Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 4 з дисципліни
«Проектування алгоритмів»

„Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.1”

Виконав(ла) *ІП-14 Бабіч Денис* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Головченко Максим Миколайович* _____
(прізвище, ім'я, по батькові)

Київ 2022

Зміст

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи формалізації метаевристичних алгоритмів і вирішення типових задач з їхньою допомогою.

2 Завдання

Згідно варіанту, розробити алгоритм вирішення задачі і виконати його програмну реалізацію на будь-якій мові програмування.

Задача, алгоритм і його параметри наведені в таблиці 2.1.

Зафіксувати якість отриманого розв'язку (значення цільової функції) після кожних 20 ітерацій до 1000 і побудувати графік залежності якості розв'язку від числа ітерацій.

Зробити узагальнений висновок.

Таблиця 2.1 – Варіанти алгоритмів

№	**Задача і алгоритм**
2	Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0{,}4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 30, починають маршрут в різних випадкових вершинах).

3 Виконання

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

Program.cs

```
using TravellingSalesmanProblem;

namespace AntColony
{
    internal class AntColonyProgram
    {
        private static void Main()
        {
            System.Console.InputEncoding = System.Text.Encoding.Unicode;
            System.Console.OutputEncoding = System.Text.Encoding.Unicode;

            int a = InputInt("Введіть значення АЛЬФА. Стандарте значення згідно до варіанту - 2. Введіть число: ", 0, Int32.MaxValue);
            int b = InputInt("Введіть значення БЕТА. Стандарте значення згідно до варіанту - 4. Введіть число: ", 0, Int32.MaxValue);
            float p = InputFloat("Введіть значення випаровування ферамону. Стандарте значення згідно до варіанту - 0.4. Введіть число: ", 0.0f, 1.0f);

            int numberOfIterations = InputInt("Введіть кількість ітерцій: ", 0, Int32.MaxValue);
            int numberOfAnts = InputInt("Введіть кількість мурах. Стандарте значення згідно до варіанту - 30. Введіть число: ", 0, Int32.MaxValue);
            int numberOfCities = InputInt("Введіть кількість міст. Стандарте значення згідно до варіанту - 50. Введіть число: ", 0, Int32.MaxValue);

            int MinDistance = InputInt("Введіть мінімальну відстань між містами. Стандарте значення згідно до варіанту - 5. Введіть число: ", 0, Int32.MaxValue);
            int MaxDistance = InputInt("Введіть максимальну відстань між містами. Стандарте значення згідно до варіанту - 50. Введіть число: ", 0, Int32.MaxValue);

            while (MinDistance > MaxDistance)
            {
                Console.WriteLine("Мінімальна відстань має бути менше максимальної!");
                MinDistance = InputInt("Введіть мінімальну відстань між містами. Стандарте значення згідно до варіанту - 5. Введіть число: ", 0, Int32.MaxValue);
                MaxDistance = InputInt("Введіть максимальну відстань між містами. Стандарте значення згідно до варіанту - 50. Введіть число: ", 0, Int32.MaxValue);
            }

            Graph graph = new Graph(numberOfCities);
            graph.GenerateDistanceMatrix(MinDistance, MaxDistance);
            graph.GeneratePheromoneMatrix(0.0f, 0.1f);

            Ant[] ants = new Ant[numberOfAnts];
            Ant.AllCities = graph.GetAllCities();

            for (int i = 0; i < numberOfAnts; ++i)
                ants[i] = new Ant(graph, Ant.GetRandomCity());

            List<int> distances = new List<int>();

            for (int i = 0; i < numberOfIterations; ++i)
```

```
            {
                Console.Write($"{i + 1}. ");

                foreach (Ant ant in ants)
                    ant.Move(a, b);

                distances.Add(ants.OrderBy(ant =>
ant.TravelledDistance).First().TravelledDistance);
                graph.UpdatePheromoneMatrix(p, distances[distances.Count - 1], ants);
                Console.WriteLine($"Мінімальна відстань поточної ітерації:
{distances[distances.Count - 1]}.");
            }

            Console.WriteLine($"\nМінімальна відстань за результатом всіх ітерацій:
{distances.OrderBy(distance => distance).First()}");
            graph.PrintPath();
        }

        private static int InputInt(string text, int min, int max)
        {
            int value;

            do
            {
                Console.Write(text);

                if (Int32.TryParse(Console.ReadLine(), out value))
                {
                    if (value <= min)
                    {
                        Console.WriteLine($"Введення значення має бути більше
{min}!");
                        continue;
                    }

                    if (value >= max)
                    {
                        Console.WriteLine($"Введення значення має бути менше {max}!");
                        continue;
                    }

                    return value;
                }

                Console.WriteLine("Ввена величина має бути числом!");

            } while (true);
        }

        private static float InputFloat(string text, float min, float max)
        {
            float value;

            do
            {
                Console.Write(text);

                if (Single.TryParse(Console.ReadLine(), out value))
                {
                    if (value <= min)
                    {
                        Console.WriteLine($"Введення значення має бути більше
{min}!");
```

```
                    continue;
                }

                if (value >= max)
                {
                    Console.WriteLine($"Введення значення має бути менше {max}!");
                    continue;
                }

                return value;
            }

            Console.WriteLine("Ввена величина має бути числом!");

        } while (true);
    }
  }
}
```

