**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)**

(шифр, прізвище, ім'я, по батькові)

*Піонтківський Віталій ІП-12*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc51260917)

[2 Завдання 4](#_Toc51260918)

[3 Виконання 10](#_Toc51260919)

[3.1 Програмна реалізація алгоритму 10](#_Toc51260920)

[3.1.1 Вихідний код 10](#_Toc51260921)

[3.1.2 Приклади роботи 10](#_Toc51260922)

[3.2 Тестування алгоритму 11](#_Toc51260923)

[3.2.1 Значення цільової функції зі збільшенням кількості ітерацій 11](#_Toc51260924)

[3.2.2 Графіки залежності розв'язку від числа ітерацій 11](#_Toc51260925)

[Висновок 12](#_Toc51260926)

[Критерії оцінювання 13](#_Toc51260927)

# Мета лабораторної роботи

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

# 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** |
| 20 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 3, β = 2, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (10 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах). |

# Виконання

## Програмна реалізація алгоритму

### Вихідний код

import java.util.Arrays;  
  
public class Main {  
 public static void main(String[] args) {  
 Graph world = new Graph(200);  
  
 GreedySearch search = new GreedySearch(world);  
 int LMin = search.getMinLength();  
 System.*out*.println("Lmin " + LMin);  
 //-------------------------------------------------------  
 double[][] pheromone = new double[world.getMatrix().length][world.getMatrix().length];  
 for (int i = 0; i < pheromone.length; i++) {  
 for (int j = 0; j < pheromone.length; j++) {  
 pheromone[i][j] = 0.0001;  
 }  
 }  
 ACO algorithm = new ACO(world.getMatrix(), pheromone);  
 algorithm.setLMin(LMin);  
  
 algorithm.solveProblem();  
 }  
}

import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Random;  
  
public class ACO {  
 static final double *ALPHA* = 3;  
 static final double *BETA* = 2;  
 static final double *RO* = 0.7;  
 int LMin;  
 int LBest; // = Integer.MAX\_VALUE;  
 int[][] distancesMatrix;  
 double[][] pheromoneMatrix;  
  
 public ACO(int[][] distancesMatrix, double[][] pheromoneMatrix) {  
 this.distancesMatrix = distancesMatrix;  
 this.pheromoneMatrix = pheromoneMatrix;  
 }  
  
 public void setLMin(int LMin) {  
 this.LMin = LMin;  
 }  
  
 static final int *NUMBER\_OF\_ANTS* = 45;  
 static final int *NUMBER\_OF\_ELITE\_ANTS* = 10;  
 static final int *tMax* = 1000;  
  
 public void solveProblem() {  
 Ant[] ants = setAnts();  
 int bestLength = Integer.*MAX\_VALUE*;  
 int[] bestWay = new int[distancesMatrix.length];  
 for (int t = 0; t < *tMax*; t++) {  
 ArrayList<int[]> ways = new ArrayList<>();  
  
 for (int i = 0; i < ants.length; i++) {  
 //visit all vertices  
 int currentPosition = ants[i].getPosition();  
 ArrayList<Integer> visitedVertices = new ArrayList<Integer>();  
 while (visitedVertices.size() < distancesMatrix.length) {  
 visitedVertices.add(currentPosition);  
 currentPosition = calculateNewPosition(currentPosition, visitedVertices);  
 }  
 //add way to ways  
 int[] array = new int[visitedVertices.size()];  
 for (int j = 0; j < array.length; j++) {  
 array[j] = visitedVertices.get(j);  
 }  
 ways.add(array);  
 }  
 for (var way : ways) {  
 int length = countLength(way);  
 if (length < bestLength){  
 bestWay = way;  
 bestLength = length;  
 }  
 }  
  
 pheromoneMatrix = rewritePheromoneMatrix(ways);  
 if (t%20 == 0){  
 //System.out.println(Arrays.toString(bestWay));  
 System.*out*.print(bestLength+ " ");  
 }  
 }  
  
 //sout lBest tBest  
 System.*out*.println();  
 System.*out*.println("Best way vertices"+Arrays.*toString*(bestWay));  
 System.*out*.println("It's length - "+ bestLength);  
 }  
  
