**Лабораторна робота №7**

**ДОСЛІДЖЕННЯ МУРАШИНИХ АЛГОРИТМІВ**

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

**Хід роботи:**

**Завдання 2.1**. Дослідження мурашиного алгоритму на прикладі рішення задачі комівояжера.

У файлі *Відстані між обласними центрами України.docx* міститься таблиця відстаней між обласними центрами України. Міста пронумеровані по алфавіту. Ваш номер за журналом групи (або номер за табличкою у рейтингу) повинен відповідати місту з якого ви будете починати виїзд. Наприклад: за рейтинговою таблицею - № 6 Драк Тарас Сергійович починає з № 6. Івано-Франківськ. Виїзд починається і закінчується в цьому місті.

Використовуючи мову Python, розробити програму, що реалізує метод мурашиних колоній, для рішення задачі комівояжера, який їздить по містах України.

Таблиця №1

|  |  |
| --- | --- |
| **Варіант №** | **Місто** |
| 6 | Івано-Франківськ |

Лістинг програми:

import random  
import matplotlib.pyplot as plt  
  
# 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'  
]  
  
class Ant:  
 def \_\_init\_\_(self, colony, start\_city):  
 self.colony = colony  
 self.pheromone\_map = [1] \* len(self.colony.cities) # Створення масиву феромонів для кожного міста  
 self.visited\_cities = [start\_city] # Список відвіданих міст  
 self.total\_distance = 0 # Загальна відстань подорожі  
  
 def choose\_next\_city(self):  
 current\_city = self.visited\_cities[-1] # Поточне місто  
 unvisited\_cities = [city for city in self.colony.cities if city not in self.visited\_cities] # Невідвідані міста  
  
 probabilities = []  
 for city in unvisited\_cities:  
 pheromone\_level = self.pheromone\_map[self.colony.cities.index(city)] # Рівень феромонів для поточного міста  
 distance = self.colony.distance[self.colony.cities.index(current\_city)][self.colony.cities.index(city)] # Відстань між поточним та наступним містом  
 probability = pheromone\_level \*\* self.colony.alpha / distance \*\* self.colony.beta # Ймовірність вибору наступного міста  
 probabilities.append(probability)  
  
 total\_probability = sum(probabilities)  
 probabilities = [prob / total\_probability for prob in probabilities] # Нормалізація ймовірностей  
  
 next\_city = random.choices(unvisited\_cities, probabilities)[0] # Вибір наступного міста з використанням рулеткового вибору  
 return next\_city  
  
 def travel(self):  
 while len(self.visited\_cities) < len(self.colony.cities):  
 next\_city = self.choose\_next\_city()  
 self.visited\_cities.append(next\_city)  
 current\_city\_index = self.colony.cities.index(self.visited\_cities[-2]) # Індекс поточного міста у списку міст  
 next\_city\_index = self.colony.cities.index(next\_city) # Індекс наступного міста у списку міст  
 self.total\_distance += self.colony.distance[current\_city\_index][next\_city\_index] # Додавання відстані між поточним та наступним містом  
  
 # Повертаємось в початкове місто  
 self.total\_distance += self.colony.distance[self.colony.cities.index(self.visited\_cities[-1])][self.colony.cities.index(self.visited\_cities[0])]  
 self.visited\_cities.append(self.visited\_cities[0]) # Додаємо початкове місто в кінець списку  
  
  
class AntColony:  
 def \_\_init\_\_(self, distance, cities, n\_ants, alpha, beta, evaporation\_rate, initial\_pheromone):  
 self.distance = distance # Відстані між містами  
 self.cities = cities # Список міст  
 self.pheromone\_map = [initial\_pheromone] \* len(cities) # Початковий рівень феромонів для кожного міста  
 self.n\_ants = n\_ants # Кількість мурах  
 self.alpha = alpha # Параметр альфа для впливу феромонів на вибір наступного міста  
 self.beta = beta # Параметр бета для впливу відстані на вибір наступного міста  
 self.evaporation\_rate = evaporation\_rate # Швидкість випаровування феромонів  
 self.initial\_pheromone = initial\_pheromone # Початковий рівень феромонів  
 self.best\_path = None # Найкращий шлях  
 self.best\_distance = float('inf') # Найменша відстань  
  
