation = 1.0 - evaporation\_rate  
  
 for i in range(self.num\_cities - 1):  
 city\_a = self.tour[i]  
 city\_b = self.tour[i+1]  
 pheromone[city\_a][city\_b] = evaporation \* pheromone[city\_a][city\_b] + (1.0 / distance)  
  
 last\_city = self.tour[-1]  
 first\_city = self.tour[0]  
 pheromone[last\_city][first\_city] = evaporation \* pheromone[last\_city][first\_city] + (1.0 / distance)  
  
  
num\_ants = 10  
evaporation\_rate = 0.5  
alpha = 1  
beta = 2  
  
generate\_route\_map('route\_map.txt', (25, 35), (10, 100))  
route\_map = load\_route\_map('route\_map.txt')  
  
simulation\_results = []  
  
for simulation in range(10):  
 best\_distance, best\_tour = ant\_colony\_tsp(route\_map, num\_ants, evaporation\_rate, alpha, beta)  
 simulation\_results.append([simulation + 1, best\_distance, best\_tour])  
  
table\_headers = ["Simulation", "Best Distance", "Best Tour"]  
table = tabulate(simulation\_results, headers=table\_headers, tablefmt="grid")  
print(table)

**Результат виконання:**



**Висновок:**

На лабораторній роботі я ознайомилась з мурашиним алгоритмом

та використала його на практиці для розв’язання задачі комівояжера