  
 private int countLength(int[]way){  
 int length = 0;  
 for (int i = 0; i < way.length-1; i++) {  
 length += distancesMatrix[way[i]][way[i+1]];  
 }  
 length+=distancesMatrix[distancesMatrix.length-1][0];  
 return length;  
 }  
 private double[][] rewritePheromoneMatrix(ArrayList<int[]> ways) {  
 for (int i = 0; i < pheromoneMatrix.length; i++) {  
 for (int j = 0; j < pheromoneMatrix.length; j++) {  
 pheromoneMatrix[i][j]\*= (1-*RO*);  
 }  
 }  
 for (var way: ways){  
 int deltaPheromone = LMin/countLength(way);  
 for (int i = 0; i < way.length-1; i++) {  
 pheromoneMatrix[way[i]][way[i+1]] += deltaPheromone;  
 pheromoneMatrix[way[i+1]][way[i]] += deltaPheromone;  
 if(i<=*NUMBER\_OF\_ELITE\_ANTS*){  
 pheromoneMatrix[way[i]][way[i+1]] += deltaPheromone;  
 pheromoneMatrix[way[i+1]][way[i]] += deltaPheromone;  
 }  
 }  
 pheromoneMatrix[way[way.length-1]][way[0]] += deltaPheromone;  
 pheromoneMatrix[way[0]][way[way.length-1]] += deltaPheromone;  
 }  
 return pheromoneMatrix;  
 }  
  
  
 private int calculateNewPosition(int currentPosition, ArrayList<Integer> visitedVertices) {  
 double[] probabilities = new double[distancesMatrix.length];  
 //calculate probabilities  
 for (int i = 0; i < probabilities.length; i++) {  
 if (visitedVertices.contains(i)) {  
 probabilities[i] = 0;  
 } else {  
 probabilities[i] = Math.*pow*(pheromoneMatrix[currentPosition][i], *ALPHA*) \* (Math.*pow*((1.0 / distancesMatrix[currentPosition][i]), *BETA*));  
 }  
 }  
 Random rand = new Random();  
 double sum = 0;  
 for (var i : probabilities) {  
 sum += i;  
 }  
 double nextPosProb = rand.nextDouble() \* sum;  
 double sumOfLittleProb = 0;  
 int result = findMax(probabilities);  
 while (sumOfLittleProb < nextPosProb) {  
 result = findMax(probabilities);  
 sumOfLittleProb += probabilities[result];  
 probabilities[result] = 0;  
 }  
 return result;  
 }  
  
 private int findMax(double[] probabilities) {  
 double max = Integer.*MIN\_VALUE*;  
 int maxIndex = 0;  
 for (int i = 0; i < probabilities.length; i++) {  
 if (probabilities[i] > max && probabilities[i]!=0) {  
 max = probabilities[i];  
 maxIndex = i;  
 }  
 }  
 return maxIndex;  
 }  
  
  
 */\*\*  
 \* method to set initial unique vertex for each ant  
 \*  
 \** ***@return*** *array of ants, elite ants at start  
 \*/* public Ant[] setAnts() {  
 Ant[] ants = new Ant[*NUMBER\_OF\_ANTS*];  
 int countOfEliteAnts = 0;  
 ArrayList<Integer> positions = new ArrayList<>();  
 Random rand = new Random();  
 for (int i = 0; i < ants.length; ) {  
  
 int pos = rand.nextInt(distancesMatrix.length);  
 if (!positions.contains(pos)) {  
 Ant ant;  
 if (countOfEliteAnts < *NUMBER\_OF\_ELITE\_ANTS*) {  
 ant = new Ant(true, pos);  
 countOfEliteAnts++;  
 } else {  
 ant = new Ant(false, pos);  
 }  
 positions.add(pos);  
 ants[i] = ant;  
 i++;  
 }  
 }  
 return ants;  
 }  
  
}

import java.util.ArrayList;  
  
public class GreedySearch {  
 Graph graph;  
  
 public GreedySearch(Graph graph) {  
 this.graph = graph;  
 }  
  
 public int getMinLength() {  
 ArrayList<Integer> revisedVertices = new ArrayList<>();  
 int result = 0;  
 int currentVertex = 0;  
 while (revisedVertices.size() < graph.getMatrix().length - 1) {  
 int newVertex = findMin(graph.getMatrix()[currentVertex], revisedVertices);  
 result += graph.getMatrix()[newVertex][currentVertex];  
 revisedVertices.add(currentVertex);  
 currentVertex = newVertex;  
 }  
 result += graph.getMatrix()[currentVertex][0];  
 revisedVertices.add(currentVertex);  
 System.*out*.println(revisedVertices);  
 return result;  
 }  
  
 private int findMin(int[] distances, ArrayList<Integer> revisedVertices) {  
 int min = Graph.*MAX\_DISTANCE* + 1;  
 int minIndex = 0;  
 for (int i = 0; i < distances.length; i++) {  
 if (distances[i] < min && !revisedVertices.contains(i) && distances[i] > 0) {  
 minIndex = i;  
 min = distances[i];  
 }  
 }  
 return minIndex;  
 }  
}

import java.util.Random;  
  
public class Graph {  
 private int[][] matrix;  
  
 public static final int *MIN\_DISTANCE* = 1;  
 public static final int *MAX\_DISTANCE* = 40;  
  
 public Graph(int numberOfVertexes) {  
 this.matrix = new int[numberOfVertexes][numberOfVertexes];  
 for (int i = 0; i < matrix.length; i++) {  
 for (int j = i; j < matrix[i].length; j++) {  
 if (j == i) {  
 matrix[i][j] = 0;  
 } else {  
 matrix[i][j] = *getRandomDistance*();  
 matrix[j][i] = matrix[i][j];  
 }  
  
