МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №** 3

з дисципліни “Дискретні структури ”

тема “ Задача комівояжера ”

|  |  |  |
| --- | --- | --- |
| Виконав  студент I курсу  групи КП-62  Лук’янець Михайло Олександрович |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Жабіна Валентина Валеріївна  (*прізвище, ім’я, по батькові*) |

Київ 2017

**Завдання на лабораторну роботу**

1. Для даного викладачем графа G знайти матрицю суміжності розмірності n, елементами якої є ваги відповідних ребер.
2. Написати програму, яка на основі матриці суміжності знаходить найкоротший гамільтонів цикл алгоритмом найближчого міста та алгоритмом Літла.
3. Порівняти отримані результати. Зробити висновок.

**Вказівки щодо виконання завдання**

Програма може бути розроблена студентом на будь-якій мові програмування високого рівня: C++, C#, Java та інш. При написанні програми студент має використовувати відомі йому алгоритми, такі як алгоритми знаходження мінімального елемента масива, при цьому основні алгоритми мають виділятися в програмі в окремі підпрограми або функції. Програма має працювати для матриці довільної розмірності.

**Опис роботи програми.**

Програма, написана мовою Java, містить наступні методи, описані у абстрактному класі:

1. public static void printMatrix(int[][] matrix)

*виводить матрицю на екран*

1. public static int NearestCityAlg(int[][] matrix)

*обчислення Гамільтонового циклу методом найближчого міста*

1. public static int findMinInRow(int[][] matrix, int row, boolean[] bannedI, boolean[] bannedJ, int bannedIndex)

*знайти мінімальний елемент у строці*

1. public static int findMinInColumn(int[][] matrix, int column, boolean[] bannedI, boolean[] bannedJ, int bannedIndex)

*знайти мінімальний елемент у стовпці*

1. public static int leadMatrixByRowsAndCols(int[][] matrix, boolean[] bannedI, boolean[] bannedJ)

*приведення матриці по рядкам і стовпцям*

1. public static int LittleAlg(int[][] matrix)

*обчислення гамільтонового циклу алгоритмом Літтла*

**Алгоритм роботи програми**

1. Обчислення найкоротшого гамільтонового циклу і його довжини методом найближчого міста (жадібний алгоритм)
2. Вивід результатів обчислення методой найближчого міста
3. Обчислення найкоротшого гамільтоногового циклу і його довжини методом Літтла
4. Вивід результатів роботи алгоритму Літтла

