ЛАБОРАТОРНА РОБОТА 7

ДОСЛІДЖЕННЯ МУРАШИНИХ АЛГОРИТМІВ

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

Хід роботи

Завдання 2.1. Дослідження мурашиного алгоритму на прикладі рішення задачі комівояжера

Лістинг програми:

```
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 = 14 # 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


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
```

					ДУ «Житомирська політехніка».22.121.14.000 – Лр7		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Сірач А.С.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів 8
Перевір.		Філіпов В.О.					
Керівник						ФІКТ Гр. ІПЗ-19-3[2]	
Н. контр.							
Зав. каф.							

```
        #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. 7. 1 Результат виконання програми

```
Оптимальний результат: 5346.0,
 ШЛЯХ: Полтава
-> Харків
-> Суми
-> Луганськ
-> Донецьк
-> Запоріжжя
-> Дніпро
-> Кропивницький
-> Черкаси
-> Київ
-> Житомир
-> Вінниця
-> Хмельницький
-> Тернопіль
-> Івано-Франківськ
-> Чернівці
-> Ужгород
-> Львів
-> Луцьк
-> Рівне
-> Чернігів
-> Одеса
-> Миколаїв
-> Херсон
-> Сімферополь
-> Полтава
```

Рис. 7. 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 7 як половина реального варіанту 14, через обмеження в вхідних даних.

Лістинг програми:

```
# 2 task
manager_map = ["Дзвінок клієнту", "Зідзвон", "Зум-міт", "Документація",
"Тестування", "Здача проекту", "Звітність"]
manager_dis = [
    [0, 1, 2, 3, 4, 5, 6],
    [6, 0, 1, 2, 3, 4, 5],
    [5, 6, 0, 1, 2, 3, 4],
    [4, 5, 6, 0, 1, 2, 3],
    [3, 4, 5, 6, 0, 1, 2],
    [2, 3, 4, 5, 6, 0, 1],
    [1, 2, 3, 4, 5, 6, 0],
]
```

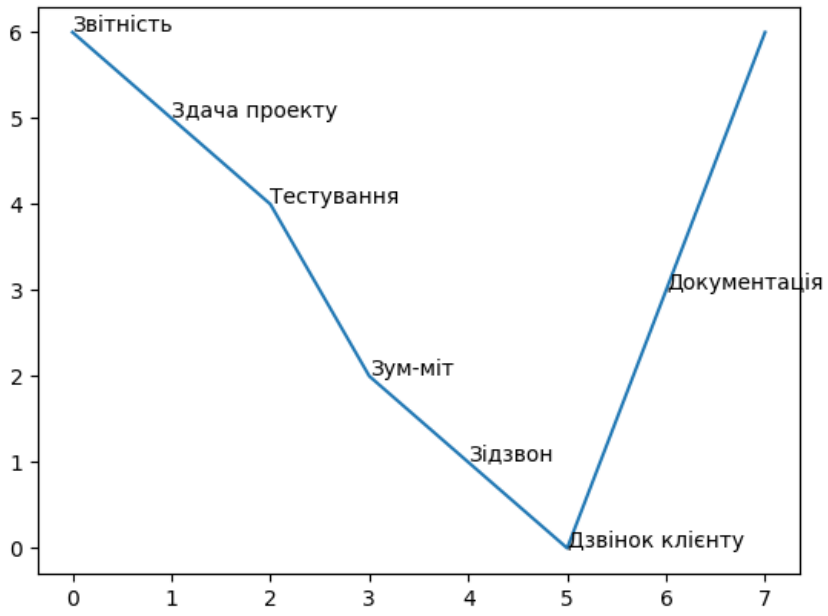

Рис. 7. 3 Результат виконання програми

```
Оптимальний результат: 35.0,
ШЛЯХ:
Звітність
-> Здача проекту
-> Тестування
-> Зум-міт
-> Зідзвон
-> Дзвінок клієнту
-> Документація
-> Звітність
```

Рис. 7. 4 Результат виконання програми

Задача 2: Мама приїжджає в гості, а вдома неприбрано. Знайти найшвидший спосіб аби встигнути привести доміку до охайного ладу та встигнути приготувати вечерю. Як змінну для варіанту обрано – 2.

Лістинг програми:

```
mom_map = ["Позамітати", "Вимити підлогу", "Провітрити", "Витерти пил",
"Приготувати вечерю"]
mom_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],
]
```

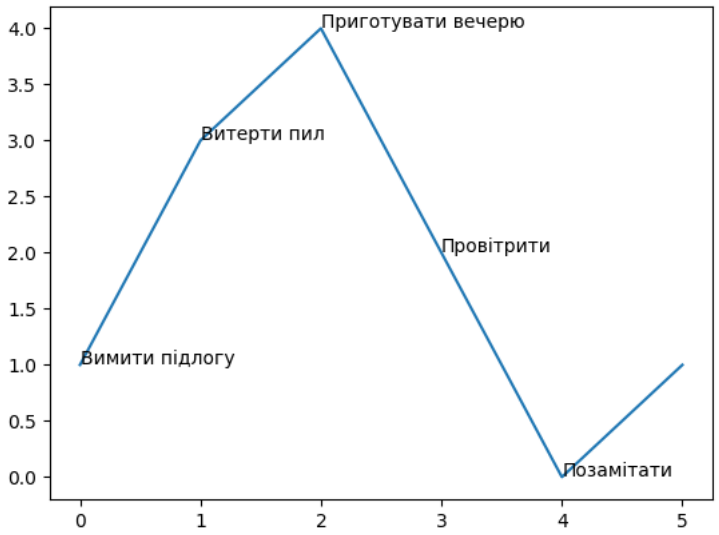

Рис. 7. 5 Результат виконання програми

```
Оптимальний результат: 10.0,
ШЛЯХ:
Вимити підлогу
-> Витерти пил
-> Приготувати вечерю
-> Провітрити
-> Позамітати
-> Вимити підлогу
```

Рис. 7. 6 Результат виконання програми

Задача 3: Студенту необхідно в найбільш швидкий спосіб закрити всі дисципліни до кінця сесії, яка вже через 2 тижні. Як змінну варіанту обрано – 7.

Лістинг програми:

```
student_map = ["США", "Лінукс", "Реакт", "Моделювання", "Хмарні", "Іноземна",
"Політологія"]
student_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],
]
```

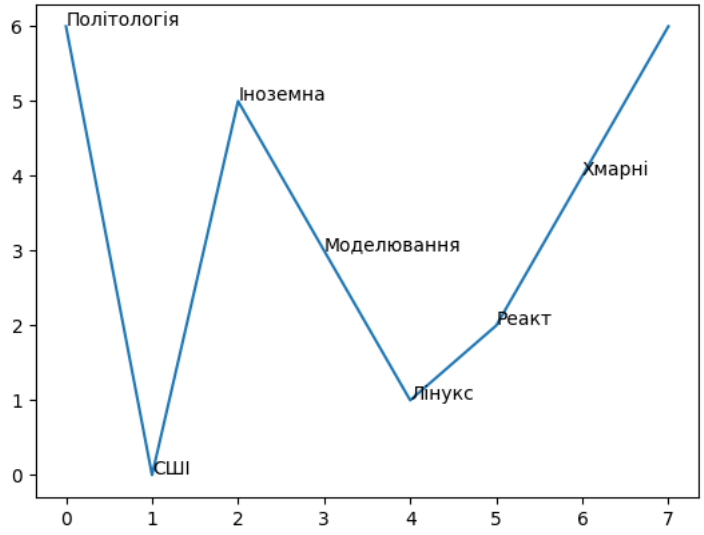

Рис. 7. 7 Результат виконання програми

```
Оптимальний результат: 40.0,
ШЛЯХ:
Політологія
-> США
-> Іноземна
-> Моделювання
-> Лінукс
-> Реакт
-> Хмарні
-> Політологія
```

Рис. 7. 8 Результат виконання програми

Критерії	Подорож по Україні	Задача 1	Задача 2	Задача 3
Швидкість виконання	Середня	Дуже швидко	Дуже швидко	Дуже швидко
Ефективність	Середня	Висока	Висока	Висока
Кількість агентів	20	7	5	7
Кількість ітерацій	1000	1	1	1

Висновок: під час виконання лабораторної роботи було досліджено метод мурашиних колоній використовуючи спеціалізовані бібліотеки і мову програмування Python.

Посилання на GitHub: https://github.com/annasirach/AI_IPZ193_Sirach