ЛАБОРАТОРНА РОБОТА №7

ДОСЛІДЖЕННЯ МУРАШИНИХ АЛГОРИТМІВ

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

Хід роботи:

GitHub: https://github.com/ipz201svo/AI

Завдання 1: Дослідження мурашиного алгоритму на прикладі рішення задачі комівояжера.

```
import math
import random
import matplotlib.pyplot as pl

# file with input data
from data import (
    Ukraine_dis,
    Ukraine_map,
    manager_map,
    manager_dis,
    mom_map,
    mom_dis,
    student_map,
    student_dis,
)

# variables
variant = 17  # номер варіанта за журналом
alpha = 1.0
beta = 5.0
rho = 0.5  # P
Q = 10
iterations = 10000

mapping = Ukraine_map  # cities names
object_count = len(mapping)
distances = Ukraine_dis  # distance
y = list()  # graphic for best way
```

					ДУ «Житомирська політехніка».23.121.17.000 – Лр7		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Скаковський В.О.			Звіт з лабораторної роботи №7	Літ.	Арк. 1 / Аркушів 10
Перевір.		Голенко М.Ю.					
Керівник						ФІКТ Гр. ІПЗ-20-1	
Н. контр.							
Зав. каф.							

```
class Ant:
    def __init__(self, parent):
        self.parent = parent
        self.position = variant - 1
        self.start = self.position
        self.totalDist = 0.0
        self.tList = [self.position]
        self.myPheromone = []

    def travel(self):
        if len(self.tList) == object_count:
            self.tList.remove(self.start)

        p_array = [0 for _ in range(object_count)]
        summa = 0
        # formula for counting probability of visiting the following points
        for i in range(object_count):
            if i not in self.tList:
                summa += (self.parent.pheromones[self.position][i] ** alpha) * (
                    self.parent.visibility[self.position][i] ** beta
                )
        for i in range(object_count):
            if i not in self.tList:
                try:
                    p_array[i] = (
                        (self.parent.pheromones[self.position][i] ** alpha)
                        * (self.parent.visibility[self.position][i] ** beta)
                        / summa
                    )
                except:
                    pass
        revers = list(
            filter(lambda p: p not in self.tList, [i for i in range(object_count)])
        )  # check place if city been used for a once
        revers.reverse()
        next_city = revers[0]
        winner_num = random.random() * sum(p_array)  # choosing next point using
random
        for i, probability in enumerate(p_array):
            winner_num -= probability
            if winner_num <= 0:
                next_city = i
                break
        newd = distances[self.position][next_city]  # writting in a next city
        self.totalDist += newd if newd > 0 and next_city not in self.tList else
math.inf
        self.tList.append(next_city)
        self.position = next_city

    def update_ways(self):  # refreshing pheromone for visiting border
        self.myPheromone = [
            [0.0 for _ in range(object_count)] for __ in range(object_count)
        ]
        for i in range(1, len(self.tList)):
```

```
            k = self.tList[i - 1]
            j = self.tList[i]
            self.myPheromone[k][j] = Q / self.totalDist


class Colony:
    smallestCost = math.inf
    optimal_way = []
    ants = []
    pheromones = None
    visibility = None

    def __init__(self):
        self.pheromones = [
            [1 / (object_count * object_count) for _ in range(object_count)]
            for __ in range(object_count)
        ]  # initial amount of pheromone for the start
        self.visibility = [
            [0 if i == j else 1 / distances[i][j] for i in range(object_count)]
            for j in range(object_count)
        ]  # this is an inverse distance

    def do_main(self):
        self.smallestCost = math.inf
        self.optimal_way = []

        for t in range(iterations):  # main cycle
            self.reload_ants()
            self.move_ants()
            self.update_ways()
            y.append(self.smallestCost)  # graphics data
        return self.smallestCost, self.optimal_way

    def move_ants(self):
        for ant in self.ants:
            for i in range(object_count):
                ant.travel()

            if ant.totalDist < self.smallestCost:  # determine the optimal path
                self.smallestCost = ant.totalDist
                self.optimal_way = [ant.tList[-1]] + ant.tList
            ant.update_ways()

    def update_ways(self):  # evaporate and add pheromones
        for i, row in enumerate(self.pheromones):
            for j, col in enumerate(row):
                self.pheromones[i][j] *= rho
                for ant in self.ants:
                    self.pheromones[i][j] += ant.myPheromone[i][j]

    def reload_ants(self):  # updating agents
        self.ants = [Ant(self) for _ in range(round(object_count * 0.8))]


newLineSymbol = "\n-> "
```

```
dist, path = Colony().do_main()
print(
    f"Оптимальний результат: {dist}, \nШЛЯХ:\n{newLineSymbol.join(mapping[i] for i
in path)}"
)
pl.figure()
pl.plot(
    [x for x in range(object_count + 1)],
    path,
)
for i, txt in enumerate(mapping):
    pl.annotate(txt, (path.index(i), i))
pl.show()
```

