**ЛАБОРАТОРНА РОБОТА № 7**

**ДОСЛІДЖЕННЯ МУРАШИНИХ АЛГОРИТМІВ**

***Мета:*** використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися дослідити метод мурашиних колоній

**Хід роботи**

Завдання. Дослідження мурашиного алгоритму на прикладі рішення задачі комівояжера

При розробці програми було використано мережу Інтернет для пошуку можливих реалізацій необхідних методів з огляду на наданий приклад мовою Matlab.

Лістинг коду файлу Task\_1.py:

import numpy as np  
import matplotlib.pyplot as plt  
  
  
# Карта відстаней з феромонами  
class CityMap:  
 def \_\_init\_\_(*self*, distances\_matrix, cities\_count):  
 *self*.distances = distances\_matrix  
 *self*.numberOfCities = cities\_count  
 *self*.pheromones = [[np.random.rand() for j in range(cities\_count)] for i in range(cities\_count)]  
  
 # Оновлення значення феромонів  
 def upd\_pheromones(*self*, evaporation\_rate, pheromone\_delta):  
 for i, row in enumerate(*self*.pheromones):  
 for j, col in enumerate(row):  
 *self*.pheromones[i][j] \*= (1 - evaporation\_rate)  
 *self*.pheromones[i][j] += pheromone\_delta[i][j]  
  
  
class Ant:  
 def \_\_init\_\_(*self*, city\_start):  
 *self*.startingCity = city\_start  
 *self*.currentCity = city\_start  
 *self*.distance = 0  
 *self*.visitedCities = [city\_start]  
  
 # Переміщення мурахи в нове місто

def move(*self*, city\_new, distance):  
 *self*.currentCity = city\_new  
 *self*.visitedCities.append(city\_new)  
 *self*.distance += distance

class Colony:  
 maxColonyCycles = 50  
 pheromoneAddition = 0.0005  
 pheromoneEvaporationRate = 0.2  
 pheromoneImportance = 0.01  
 distanceImportance = 9.5  
 antCanVisitPreviousCities = False  
  
 def \_\_init\_\_(*self*, ants\_num):  
 *self*.numberOfAnts = ants\_num  
  
 # Пошук найкоротшого шляху  
 def find\_route(*self*, city\_map, city\_num):  
 min\_dist = float('inf')  
 route = []  
 for cycle in range(*self*.maxColonyCycles):  
 pheromones\_delta = [[0.0 for i in range(city\_map.numberOfCities)] for j in range(city\_map.numberOfCities)]  
 for antNumber in range(*self*.numberOfAnts):  
 ant = Ant(city\_num)  
 while len(ant.visitedCities) < city\_map.numberOfCities:  
 next\_city = *self*.get\_next\_city(ant, city\_map)  
 ant.move(next\_city, city\_map.distances[ant.currentCity][next\_city])  
 ant\_dist = ant.distance + city\_map.distances[ant.currentCity][ant.startingCity]  
 if ant\_dist < min\_dist:  
 min\_dist = ant\_dist  
 route = ant.visitedCities  
 for city in range(len(ant.visitedCities) - 1):  
 pheromones\_delta[ant.visitedCities[city]][  
 ant.visitedCities[city + 1]] += *self*.pheromoneAddition / ant\_dist  
 city\_map.upd\_pheromones(*self*.pheromoneEvaporationRate, pheromones\_delta)  
  
 return min\_dist, route  
  
 # Формування списку ймовірностей переміщення в місто для мурахи  
 def get\_probabilities(*self*, ant, city\_map):  
 result = [0 for i in range(city\_map.numberOfCities)]  
 total\_probability = 0  
 for newCity in range(city\_map.numberOfCities):  
 if (newCity != ant.currentCity) and (*self*.antCanVisitPreviousCities or newCity not in ant.visitedCities):  
 probability = pow(city\_map.pheromones[ant.currentCity][newCity], *self*.pheromoneImportance) \* pow(  
 1 / city\_map.distances[ant.currentCity][newCity], *self*.distanceImportance)  
 result[newCity] = probability  
 total\_probability += probability  
 result = [result[i] / total\_probability for i in range(city\_map.numberOfCities)]  
 return result  
  
