***Лабораторна робота №7***

**Тема:** ДОСЛІДЖЕННЯ МУРАШИНИХ АЛГОРИТМІВ

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

Хід роботи

GitHub репозиторій:
**https://github.com/BashmanivskiyMaxim/Artificial_intelligence_labs**

**Завдання 2.1 (3 варіант - Донецьк): Дослідження мурашиного алгоритму на прикладі рішення задачі комівояжера**

Лістинг програми:

```
import random
import random
import matplotlib.pyplot as plt
import csv
import numpy as np


# Зчитування даних з файлу .csv
def read_distance_matrix(csv_file):
    with open(csv_file, newline="", encoding="mac_cyrillic") as file:
        reader = csv.reader(file)
        rows = list(reader)
    cities = rows[0][1:]  # Перша рядок містить назви міст, відкидаємо перший стовпець
    distance_data = [
        list(map(int, row[1:])) for row in rows[1:]
    ]  # Зчитуємо дані з файла, відкидаємо перший стовпець
    distance_matrix = np.array(distance_data)
    return cities, distance_matrix


csv_file = "Відстань.csv"
csv_file_test1 = "test1.csv"
csv_file_test2 = "test2.csv"
cities, distances = read_distance_matrix(csv_file_test2)
num_cities = len(cities)

# Параметри методу мурашиних колоній
num_ants = 40
max_iterations = 1000
pheromone_evaporation = 0.5
pheromone_deposit = 1.0
alpha = 1.0
beta = 1.0

# Ініціалізація феромонів на шляхах
pheromone = [[1.0] * num_cities for _ in range(num_cities)]
```