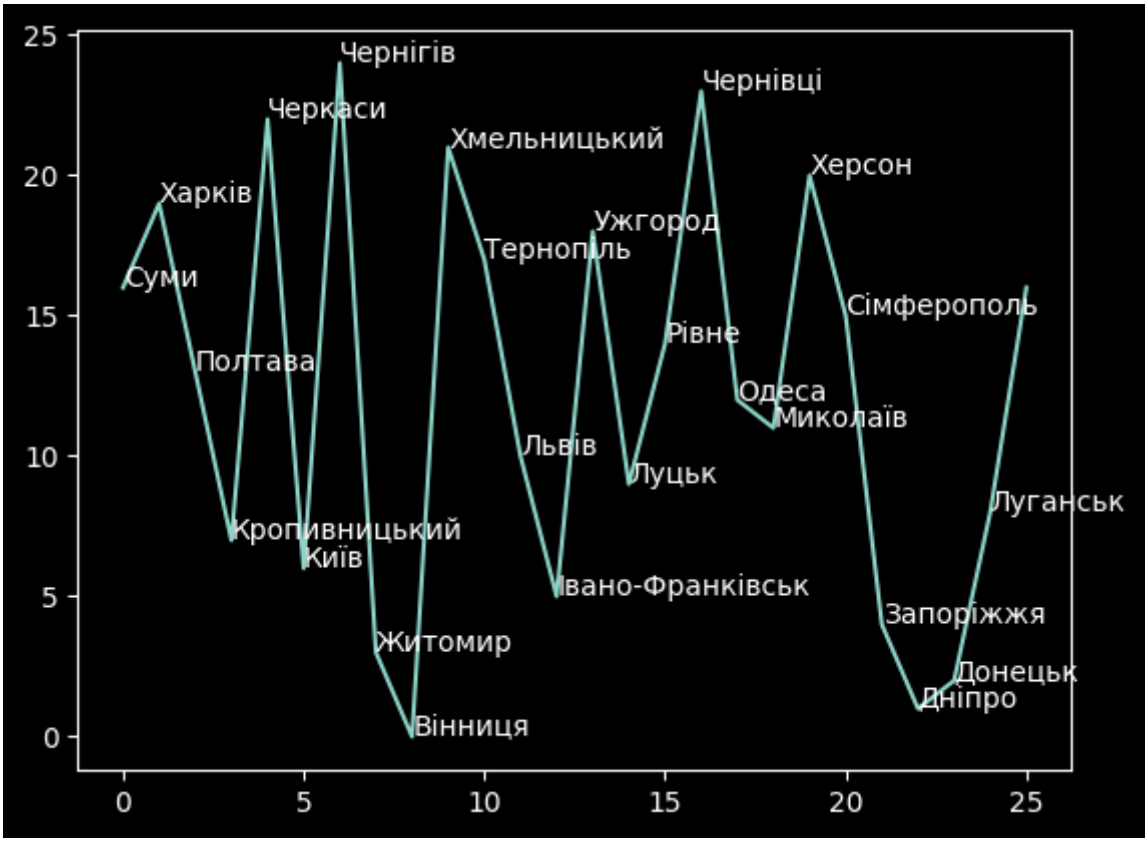

Рис. 1.1 – Результат виконання програми

```
Оптимальний результат: 5033.0,
ШЛЯХ:
Суми
-> Харків
-> Полтава
-> Кропивницький
-> Черкаси
-> Київ
-> Чернігів
-> Житомир
-> Вінниця
-> Хмельницький
-> Тернопіль
-> Львів
-> Івано-Франківськ
-> Ужгород
-> Луцьк
-> Рівне
-> Чернівці
-> Одеса
-> Миколаїв
-> Херсон
-> Сімферополь
-> Запоріжжя
-> Дніпро
-> Донецьк
-> Луганськ
-> Суми
```

Рис. 1.2 – Результат виконання програми

Тестування на інших задачах:

Задача 1: Менеджеру необхідно виконати велику кількість завдань по проекту. Потрібно прорахувати найменший шлях, аби встигнути виконати поставлені задачі до кінця робочого дня. Варіант обрано 9 як половина реального варіанту 17, через обмеження в вхідних даних.

```python
# 2 task
manager_map = [
    "Daily SCRUM",
    "Перевірка бюджета",
    "Дзвінок із клієнтом",
    "Оновлення планування",
    "Дзвінок із керівництвом",
    "DEMO",
    "Написання e-mail",
    "Ескалація проблеми",
    "Трекінг часу",
]
manager_dis = [
    [0, 1, 2, 3, 4, 5, 6, 7, 8],
    [8, 0, 1, 2, 3, 4, 5, 6, 7],
    [7, 8, 0, 1, 2, 3, 4, 5, 6],
    [6, 7, 8, 0, 1, 2, 3, 4, 5],
    [5, 6, 7, 8, 0, 1, 2, 3, 4],
    [4, 5, 6, 7, 8, 0, 1, 2, 3],
    [3, 4, 5, 6, 7, 8, 0, 1, 2],
    [2, 3, 4, 5, 6, 7, 8, 0, 1],
    [1, 2, 3, 4, 5, 6, 7, 8, 0],
]
```

Рис. 1.3 – Результат виконання програми

Рис. 1.4 – Результат виконання програми

```
Оптимальний результат: 63.0,
ШЛЯХ:
Трекінг часу
-> Ескалація проблеми
-> Написання e-mail
-> Дзвінок із керівництвом
-> Оновлення планування
-> Дзвінок із клієнтом
-> Перевірка бюджета
-> Daily SCRUM
-> DEMO
-> Трекінг часу
```

Рис. 1.5 – Результат виконання програми

Задача 2: Є список покупок, проте усі предмети продаються в різних магазинах. Знайти найшвидший шлях щоб купити усі речі. Як змінну варіанта було обрано 5.

```python
# 3 task
list_map = [
    "Продукти",
    "Побутова хімія",
    "Інструменти",
    "Одяг",
    "Навушники",
]
list_dis = [
    [0, 4, 1, 3, 9],
    [9, 0, 3, 1, 4],
    [1, 3, 0, 9, 5],
    [7, 6, 2, 0, 1],
    [12, 8, 3, 10, 0],
]
```

Рис. 1.6 – Результат виконання програми

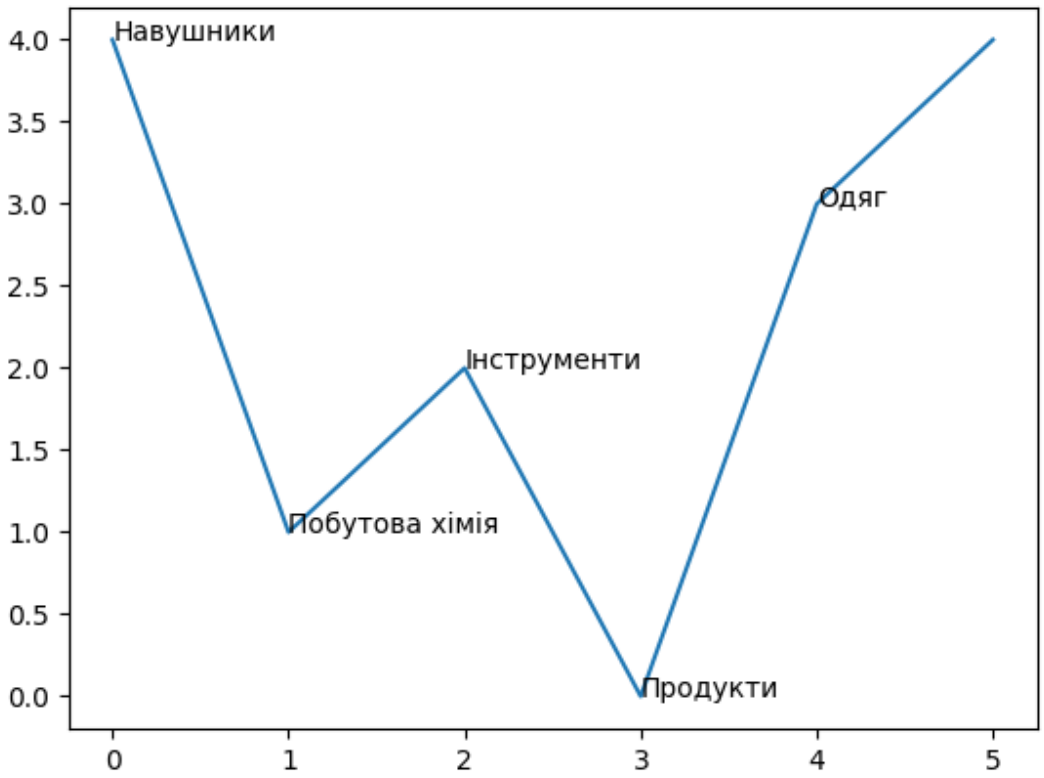

Рис. 1.7 – Результат виконання програми

```
Оптимальний результат: 16.0,
ШЛЯХ:
Навушники
-> Побутова хімія
-> Інструменти
-> Продукти
-> Одяг
-> Навушники
```