 # Вибір наступного міста для мурахи  
 def get\_next\_city(*self*, ant, city\_map):  
 probabilities = *self*.get\_probabilities(ant, city\_map)  
 random\_value = np.random.rand()  
 for i in range(city\_map.numberOfCities):  
 if probabilities[i] > random\_value:  
 return i  
 else:  
 random\_value -= probabilities[i]  
 return -1  
  
  
# Відстані між містами  
distance = [  
 [0, 645, 868, 125, 748, 366, 256, 316, 1057, 382, 360, 471, 428, 593, 311, 844, 602, 232, 575, 734, 521, 120,  
 343, 312, 396],  
 [645, 0, 252, 664, 81, 901, 533, 294, 394, 805, 975, 343, 468, 196, 957, 446, 430, 877, 1130, 213, 376, 765,  
 324, 891, 672],  
 [868, 252, 0, 858, 217, 1171, 727, 520, 148, 1111, 1221, 611, 731, 390, 1045, 591, 706, 1100, 1391, 335, 560,  
 988, 547, 1141, 867],  
 [125, 664, 858, 0, 738, 431, 131, 407, 1182, 257, 423, 677, 557, 468, 187, 803, 477, 298, 671, 690, 624, 185,  
 321, 389, 271],  
 [748, 81, 217, 738, 0, 1119, 607, 303, 365, 681, 833, 377, 497, 270, 925, 365, 477, 977, 1488, 287, 297, 875,  
 405, 957, 747],  
 [366, 901, 1171, 431, 1119, 0, 561, 618, 1402, 328, 135, 747, 627, 898, 296, 1070, 908, 134, 280, 1040, 798,  
 246, 709, 143, 701],  
 [256, 533, 727, 131, 607, 561, 0, 298, 811, 388, 550, 490, 489, 337, 318, 972, 346, 427, 806, 478, 551, 315,  
 190, 538, 149],  
 [316, 294, 520, 407, 303, 618, 298, 0, 668, 664, 710, 174, 294, 246, 627, 570, 506, 547, 883, 387, 225, 435,  
 126, 637, 363],  
 [1057, 394, 148, 1182, 365, 1402, 811, 668, 0, 1199, 1379, 857, 977, 474, 1129, 739, 253, 1289, 1539, 333, 806,  
 1177, 706, 1292, 951],  
 [382, 805, 1111, 257, 681, 328, 388, 664, 1199, 0, 152, 780, 856, 725, 70, 1052, 734, 159, 413, 866, 869, 263,  
 578, 336, 949],  
 [360, 975, 1221, 423, 833, 135, 550, 710, 1379, 152, 0, 850, 970, 891, 232, 1173, 896, 128, 261, 1028, 1141,  
 240, 740, 278, 690],  
 [471, 343, 611, 677, 377, 747, 490, 174, 857, 780, 850, 0, 120, 420, 864, 282, 681, 754, 999, 556, 51, 590, 300,  
 642, 640],  
 [428, 468, 731, 557, 497, 627, 489, 294, 977, 856, 970, 120, 0, 540, 741, 392, 800, 660, 1009, 831, 171, 548,  
 420, 515, 529],  
 [593, 196, 390, 468, 270, 898, 337, 246, 474, 725, 891, 420, 540, 0, 665, 635, 261, 825, 1149, 141, 471, 653,  
 279, 892, 477],  
 [311, 957, 1045, 187, 925, 296, 318, 627, 1129, 70, 232, 864, 741, 665, 0, 1157, 664, 162, 484, 805, 834, 193,  
 508, 331, 458],  
 [844, 446, 591, 803, 365, 1070, 972, 570, 739, 1052, 1173, 282, 392, 635, 1157, 0, 896, 1097, 1363, 652, 221,  
 964, 696, 981, 1112],  
 [602, 430, 706, 477, 477, 908, 346, 506, 253, 734, 896, 681, 800, 261, 664, 896, 0, 774, 1138, 190, 732, 662,  
 540, 883, 350],  
 [232, 877, 1100, 298, 977, 134, 427, 547, 1289, 159, 128, 754, 660, 825, 162, 1097, 774, 0, 338, 987, 831, 112,  
 575, 176, 568],  
 [575, 1130, 1391, 671, 1488, 280, 806, 883, 1539, 413, 261, 999, 1009, 1149, 484, 1363, 1138, 338, 0, 1299,  
 1065, 455, 984, 444, 951],  
 [734, 213, 335, 690, 287, 1040, 478, 387, 333, 866, 1028, 556, 831, 141, 805, 652, 190, 987, 1299, 0, 576, 854,  
 420, 1036, 608],  
 [521, 376, 560, 624, 297, 798, 551, 225, 806, 869, 1141, 51, 171, 471, 834, 221, 732, 831, 1065, 576, 0, 641,  
 351, 713, 691],  
 [120, 765, 988, 185, 875, 246, 315, 435, 1177, 263, 240, 590, 548, 653, 193, 964, 662, 112, 455, 854, 641, 0,  
 463, 190, 455],  
 [343, 324, 547, 321, 405, 709, 190, 126, 706, 578, 740, 300, 420, 279, 508, 696, 540, 575, 984, 420, 351, 463,  
 0, 660, 330],  
 [312, 891, 1141, 389, 957, 143, 538, 637, 1292, 336, 278, 642, 515, 892, 331, 981, 883, 176, 444, 1036, 713,  
 190, 660, 0, 695],  
 [396, 672, 867, 271, 747, 701, 149, 363, 951, 949, 690, 640, 529, 477, 458, 1112, 350, 568, 951, 608, 691, 455,  
 330, 695, 0]  
]  
  