 def update\_pheromones(self):  
 for ant in self.ants:  
 for i in range(len(self.cities) - 1):  
 current\_city = ant.visited\_cities[i]  
 next\_city = ant.visited\_cities[i + 1]  
 current\_city\_index = self.cities.index(current\_city)  
 next\_city\_index = self.cities.index(next\_city)  
 self.pheromone\_map[current\_city\_index] \*= (1 - self.evaporation\_rate) # Випаровування феромонів  
 self.pheromone\_map[next\_city\_index] \*= (1 - self.evaporation\_rate) # Випаровування феромонів  
 self.pheromone\_map[current\_city\_index] += self.initial\_pheromone / ant.total\_distance # Додавання феромонів  
 self.pheromone\_map[next\_city\_index] += self.initial\_pheromone / ant.total\_distance # Додавання феромонів  
  
 def run(self, n\_iterations, name\_city):  
 for iteration in range(n\_iterations):  
 self.ants = [Ant(self, name\_city) for \_ in range(self.n\_ants)] # Створення мурах  
 for ant in self.ants:  
 ant.travel()  
  
 if ant.total\_distance < self.best\_distance:  
 self.best\_distance = ant.total\_distance  
 self.best\_path = ant.visited\_cities  
  
 self.update\_pheromones() # Оновлення феромонів  
  
 # print(f"Iteration {iteration + 1}: Best Distance = {self.best\_distance}, Best Path = {self.best\_path}")  
  
# Параметри мурашиного алгоритму  
n\_ants = 40  
alpha = 1  
beta = 2  
evaporation\_rate = 0.5  
initial\_pheromone = 1  
  
# Створення об'єкта класу AntColony та запуск алгоритму  
ant\_colony = AntColony(distance, cities, n\_ants, alpha, beta, evaporation\_rate, initial\_pheromone)  
ant\_colony.run(n\_iterations=70, name\_city='Івано-Франківськ')  
print('\nDistance:', ant\_colony.best\_distance) # Вивід найкращої відстані  
print('\nBest path:', ' -> '.join(ant\_colony.best\_path)) # Вивід найкращого шляху  
  
# Графік  
plt.figure(figsize=(12, 6))  
plt.xticks([i for i in range(len(ant\_colony.best\_path))])  
plt.yticks(range(len(cities)), cities)  
plt.xlabel("Номери міст")  
plt.ylabel("Назви міст")  
plt.title("Маршрут пройдений комівояжером")  
plt.plot([i for i in range(len(ant\_colony.best\_path))], [cities.index(path) for path in ant\_colony.best\_path], ms=10, marker='o', mfc='r')  
plt.grid()  
plt.tight\_layout()  
plt.show()

Результат виконання програми:

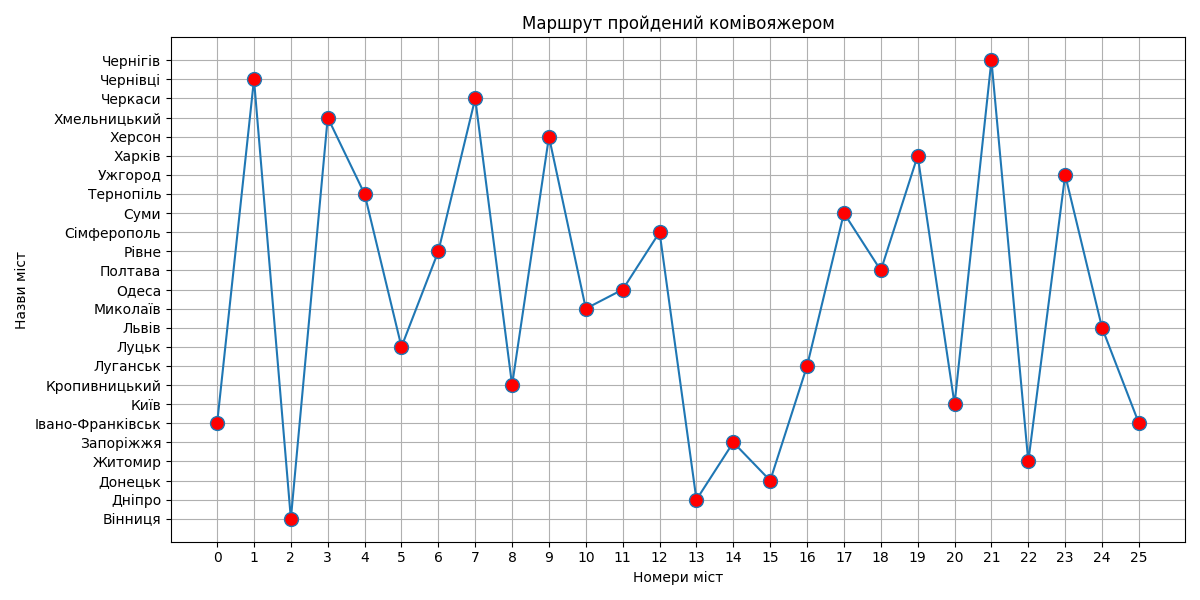


Рис. 2.1.1 – Результат виконання завдання (графік 1).

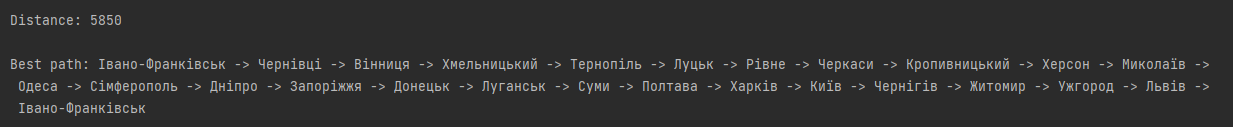


Рис. 2.1.2 – Результат виконання завдання.

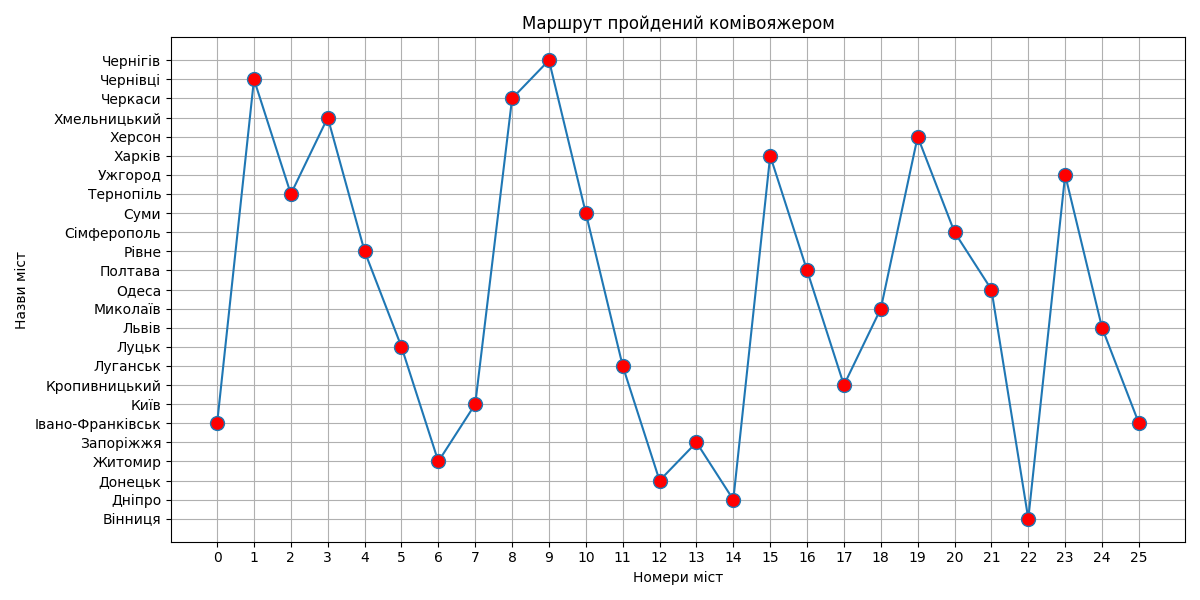


Рис. 2.1.3 – Результат виконання завдання (графік 2).