Рис. 1.8 – Результат виконання програми

Задача 3: На зимовій риболовлі зроблено декілька лунок. Рибалці потрібно обійти їх всіх за найкоротший проміжок часу. Як змінну варіанту обрано – 7.

```python
# 4 task
fish_map = [
    "Лунка 1",
    "Лінка 2",
    "Лунка 3",
    "Лунка 4",
    "Лунка 5",
    "Лунка 6",
    "Лунка 7",
]
fish_dis = [
    [0, 5, 3, 16, 12, 8, 10],
    [10, 0, 5, 3, 16, 12, 8],
    [8, 10, 0, 5, 3, 16, 12],
    [12, 8, 10, 0, 5, 3, 16],
    [16, 12, 8, 10, 0, 5, 3],
    [3, 16, 12, 8, 10, 0, 5],
    [5, 3, 16, 12, 8, 10, 0],
]
```

Рис. 1.9 – Результат виконання програми

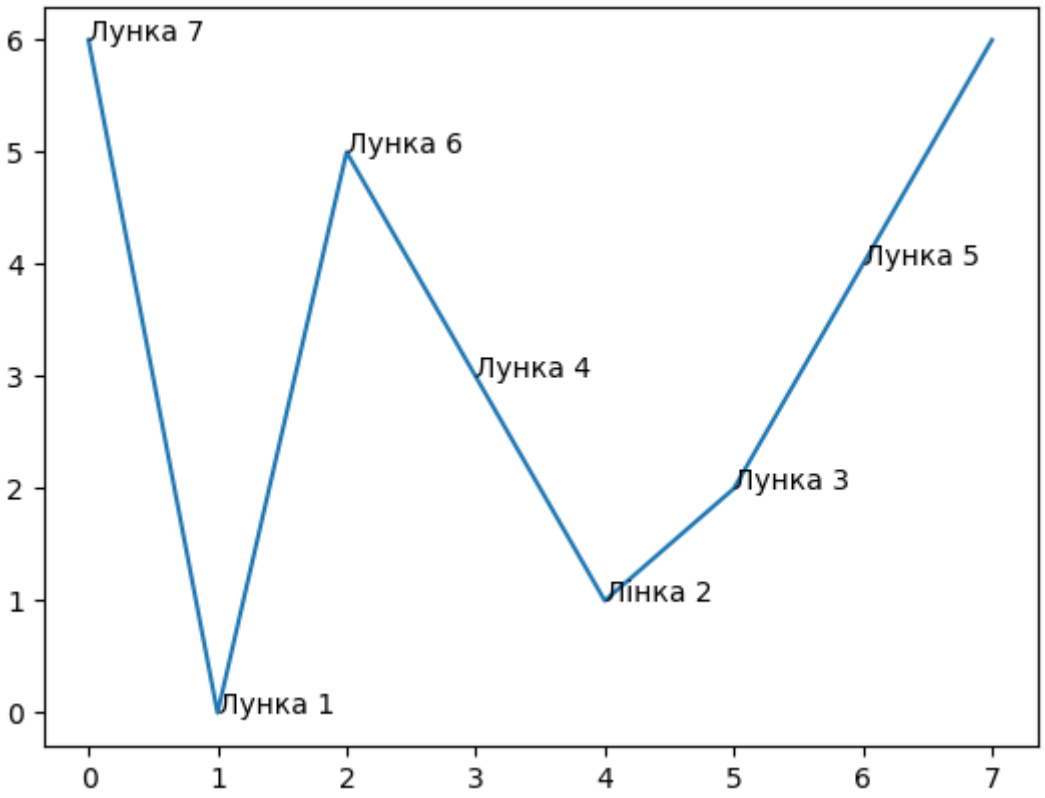

Рис. 1.10 – Результат виконання програми

```
Оптимальний результат: 40.0,
ШЛЯХ:
Лунка 7
-> Лунка 1
-> Лунка 6
-> Лунка 4
-> Лінка 2
-> Лунка 3
-> Лунка 5
-> Лунка 7
```

Рис. 1.11 – Результат виконання програми

Висновок: під час виконання лабораторної роботи, використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити метод мурашиних колоній.