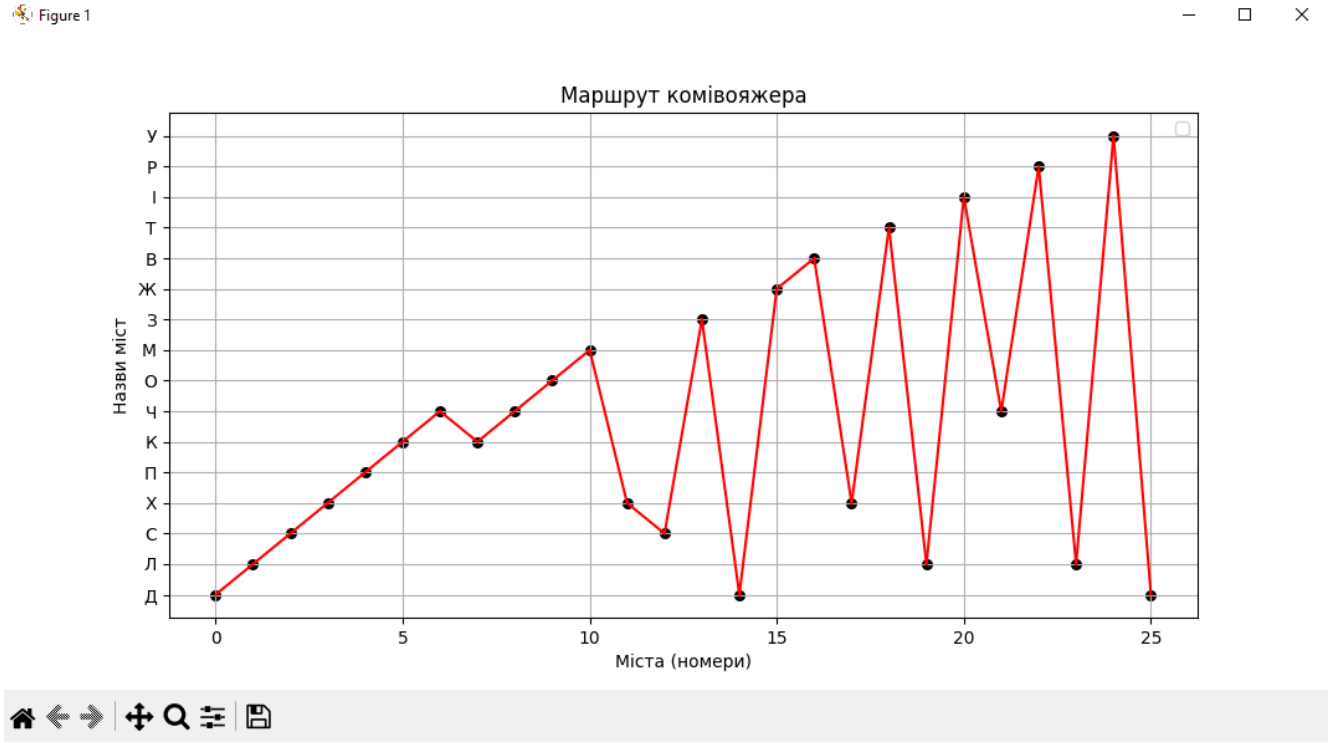
```
# Основна функція для розв'язання задачі комівояжера методом мурашиних колоній
def solve_tsp():
    best_tour = None
    best_distance = float("inf")

    for iteration in range(max_iterations):
        ant_tours = []

        for ant in range(num_ants):
            tour = construct_tour()
            ant_tours.append(tour)

        update_pheromone(ant_tours)

        for tour in ant_tours:
            distance = tour_distanceCalc(tour)
            if distance < best_distance:
                best_distance = distance
                best_tour = tour

    return best_tour, best_distance


# Функція для конструювання маршруту одного мурахи
def construct_tour():
    tour = []
    start_city = random.randint(0, num_cities - 1)
    tour.append(start_city)

    while len(tour) < num_cities:
        next_city = select_next_city(tour, pheromone[tour[-1]])
        tour.append(next_city)
    return tour
# Функція для вибору наступного міста для мурахи з урахуванням феромонів і відстаней
def select_next_city(visited, pheromone_values):
    unvisited_cities = [city for city in range(num_cities) if city not in visited]
    probabilities = [
        calculate_probability(visited[-1], city, pheromone_values)
        for city in unvisited_cities
    ]
    selected_city = random.choices(unvisited_cities, probabilities)[0]

    return selected_city
# Функція для розрахунку ймовірностей для вибору наступного міста
def calculate_probability(current_city, next_city, pheromone_values):
    pheromone = pheromone_values[next_city]
    distance = distances[current_city][next_city]
    probability = (pheromone**alpha) * ((1 / distance) ** beta)
    return probability


# Функція для оновлення рівня феромонів на шляхах після кожної ітерації
def update_pheromone(ant_tours):
    for i in range(num_cities):
        for j in range(num_cities):
```

```
            if i != j:
                pheromone[i][j] *= 1 - pheromone_evaporation

    for tour in ant_tours:
        tour_distance = tour_distanceCalc(tour)
        for i in range(num_cities - 1):
            city1, city2 = tour[i], tour[i + 1]
            pheromone[city1][city2] += pheromone_deposit / tour_distance
# Функція для обчислення відстані подорожі
def tour_distanceCalc(tour):
    distance = 0
    for i in range(len(tour) - 1):
        city1, city2 = tour[i], tour[i + 1]
        distance += distances[city1][city2]
    return distance
# Візуалізація результатів
def visualize_tsp_solution_with_dots(cities, tour):
    # З'єднання міст у порядку маршруту
    for i in range(len(tour) - 1):
        city1 = tour[i]
        city2 = tour[i + 1]
        x1, y1 = i, cities[city1][0]
        x2, y2 = i + 1, cities[city2][0]
        plt.plot([x1, x2], [y1, y2], "r")
    # Додавання чорних точок на кожну точку маршруту
    for i in range(len(tour)):
        x, y = i, cities[tour[i]][0]
        plt.scatter(x, y, color="black", s=30)
    plt.title("Маршрут комівояжера")
    plt.xlabel("Міста (номери)")
    plt.ylabel("Назви міст")
    plt.legend(loc="best")
    plt.grid(True)
    plt.show()
def visualize_tsp_solution(cities, tour):
    plt.plot([cities[i] for i in tour], "o-")
    plt.xlabel("Міста")
    plt.ylabel("Відстань")
    plt.title("Найкращий маршрут задачі комівояжера")
    plt.show()


# Розв'язання задачі комівояжера і виведення результату
best_tour, best_distance = solve_tsp()
best_tour.append(best_tour[0])

print("Найкращий маршрут:", best_tour)
print("Загальна відстань:", best_distance)

# Візуалізуємо найкоротший маршрут
visualize_tsp_solution_with_dots(cities, best_tour)
visualize_tsp_solution(cities, best_tour)
```

Результат виконання програми:

```
Найкращий маршрут: [18, 10, 9, 14, 17, 5, 23, 21, 0, 3, 6, 24, 22, 7, 13, 19, 16, 8, 2, 4, 1, 12, 11, 20, 15, 18]
Загальна відстань: 4229
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored
PS C:\AI_labs\lab7>
```

Отриманий графік:




Графік відображає оптимальний маршрут, який починається з міста Донецьк. Функція `visualize_tsp_solution_with_dots` відображає маршрут у вигляді точок і ліній, які з'єднують міста в порядку маршруту. Кожна точка відповідає одному місту, і лінії показують послідовність відвідування міст. Цей графік нагадує класичне представлення задачі комівояжера.

