**Практична робота № 7**

**Варіант 13**

**Дослідження мурашиних алгоритмів**

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

**Хід роботи:**

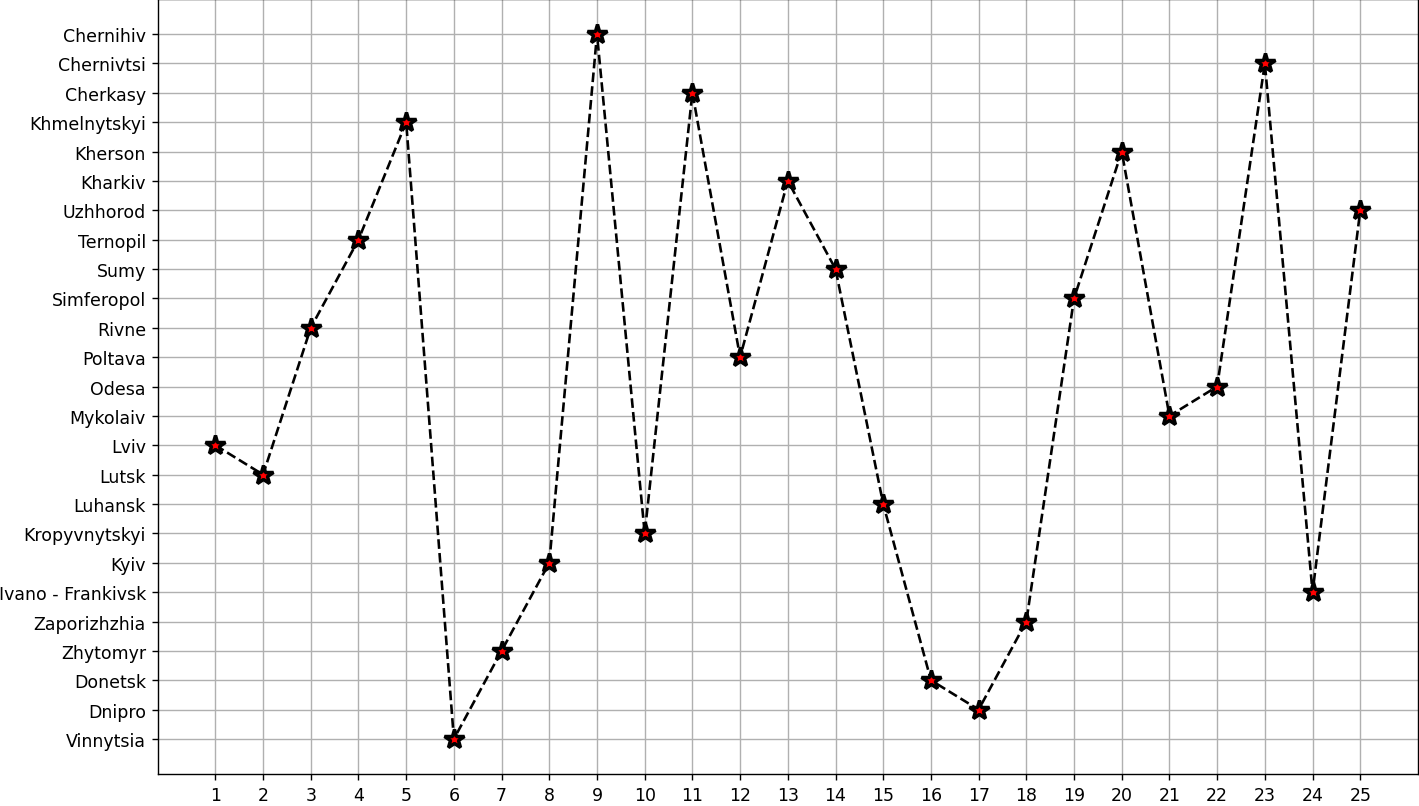
**Завдання 7.1:** Завдання. Дослідження мурашиного алгоритму на прикладі рішення задачі комівояжера

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

Фрагмент лістингу:

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)]  
  
 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]  
  
 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  
  
 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  
  
 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  
  
 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  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 cityMap = CityMap(distance, len(distance[0]))  
 colony = Colony(len(distance[0]))  
 result = colony.find\_route(cityMap, 10)  
 print(f"The shortest path obtained: {result[0]} km")  
  
 cityRoutes = "Received route: "  
 for i in result[1]:  
 cityRoutes += cities[i]  
 if i != result[1][-1]:  
 cityRoutes += "->"  
 print(cityRoutes)  
  
 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("City Numbers")  
 plt.ylabel("City Names")  
 plt.title("Ant's optimal route")  
 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()



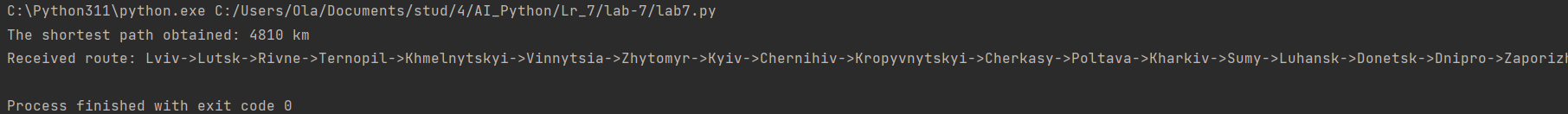


Рис.7.1. Результат виконання

[**https://github.com/avrorilka/AI\_Python**](https://github.com/avrorilka/AI_Python)

**Висновки:** в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python дослідили метод мурашиних колоній.

Метою цього проекту було навчитися досліджувати метод мурашиних колоній за допомогою спеціалізованих бібліотек і мови програмування Python. Завдяки дослідженням і дослідженням вдалося успішно розробити програму, яка використовує мурашиний алгоритм для вирішення проблеми комівояжера.

Мурашиний алгоритм є потужним інструментом для вирішення задач оптимізації, і його можна використовувати в багатьох різних програмах. При розробці програми в мережі Інтернет здійснювався пошук можливих реалізацій необхідних методів на основі наведеного прикладу мовою Matlab.