# Список міст  
  
cities = [  
 'Вінниця', 'Дніпро', 'Донецьк', 'Житомир', 'Запоріжжя', 'Івано-Франківськ', 'Київ', 'Кропивницький',  
 'Луганськ', 'Луцьк', 'Львів', 'Миколаїв', 'Одеса', 'Полтава', 'Рівне', 'Сімферополь', 'Суми', 'Тернопіль',  
 'Ужгород', 'Харків', 'Херсон', 'Хмельницький', 'Черкаси', 'Чернівці', 'Чернігів'  
]  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Пошук відповіді задачі  
 cityMap = CityMap(distance, len(distance[0]))  
 colony = Colony(len(distance[0]))  
 result = colony.find\_route(cityMap, 10)  
 print(f"Отриманий найкоротший шлях: {result[0]} км")  
  
 # Вивід отриманого маршруту  
 cityRoutes = "Отриманий маршрут: "  
 for i in result[1]:  
 cityRoutes += cities[i]  
 if i != result[1][-1]:  
 cityRoutes += "->"  
 print(cityRoutes)  
  
 # Графічне відображення отриманих даних  
 fig = plt.figure(figsize=(13, 13))  
 plt.xticks([i + 1 for i in range(25)])  
 plt.yticks([i for i in range(25)], cities)  
 plt.xlabel("Номери міст")  
 plt.ylabel("Назви міст")  
 plt.title("Оптимальний маршрут мурахи")  
 plt.plot([i + 1 for i in range(25)], result[1], ms=12, marker='\*', mfc='r',  
 mec='black', mew=2, color='black', ls="--")  
 plt.grid()  
 plt.show()