Під час виконання лабораторної роботи я протестував алгоритм на 3 наборах даних.
**Малий набір даних:** Такий тест допоможе переконатися, що програма правильно знаходить оптимальний маршрут для невеликої задачі.
**Симетрична матриця відстаней:** Де відстань між будь-якою парою міст однакова в обидві сторони. Це дозволить перевірити, чи правильно програма працює зі симетричними даними.
**Великий набір міст:** Це вже більш складна задача і може вимагати більше обчислювальних ресурсів.

Розроблення системи критеріїв порівняння результатів вирішення задачі комівояжера може бути корисним для об'єктивного аналізу і порівняння різних рішень. Ось система критеріїв, які можна використовувати для порівняння результатів вирішення задачі комівояжера:

1. **Загальна відстань:** Відстань, пройдена в оптимальному маршруті, є одним із найважливіших критеріїв. Задача з меншою загальною відстанню вважатиметься кращою з точки зору мінімізації відстані.
2. **Час вирішення:** Час, необхідний для обчислення оптимального маршруту, є важливим критерієм для оцінки продуктивності різних методів розв'язання. Швидше рішення може бути більш ефективним.
3. **Складність обчислень:** Кількість операцій, необхідних для обчислення оптимального маршруту, може бути іншим важливим критерієм. Задача, яка вимагає менше обчислень, може бути ефективнішою з точки зору обчислювальної складності.
4. **Робустність:** Робустність вказує на те, наскільки вразливе рішення до змін у вхідних даних або параметрах. Рішення, яке залишається стабільним при деяких змінах, може бути перевагою.
5. **Параметри методу:** Параметри методу мурашиних колоній, такі як кількість мурах, швидкість випаровування феромонів, параметри впливу феромонів і відстаней, також важливі для порівняння результатів.
6. **Структура відстаней:** Різна структура матриці відстаней (наприклад, розріджена або симетрична) може впливати на результати задачі.

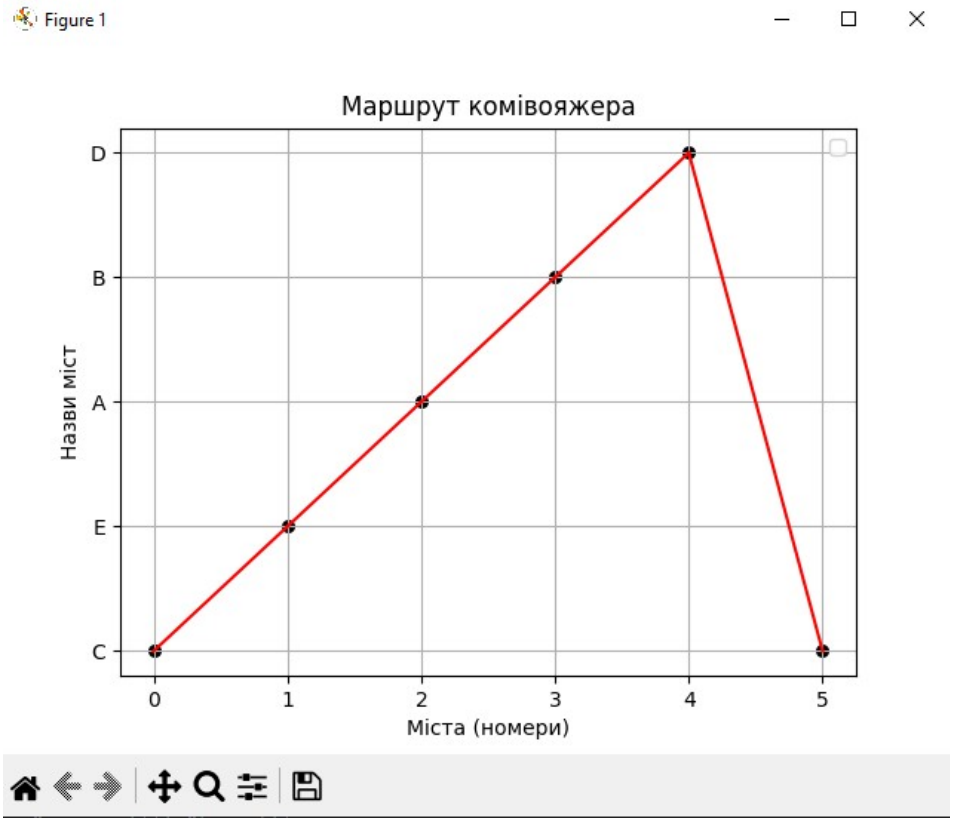
Складена таблиця:

| Складність обчислень | Робустність | Параметри методу | Структура відстаней | Набір даних |
|---|---|---|---|---|
| 35000 операцій | Рішення не вразливе до змін | num_ants = 5<br>max_iterations = 1000<br>pheromone_evaporation = 0.5<br>pheromone_deposit = 1.0<br>alpha = 1.0<br>Beta = 1.0 | розріджена | **Малий набір даних** |
| 35000 операцій | Рішення не вразливе до змін | num_ants = 5<br>max_iterations = 1000<br>pheromone_evaporation = 0.5<br>pheromone_deposit = 1.0<br>alpha = 1.0<br>Beta = 1.0 | симетрична | **Симетрична матриця відстаней** |
| 150000 операцій | Рішення дуже вразливе до змін | num_ants = 25<br>max_iterations = 1000<br>pheromone_evaporation = 0.5 | розріджена | **Великий набір міст** |

pheromone_deposit =
1.0
alpha = 1.0
Beta = 1.0

**Малий набір даних**

Результат виконання програми:




Набір даних:

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | | A | B | C | D | E |
| 2 | A | 0 | 2 | 4 | 5 | 1 |
| 3 | B | 2 | 0 | 3 | 2 | 6 |
| 4 | C | 4 | 3 | 0 | 7 | 2 |
| 5 | D | 5 | 2 | 7 | 0 | 3 |
| 6 | E | 1 | 6 | 2 | 3 | 0 |

```
Найкращий маршрут: [2, 4, 0, 1, 3, 2]
Загальна відстань: 7
```

Висновок: Програма коректно працює на мали наборах даних. З кожним запуском програми найкращий маршрут залишається сталим. Зміна параметрів взагалі не впливає на результат найкращого маршруту.

**Симетрична матриця відстаней:**

Результат виконання програми:


```
Найкращий маршрут: [2, 0, 1, 3, 2]
Загальна відстань: 55
```

```
Найкращий маршрут: [2, 1, 0, 3, 2]
Загальна відстань: 55
```

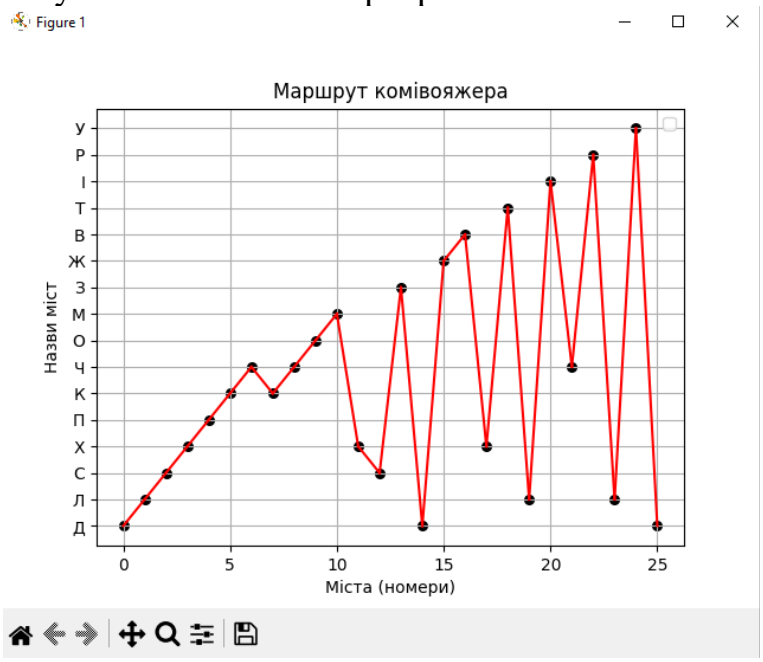
Набір даних:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | | A | B | C | D |
| 2 | A | 0 | 10 | 15 | 20 |
| 3 | B | 10 | 0 | 25 | 30 |
| 4 | C | 15 | 25 | 0 | 35 |
| 5 | D | 20 | 30 | 35 | 0 |

Висновок: По результатам видно що найкращий маршрут змінюється при кожному запуску програми. Це відбувається через те що відстані між точками є симетричними, “мурахи” при виборі шляху можуть обирати різний шлях, кінцевий результат не зміниться і залишиться — 55.