Ant.cs

```
namespace TravellingSalesmanProblem
{
    public sealed class Ant
    {
        public static List<int> AllCities { get; set; } = new List<int>();

        private Graph graph;
        private int nextCity;
        private int currentCity;
        private List<int> citiesToVisit;

        public int StartCity { get; }

        public int TravelledDistance { get; private set; }

        public List<int> VisitedCities { get; private set; }

        public Ant(Graph graph, int startCity)
        {
            this.graph = graph;
            this.StartCity = startCity;
            this.currentCity = startCity;
            this.VisitedCities = new List<int>();
            citiesToVisit = new List<int>();
        }

        public void Move(int a, int b)
        {
            this.VisitedCities.Clear();
            this.TravelledDistance = 0;
            this.citiesToVisit = graph.GetAllCities();

            float denominator;
            var probabilities = new[] { new { city = -1, probability = 0.0f }
}.ToList();

            while (true)
            {
                denominator = 0.0f;
                probabilities.Clear();
                this.VisitedCities.Add(currentCity);
                this.citiesToVisit.Remove(currentCity);
```

```
                for (int i = 0, length = this.citiesToVisit.Count; i < length; ++i)
                        denominator += MathF.Pow(graph.PheromoneMatrix[currentCity,
citiesToVisit[i]], a) * MathF.Pow(1.0f / graph.DistanceMatrix[currentCity,
citiesToVisit[i]], b);

                foreach (int city in this.citiesToVisit)
                    probabilities.Add(new { city = city, probability =
GetProbability(currentCity, city, denominator) });

                if (probabilities.Count > 1)
                    probabilities = probabilities.OrderByDescending(record =>
record.probability).ToList();

                if (this.citiesToVisit.Count < 1)
                {
                    this.TravelledDistance += graph.DistanceMatrix[currentCity,
StartCity];
                    this.VisitedCities.Add(this.StartCity);
                    break;
                }

                nextCity = probabilities.First().city;
                this.TravelledDistance += graph.DistanceMatrix[currentCity, nextCity];
                currentCity = nextCity;
            }

            float GetProbability(int currentCity, int nextCity, float denominator) =>
                MathF.Pow(graph.PheromoneMatrix[currentCity, nextCity], a) *
MathF.Pow(1.0f / graph.DistanceMatrix[currentCity, nextCity], b) / denominator;
        }

        public static int GetRandomCity()
        {
            int city = AllCities[new Random().Next(0, AllCities.Count)];
            AllCities.Remove(city);
            return city;
        }
    }
}
```