**Лістинг коду програми**

|  |
| --- |
| Lab3.java |
| package lab3;  public class Lab3 {   public static final int *INF* = (int)(Integer.*MAX\_VALUE* / 2);  //вывод матрицы  public static void printMatrix(int[][] matrix){  //выводит матрицу на экран  for (int i = 0; i<matrix.length; i++){  for (int j = 0; j<matrix[i].length; j++)  if (matrix[i][j] == *INF*) System.*out*.print(" - ");  else System.*out*.format("%2d ", matrix[i][j]);  System.*out*.println("");  }  System.*out*.println("");  }  //жадный алгоритм  public static int NearestCityAlg(int[][] matrix){  //жадный алгоритм поиска методой ближайшего города  boolean passed[] = new boolean[matrix.length];  int resLen = 0;  String path = new String();  for (boolean i:passed){  i = false;  }  int iterCount = 0;  int start = 0;  int i = start;  path += Integer.*toString*(start + 1) + " - ";  passed[i] = true;  while (iterCount < matrix.length - 1){  int min = *INF*;  int minJ = 0;  for (int j = 0; j<matrix.length; j++){  if (matrix[i][j] != *INF* && matrix[i][j] < min && !passed[j]){  min = matrix[i][j];  minJ = j;  }  }  resLen+=matrix[i][minJ];  path += Integer.*toString*(minJ + 1) + " - ";  i = minJ;  passed[minJ] = true;  iterCount++;  }  resLen += matrix[i][start];  path += Integer.*toString*(start + 1);  System.*out*.println("Min loop path by Nearest City Algorhytm: ");  System.*out*.println(path);  return resLen;  }  // поиск минимального элемента в строке  public static int findMinInRow(int[][] matrix, int row, boolean[] bannedI, boolean[] bannedJ, int bannedIndex){  //найти минимумальный элемент в строке  int min = *INF*;  for (int j = 0; j < matrix.length; j++) {  if (matrix[row][j] != *INF* && matrix[row][j] < min && !bannedI[row] && !bannedJ[j] && j != bannedIndex) {  min = matrix[row][j];  }  }  return min;  } // поиск минимального элемента в столбце  public static int findMinInColumn(int[][] matrix, int column, boolean[] bannedI, boolean[] bannedJ, int bannedIndex){  //найти минимальный элемент в столбце  int min = *INF*;  for (int i = 0; i < matrix.length; i++) {  if (matrix[i][column] != *INF* && matrix[i][column] < min && !bannedI[i] && !bannedJ[column] && i != bannedIndex) {  min = matrix[i][column];  }  }  return min;  }  //первичная обработка матрицы   public static int leadMatrixByRowsAndCols(int[][] matrix, boolean[] bannedI, boolean[] bannedJ){  //приведение матрицы сначала по строкам, затем по столбцам  int tempResL = 0;  for (int i = 0; i < matrix.length; i++) {  int min = *findMinInRow*(matrix, i, bannedI, bannedJ, *INF*);  for (int j = 0; j < matrix.length; j++) {  if (matrix[i][j] != *INF* && !bannedI[i] && !bannedJ[j] ) {  matrix[i][j] -= min;  }  }  tempResL += min;  }  for (int j = 0; j < matrix.length; j++) {  int min = *findMinInColumn*(matrix, j, bannedI, bannedJ, *INF*);  for (int i = 0; i < matrix.length; i++) {  if (matrix[i][j] != *INF* && !bannedI[i] && !bannedJ[j]) {  matrix[i][j] -= min;  }  }  tempResL += min;  }  return tempResL;  } //алгоритм литтла   public static int LittleAlg(int[][] matrix){  //алгоритм Литтла  int resL = 0;  int[][] matrixClone = matrix.clone();  for(int i = 0; i<matrix.length; i++){  matrixClone[i] = matrix[i].clone();  }  boolean bannedI[] = new boolean[matrix.length];  boolean bannedJ[] = new boolean[matrix.length];   for (int i = 0; i < matrix.length; i++) {  bannedI[i] = false;  bannedJ[i] = false;  }  Path path = new Path(matrix.length);   for (int iter = 0; iter < matrix.length; iter++) {  resL += *leadMatrixByRowsAndCols*(matrix, bannedI, bannedJ);  //printMatrix(matrix);  int maxWeightI = -1;  int maxWeightJ = -1;  int maxWeight = -*INF*;  for (int i = 0; i<matrix.length; i++){  for (int j = 0; j<matrix.length; j++){  if (matrix[i][j] == 0 && !bannedI[i] && !bannedJ[j]){  int weight = *findMinInRow*(matrix, i, bannedI, bannedJ, j) + *findMinInColumn*(matrix, j, bannedI, bannedJ, i);  if (weight > maxWeight && !path.checkForEquality(i, j)){  maxWeight = weight;  maxWeightI = i;  maxWeightJ = j;  }  }  }  }  path.add(new Element(maxWeightI, maxWeightJ));  bannedI[maxWeightI] = true;  bannedJ[maxWeightJ] = true;  matrix[maxWeightJ][maxWeightI] = *INF*;  }  System.*out*.println("Min loop path by Little Algorhytm: ");  System.*out*.println(path.getPath(matrixClone));  //path.printElems();  return path.getLength(matrixClone);  }   public static void main(String[] args) {  int aMatrix[][]={  {*INF*, 6,4,8,7,14},  {6,*INF* ,7 ,11,7 ,10},  {4 ,7,*INF*, 4, 3 ,10},  { 8, 11 ,4 ,*INF*, 5, 11},  {7,7 ,3 ,5, *INF* ,7},  { 14,10,10,11,7,*INF*}  };   *printMatrix*(aMatrix);   int cityLen = *NearestCityAlg*(aMatrix);  System.*out*.format("Min loop length by Nearest City Algorhytm: %d\n", cityLen);  System.*out*.println("");  int littleLen = *LittleAlg*(aMatrix);  System.*out*.format("Min loop length by Little Algorhytm: %d\n", littleLen);  }    } |
|  |

|  |
| --- |
| Path.java |
| package lab3;  */\*\*  \* Created by Dell on 13.05.2017.  \*/* public class Path {  public Element[] elemArray;  public int size;  public int count = 0;   public Path(int size){  //конструктор класса, хранящий отрезки пути  this.size = size;  elemArray = new Element[size];  }   public void add(Element elem){  //добавление в массив отрезков  elemArray[count++] = elem;  }   public boolean checkForEquality(int a, int b) {  //проверка, нет ли уже в массиве отрезков такого отрезка  for (int i = 0; i < count; i++) {  if ((elemArray[i].a == a && elemArray[i].b == b)  || (elemArray[i].a == b && elemArray[i].b == a)) {  return true;  }  }  return false;  }   public void printElems() {  //вывод элементов пути  for (int i = 0; i < count; i++) {  System.*out*.format("%d - %d\n",elemArray[i].a + 1,elemArray[i].b + 1);  }  }   public String getPath(int[][] matrix){  //получить путь поэлементно  int cur = 0;  String res = new String();  res += Integer.*toString*(cur + 1) + " - ";  for (int i = 0; i < count; i++){  for (int j = 0; j<count; j++){  if (elemArray[j].a == cur){  cur = elemArray[j].b;  if (i != count - 1){  res += Integer.*toString*(cur + 1) + " - ";  }  else res += Integer.*toString*(cur + 1);  break;  }  }  }  return res;  }   public int getLength(int[][] matrix){  //получить длину пути  int cur = 0;  int len = 0;  for (int i = 0; i < count; i++){  for (int j = 0; j<count; j++){  if (elemArray[j].a == cur){  len += matrix[cur][elemArray[j].b];  cur = elemArray[j].b;  break;  }  }  }  return len;  } } |

|  |
| --- |
| Element.java |
| package lab3;  */\*\*  \* Created by Dell on 13.05.2017.  \*/* public class Element {  public int a;  public int b;  public boolean active;   public Element(int a, int b){  //конструктор элемента матрицы  this.a = a;  this.b = b;  active = true;  }  } |

**Висновки**

В цій лабораторній роботі були використані такі алгоритми як жадібний та алгоритм Літтла для знаходження оптимального шляху в задачі комівояжера. В результаті було встановлено, що алгоритм Літтла є більш ефективним.