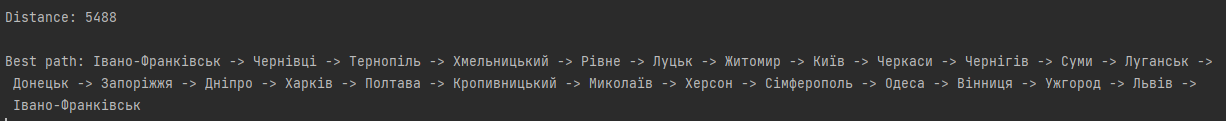


Рис. 2.1.4 – Результат виконання завдання.

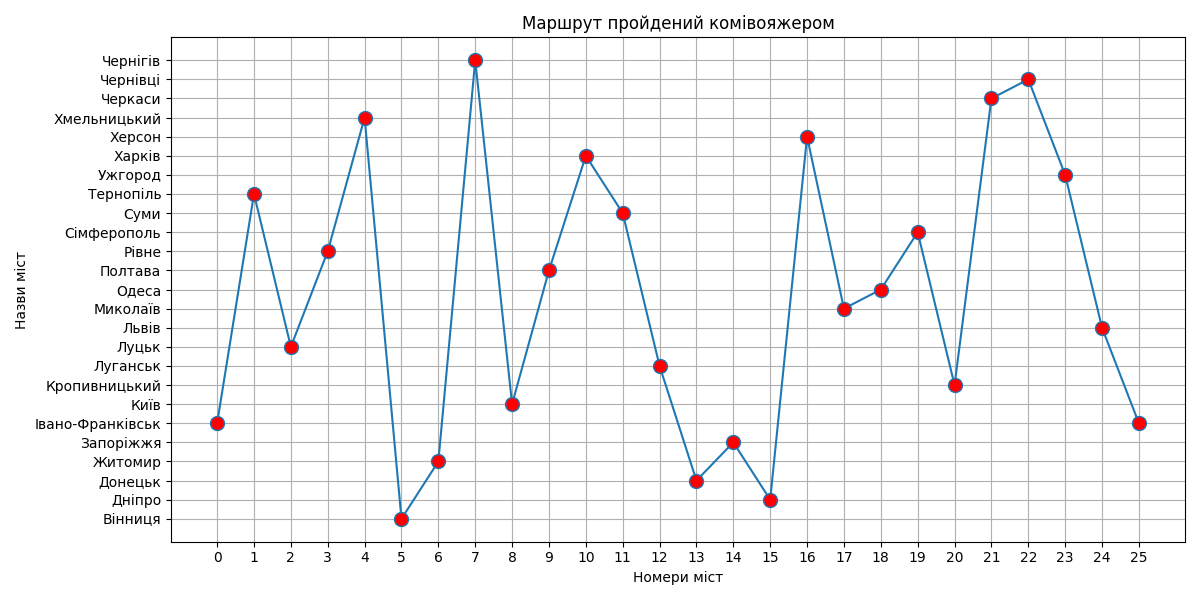
******

Рис. 2.1.5 – Результат виконання завдання (графік 3).

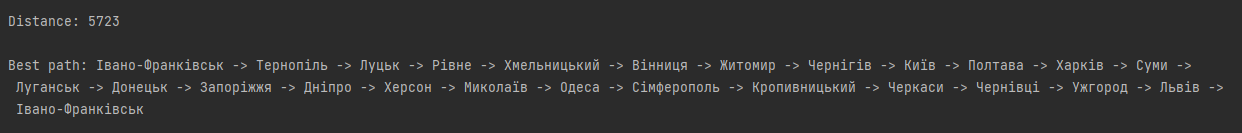


Рис. 2.1.6 – Результат виконання завдання.

***Висновок***

*Алгоритм мурашиних колоній, застосований до задачі комівояжера для подорожі по містах України, показав на кожній ітерації покращення визначеного маршруту. Проходження мурахами різних маршрутів і взаємодія їхніх феромонів призвело до знаходження оптимального маршруту з найменшою відстанню.*

***Посилання на репозиторій:*** <https://github.com/dmytrohrunytskyi/Lab7_AI.git>

***Висновки по лабораторній роботі:*** в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python навчився досліджувати метод мурашиних колоній.