Graph.cs

```
namespace TravellingSalesmanProblem
{
    public sealed class Graph
    {
        public int NumberOfCities { get; }

        public int[,] DistanceMatrix { get; private set; }

        public float[,] PheromoneMatrix { get; set; }

        public Graph(int numberOfCities)
        {
            this.NumberOfCities = numberOfCities;
            this.DistanceMatrix = new int[NumberOfCities, NumberOfCities];
            this.PheromoneMatrix = new float[NumberOfCities, NumberOfCities];
        }

        public List<int> GetAllCities() => Enumerable.Range(0,
NumberOfCities).ToList();

        public void GenerateDistanceMatrix(int min, int max)
        {
```

```
            ++max;
            Random random = new Random();

            for (int i = 0; i < NumberOfCities; ++i)
            {
                for (int j = 0; j < NumberOfCities; ++j)
                {
                    if (i == j)
                        DistanceMatrix[i, j] = 0;
                    else
                        DistanceMatrix[i, j] = random.Next(min, max);
                }
            }
        }

        public void GeneratePheromoneMatrix(float min, float max)
        {
            Random random = new Random();

            for (int i = 0; i < NumberOfCities; ++i)
            {
                for (int j = 0; j < NumberOfCities; ++j)
                {
                    if (i == j)
                        PheromoneMatrix[i, j] = 0.0f;
                    else
                        PheromoneMatrix[i, j] = random.NextSingle() * (max - min) +
min;
                }
            }
        }

        public void UpdatePheromoneMatrix(float p, int Lmin, Ant[] ants)
        {
            int cityIndex;
            float delta;

            for (int i = 0; i < NumberOfCities; ++i)
            {
                for (int j = 0; j < NumberOfCities; ++j)
                {
                    if (i == j)
                        continue;

                    delta = 0;

                    for (int k = 0; k < ants.Length; ++k)
                    {
                        cityIndex = ants[k].VisitedCities.IndexOf(i);

                        if (cityIndex + 1 < ants[k].VisitedCities.Count)
                        {
                            if (ants[k].VisitedCities[cityIndex + 1] == j)
                                delta += (float)Lmin / ants[k].TravelledDistance;
                        }
                    }

                    this.PheromoneMatrix[i, j] = (1.0f - p) * this.PheromoneMatrix[i,
j] + delta;
                }
            }
        }
```

```
        public void PrintPath()
        {
            string sequence = String.Empty;
            HashSet<int> visitedCities = new HashSet<int>();
            var row = new[] { new { city = -1, pheromone = 0.0f } }.ToList();

            for (int i = 0; i < NumberOfCities; ++i)
            {
                row.Clear();

                for (int j = 0; j < NumberOfCities; ++j)
                    row.Add(new { city = j, pheromone = this.PheromoneMatrix[i, j] });

                row = row.OrderByDescending(edge => edge.pheromone).ToList();

                for (int j = 0; j < NumberOfCities; ++j)
                {
                    if (visitedCities.Add(row[j].city))
                    {
                        sequence += $"{row[j].city} -> ";
                        break;
                    }
                }
            }

            Console.WriteLine(sequence.Remove(sequence.Length - 4));
        }
    }
}
```

3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

```
1. Мінімальна відстань поточної ітерації: 395.
2. Мінімальна відстань поточної ітерації: 363.
3. Мінімальна відстань поточної ітерації: 391.
4. Мінімальна відстань поточної ітерації: 387.
5. Мінімальна відстань поточної ітерації: 398.
6. Мінімальна відстань поточної ітерації: 387.
7. Мінімальна відстань поточної ітерації: 398.
8. Мінімальна відстань поточної ітерації: 387.
9. Мінімальна відстань поточної ітерації: 398.
10. Мінімальна відстань поточної ітерації: 387.

Мінімальна відстань за результатом всіх ітерацій: 363
32 -> 47 -> 27 -> 45 -> 11 -> 42 -> 36 -> 10 -> 37 -> 35 -> 12 -> 3 -> 30 -> 22 -> 31 -> 46 -> 17 -> 43 -> 1 -> 23 -> 2
 -> 24 -> 34 -> 18 -> 26 -> 0 -> 4 -> 48 -> 14 -> 9 -> 16 -> 15 -> 6 -> 28 -> 29 -> 44 -> 33 -> 49 -> 5 -> 25 -> 8 -> 2
0 -> 39 -> 21 -> 7 -> 13 -> 38 -> 40 -> 41 -> 19
```

Рисунок 3.2 – Приклад роботи програми про 10 ітераціях

```
1. Мінімальна відстань поточної ітерації: 362.
2. Мінімальна відстань поточної ітерації: 325.
3. Мінімальна відстань поточної ітерації: 350.
4. Мінімальна відстань поточної ітерації: 325.
5. Мінімальна відстань поточної ітерації: 333.
6. Мінімальна відстань поточної ітерації: 325.
7. Мінімальна відстань поточної ітерації: 344.
8. Мінімальна відстань поточної ітерації: 325.
9. Мінімальна відстань поточної ітерації: 344.
10. Мінімальна відстань поточної ітерації: 325.
11. Мінімальна відстань поточної ітерації: 344.
12. Мінімальна відстань поточної ітерації: 325.
13. Мінімальна відстань поточної ітерації: 344.
14. Мінімальна відстань поточної ітерації: 325.
15. Мінімальна відстань поточної ітерації: 344.
16. Мінімальна відстань поточної ітерації: 325.
17. Мінімальна відстань поточної ітерації: 344.
18. Мінімальна відстань поточної ітерації: 325.
19. Мінімальна відстань поточної ітерації: 344.
20. Мінімальна відстань поточної ітерації: 325.
21. Мінімальна відстань поточної ітерації: 344.
22. Мінімальна відстань поточної ітерації: 325.
23. Мінімальна відстань поточної ітерації: 344.
24. Мінімальна відстань поточної ітерації: 325.
25. Мінімальна відстань поточної ітерації: 344.

Мінімальна відстань за результатом всіх ітерацій: 325
16 -> 34 -> 44 -> 7 -> 25 -> 24 -> 15 -> 46 -> 49 -> 0 -> 37 -> 45 -> 13 -> 36 -> 5 -> 4 -> 22 -> 39 -> 38 -> 20 -> 32
-> 17 -> 11 -> 40 -> 8 -> 21 -> 48 -> 2 -> 18 -> 30 -> 19 -> 23 -> 1 -> 14 -> 43 -> 31 -> 12 -> 41 -> 10 -> 29 -> 33 ->
 6 -> 47 -> 35 -> 9 -> 3 -> 26 -> 28 -> 42 -> 27
```