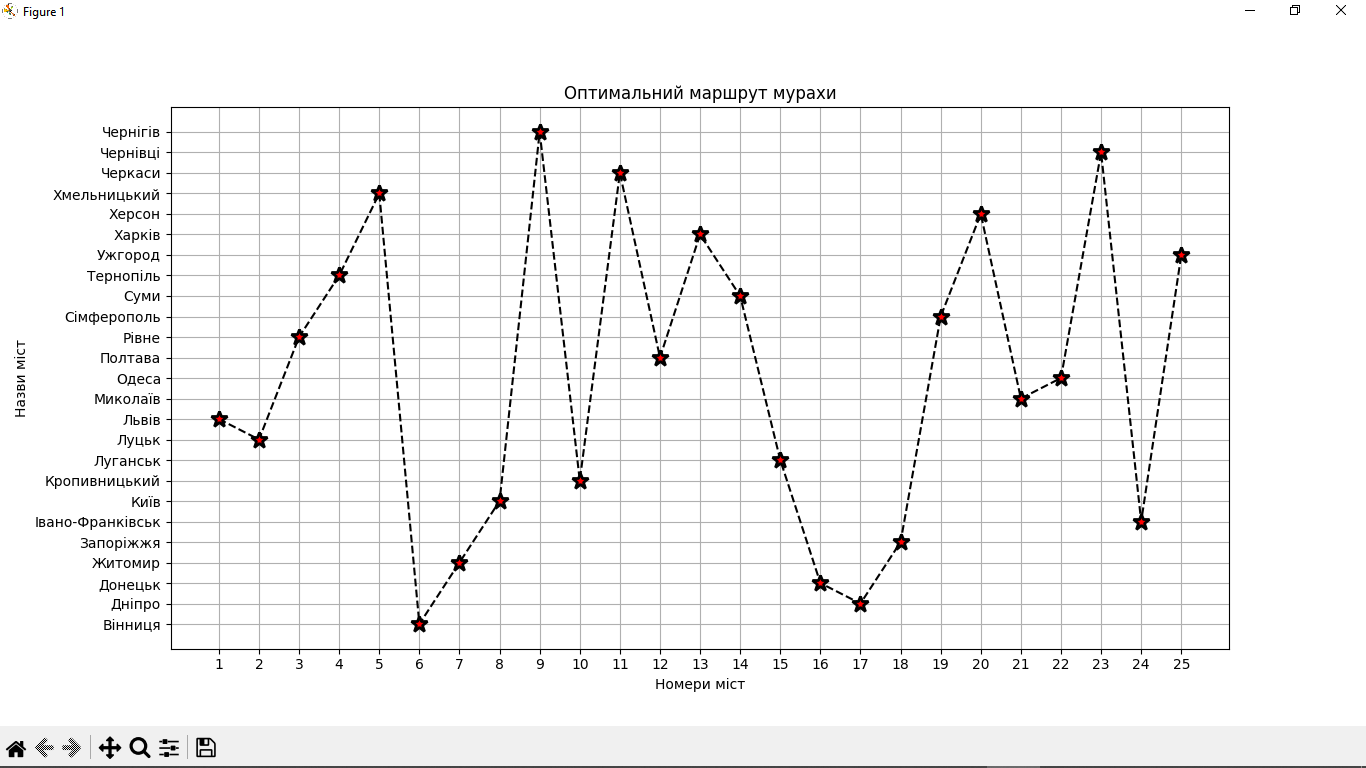


Рис.7.1 – Графічне відображення пройденого оптимального шляху



Рис.7.2 – Текстове повідомлення від програми після виконання роботи

***Висновок:*** під час виконання завдання лабораторної роботи отримано досліджено метод мурашиних колоній та навички роботи з ним використовуючи спеціалізовані бібліотеки та мову програмування Python.

Протягом виконання завдання було розроблено класи CityMap, Ant, Colony та розроблено методи для них для реалізації мурашиного алгоритму та пошуку найкоротшого шляху до кожного міста від стартового за варіантом.

Частину коду було взято та використано як основу для існуючого з мережі Інтернет, а саме з інтернет-форумів та схожих опублікованих програм.

Код програми зберігається у репозиторії за посиланням: <https://github.com/nikitoss888/AI_LR7>.