 }  
 }  
 }  
  
 public int[][] getMatrix() {  
 return matrix;  
 }  
  
 private static int getRandomDistance() {  
 Random random = new Random();  
 return random.nextInt(*MAX\_DISTANCE* - *MIN\_DISTANCE* + 1) + *MIN\_DISTANCE*;  
 }  
}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

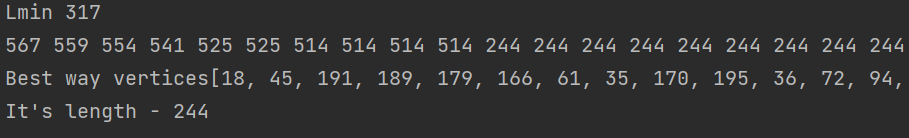


Рисунок 3.1

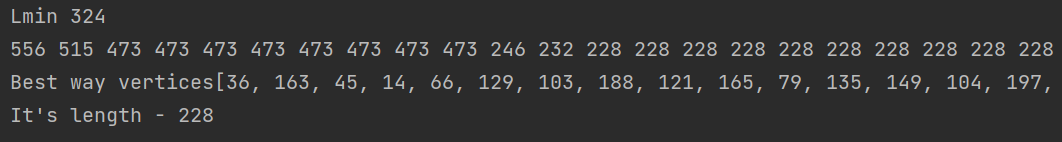


Рисунок 3.2

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Кількість ітерацій | Випробування 1 | Випробування 2 | Випробування 3 | Випробування 4 | Випробування 5 | Середнє значення |
| 0 | 601 | 628 | 526 | 576 | 519 | 570 |
| 20 | 581 | 560 | 511 | 557 | 519 | 545,6 |
| 40 | 562 | 545 | 511 | 524 | 519 | 532,2 |
| 60 | 538 | 545 | 511 | 524 | 518 | 527,2 |
| 80 | 538 | 545 | 511 | 524 | 518 | 527,2 |
| 100 | 538 | 545 | 495 | 524 | 500 | 520,4 |
| 120 | 538 | 530 | 495 | 524 | 500 | 517,4 |
| 140 | 538 | 530 | 495 | 524 | 500 | 517,4 |
| 160 | 513 | 530 | 495 | 524 | 494 | 511,2 |
| 180 | 513 | 530 | 495 | 524 | 494 | 511,2 |
| 200 | 291 | 258 | 226 | 201 | 252 | 245,6 |
| 220 | 285 | 257 | 226 | 201 | 228 | 239,4 |
| 240 | 278 | 257 | 226 | 201 | 228 | 238 |
| 260 | 278 | 257 | 226 | 201 | 228 | 238 |
| 280 | 278 | 257 | 226 | 201 | 228 | 238 |
| 300 | 278 | 257 | 226 | 201 | 228 | 238 |
| 320 | 278 | 257 | 226 | 201 | 228 | 238 |
| 340 | 278 | 257 | 226 | 201 | 228 | 238 |
| 360 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 380 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 400 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 420 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 440 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 460 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 480 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 500 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 520 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 540 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 560 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 580 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 600 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 620 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 640 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 660 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 680 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 700 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 720 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 740 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 760 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 780 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 800 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 820 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 840 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 860 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 880 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 900 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 920 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 940 | 278 | 257 | 226 | 201 | 227 | 237,8 |
| 960 | 276 | 257 | 226 | 201 | 227 | 237,4 |
| 980 | 276 | 257 | 226 | 201 | 227 | 237,4 |
| 1000 | 276 | 257 | 226 | 201 | 227 | 237,4 |

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

Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

Висновок

В рамках даної лабораторної роботи було створено програму, що реалізує мурашиний алгоритм для розв’язання задачі комівояжера. Програма складається з чотирьох класів: перший реалізує граф у вигляді матриці дистанцій між містами, другий виконує задачу жадібного алгоритму і знаходить досить оптимальний розв’язок задачі, що використовується мурашиним алгоритмом, третій клас реалізує мурашиний алгоритм з використанням результати роботи попередніх класів, четвертий містить точку входу програми і зв’язує класи між собою.

Було проаналізовано результати роботи програми. При даних налаштуваннях програми, результат, приблизно або точно відповідний найкращому, знаходиться за 100-140 ітерацій основного циклу програми. Результат різко покращується в проміжку між 100 та120 ітераціями, тоді «мурахи» починають рухатись за феромоном, а не «зором». Створено графіки п’яти випробувань та узагальненого результату роботи програми на різних матрицях відстаней, що показують ефективність роботи алгоритму.

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%.