Рисунок 3.2 – Приклад роботи програми про 25 ітераціях

Тестування алгоритму

3.1.3 Значення цільової функції зі збільшенням кількості ітерацій

У таблиці 3.1 наведено значення цільової функції зі збільшенням кількості ітерацій.

Кількість ітерацій	Значення цільової функції
0	434
100	425
200	419
300	401
400	396
500	379
600	368
700	361
800	358
900	358
1000	358

3.1.4 Графіки залежності розв'язку від числа ітерацій

На рисунку 3.3 наведений графік, який показує якість отриманого розв'язку.

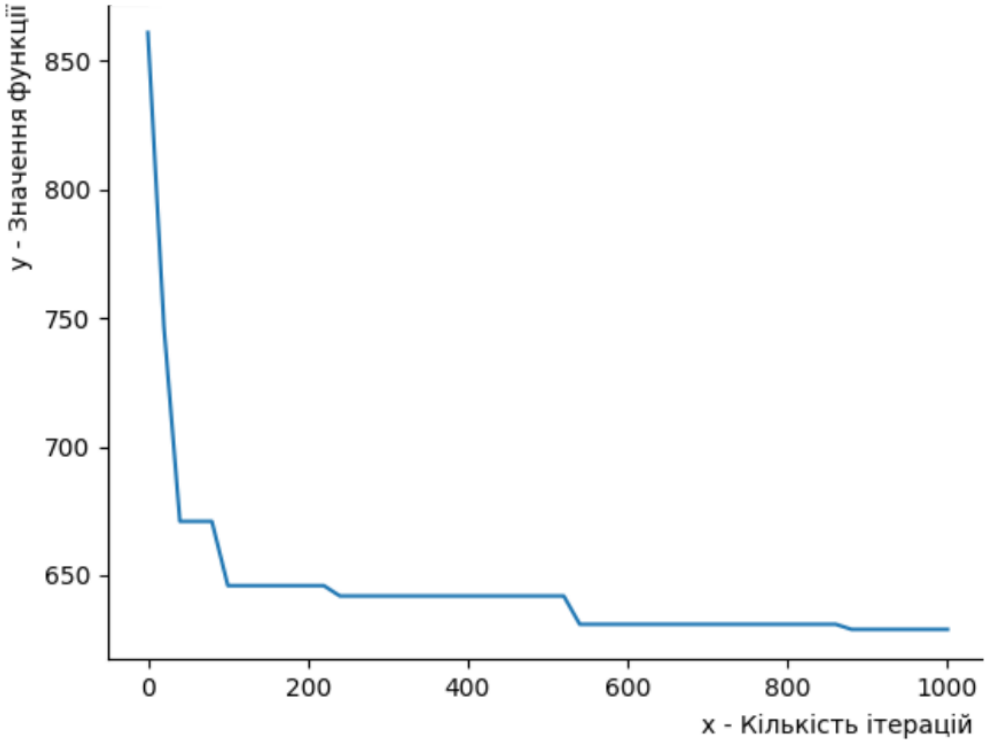

Рисунок 3.3 – Графік залежності розв'язку від числа ітерацій

Висновок

В рамках даної лабораторної роботи був розроблений алгоритм для приблизного вирішення задачі комівояжера за допомогою алгоритму мурашиної колонії. Карта міст була реалізована за допомогою неорієнтованого вагового графу, де відвідані міста кожної мурашки зберігаються у відповідному списку.

Критерії оцінювання

При здачі лабораторної роботи до 27.11.2021 включно максимальний бал дорівнює – 5. Після 27.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- програмна реалізація алгоритму – 75%;
- тестування алгоритму– 20%;
- висновок – 5%.