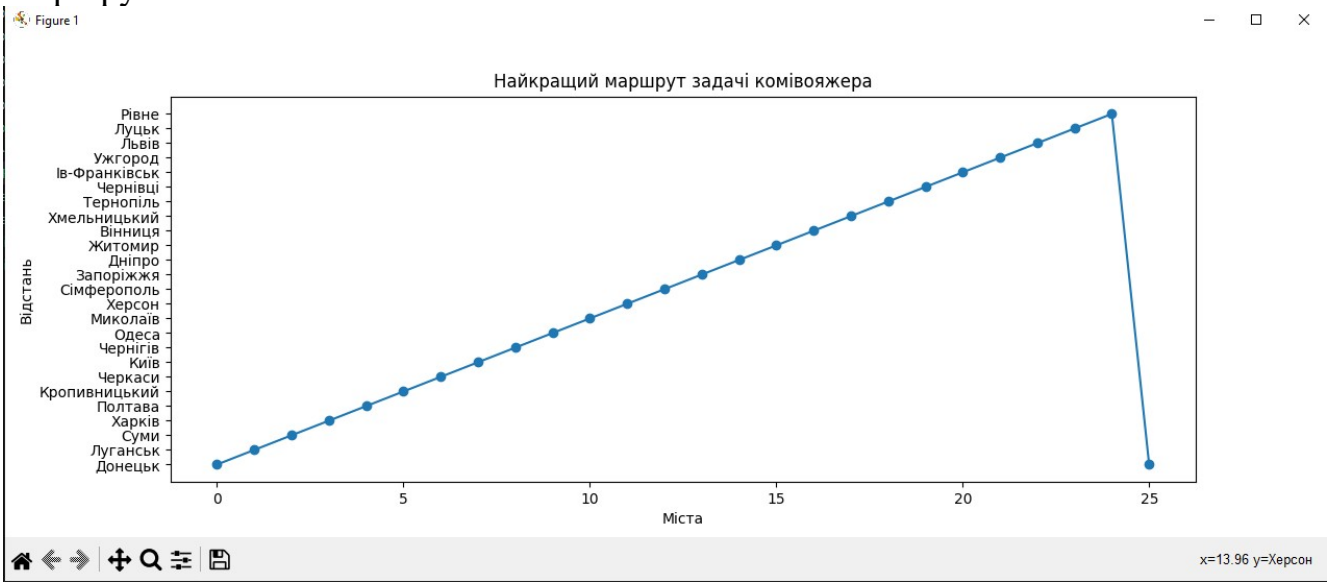
**Великий набір міст**

Результат виконання програми:




```
Найкращий маршрут: [2, 8, 16, 19, 13, 1, 4, 20, 11, 12, 15, 7, 22, 24, 6, 3, 0, 21, 14, 9, 10, 17, 23, 5, 18, 2]
Загальна відстань: 4562
```

Маршрут:




Висновок: У цьому випадку програма працює з великим об’ємом даних. В цьому випадку параметри методу вагомо впливають на результат найкращого маршруту.

Висновок: код реалізує розв'язання задачі комівояжера за допомогою алгоритму мурашиних колоній. Алгоритм мурашиних колоній - це метаевристичний алгоритм, який намагається знайти найкоротший шлях між набором міст, якого потрібно відвідати один раз і повернутися в початкове місто.
Основні кроки алгоритму включають в себе:

1. Ініціалізація феромонів на всіх можливих шляхах між містами.
2. Для кожної ітерації алгоритму:

   a. Декілька мурах конструюють свої маршрути, вибираючи наступне місто з урахуванням феромонів та відстаней.

   b. Феромон оновлюється на кожному пройденому маршруті.

   c. Зберігається найкращий знайдений маршрут.
3. Виведення найкращого маршруту та відстані.

Загалом, алгоритм мурашиних колоній - це ефективний спосіб вирішення задачі комівояжера, особливо для великих наборів міст. Проте код може бути оптимізований для зменшення часу виконання, а результати профілювання можуть бути корисні для цієї мети.