НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №5

з дисципліни **«**Комп’ютерне моделювання**»**

Виконав:

студент 3 курсу

ФІОТ гр. ІО-21

Кузьменко Володимир

Перевірив:

Марковський О.П.

Київ – 2014 р.

Варіант завдання: 24. Знайти найдовший (без повторного заходження) шлях між точками по заданій матриці відстаней між точками. Граф однонаправлений. Матриця відстаней.

Лістинг коду програми

package com.cm.lab5;

// TODO: Auto-generated Javadoc

/\*\*

\* The Class Test.

\*/

public class Test {

/\*\*

\* The main method.

\*

\* @param args the arguments

\*/

public static void main(String[] args) {

int [][] testmas = {

// 0 1 2 3 4 5 6 7 8

/\*\*0\*/ {0,5,0,0,0,0,0,0,0},

/\*\*1\*/ {0,0,2,0,3,0,0,0,0},

/\*\*2\*/ {0,0,0,0,0,7,0,0,0},

/\*\*3\*/ {5,0,0,0,4,0,0,0,0},

/\*\*4\*/ {0,0,0,0,0,3,0,2,0},

/\*\*5\*/ {0,0,0,0,0,0,0,0,4},

/\*\*6\*/ {0,0,0,2,0,0,0,5,0},

/\*\*7\*/ {0,0,0,0,0,0,0,0,3},

/\*\*8\*/ {0,0,0,0,0,0,0,0,0}

};

GraphUtil util = new GraphUtil(testmas,0,8);

util.buildGraf();

util.calcPricesNodes();

System.out.println("\nGraf");

util.printGraph();

util.calcReturnTrack();

}

}

package com.cm.lab5;

import java.util.ArrayList;

import java.util.Iterator;

// TODO: Auto-generated Javadoc

/\*\*

\* The Class Node.

\*/

public class Node {

/\*\* The id. \*/

int id;

/\*\* The price. \*/

int price = -1;

/\*\* The max node. \*/

Node maxNode;

/\*\* The max transit. \*/

int maxTransit;

/\*\* The transition from price. \*/

ArrayList<Integer> transitionFromPrice = new ArrayList<Integer>();

/\*\* The transition in price. \*/

ArrayList<Integer> transitionInPrice = new ArrayList<Integer>();

/\*\* The nodes in this node. \*/

ArrayList<Node> nodesInThisNode = new ArrayList<Node>();

/\*\* The nodes from this node. \*/

ArrayList<Node> nodesFromThisNode = new ArrayList<Node>();

/\*\*

\* Instantiates a new node.

\*

\* @param id the id

\*/

public Node(int id) {

this.id = id;

// TODO Auto-generated constructor stub

}

/\*\*

\* Adds the link from this node.

\*

\* @param price the price

\* @param node the node

\*/

public void addLinkFromThisNode(int price, Node node) {

transitionFromPrice.add(price);

nodesFromThisNode.add(node);

}

/\*\*

\* Adds the link to this node.

\*

\* @param price the price

\* @param node the node

\*/

public void addLinkToThisNode(int price, Node node) {

transitionInPrice.add(price);

nodesInThisNode.add(node);

}

/\*\*

\* Calc node.

\*/

public void calcNode() {

int max = 0;

int i = 0;

int index = 0;

for (Integer prc : transitionInPrice) {

if (nodesInThisNode.get(i).price != -1) {

if (prc + nodesInThisNode.get(i).price > max) {

max = prc;

index = i;

maxNode = nodesInThisNode.get(index);

this.price = prc + nodesInThisNode.get(i).price;

maxTransit = prc;

}

}

i++;

}

}

/\*\*

\* Prints the track.

\*/

@SuppressWarnings("unused")

public void printTrack(){

int mxNode=0;

int max=0;

int maxNodeIndex;

int i=0;

Node mNode = null;

for (Node node : nodesInThisNode) {

if (node.price+transitionInPrice.get(i)>=max){

max = node.price+transitionInPrice.get(i);

maxNodeIndex = i;

mNode = node;

}

}

if (this.id!=GraphUtil.start){

if (this.id==GraphUtil.finish){

System.out.print(""+this.id+"");

mNode.printTrack();

}else{

System.out.print(" <--- "+this.id+"");

mNode.printTrack();

}

}else{

System.out.print(" <--- "+this.id+"");

}

}

/\* (non-Javadoc)

\* @see java.lang.Object#toString()

\*/

@Override

public String toString() {

Iterator<Node> nodeItr = nodesFromThisNode.iterator();

int i = 0;

String str = new String("this node " + id + " price node = " + price

+" ");

if (maxNode==null){

str = str +("null");

}else{

str = str +(maxNode.id);

}

str = str +("\n");

while (nodeItr.hasNext()) {

Node node = (Node) nodeItr.next();

str = str + " id from= " + node.id + " price transit = "

+ transitionFromPrice.get(i);

i++;

}

str = str + "\n";

nodeItr = nodesInThisNode.iterator();

i = 0;

while (nodeItr.hasNext()) {

Node node = (Node) nodeItr.next();

str = str + " id in= " + node.id + " price node = "

+ node.price;

i++;

}

str = str + "\n";

return str;

}

}

package com.cm.lab5;

import java.util.ArrayList;

// TODO: Auto-generated Javadoc

/\*\*

\* The Class Level.

\*/

public class Level {

/\*\* The items. \*/

ArrayList<Node> items = new ArrayList<Node>();

/\*\* The util. \*/

GraphUtil util;

/\*\*

\* Instantiates a new level.

\*

\* @param util the util

\*/

public Level(GraphUtil util) {

this.util = util;

}

/\*\*

\* Instantiates a new level.

\*

\* @param node the node

\*/

public Level(Node node) {

items.add(node);

}

/\*\*

\* Adds the all.

\*

\* @param nodes the nodes

\*/

public void addAll(ArrayList<Node> nodes) {

for (Node node : nodes) {

if (contains(node)){

items.add(node);

}

}

}

/\*\*

\* Adds the node.

\*

\* @param node the node

\*/

public void addNode(Node node){

items.add(node);

}

/\*\*

\* Calc level.

\*/

public void calcLevel(){

for (Node node : items) {

node.calcNode();

}

util.endFlag=true;

}

/\*\*

\* Builds the next level.

\*

\* @return the level

\*/

public Level buildNextLevel(){

Level nextLevel = new Level(util);

for (Node node : items) {

nextLevel.addAll(node.nodesFromThisNode);

}

if (nextLevel.items.size()==0)util.endFlag=false;

return nextLevel;

}

/\* (non-Javadoc)

\* @see java.lang.Object#toString()

\*/

@ Override

public String toString(){

String str = new String(" ");

for (Node node : items) {

str = str + node.toString();

}

str = str + "\n";

return str;

}

/\*\*

\* Contains.

\*

\* @param node the node

\* @return true, if successful

\*/

private boolean contains(Node node){

boolean flag = true;

for (Node nd : items) {

if (nd.id==node.id){

flag = false;

break;

}

}

return flag;

}

}

package com.cm.lab5;

// TODO: Auto-generated Javadoc

/\*\*

\* The Class GraphUtil.

\*/

public class GraphUtil {

/\*\* The start. \*/

static int start;

/\*\* The finish. \*/

static int finish;

/\*\* The sum. \*/

static int sum=0;

/\*\* The end flag. \*/

boolean endFlag=true;

/\*\* The nodes. \*/

Node [] nodes;

/\*\* The matrix. \*/

int [][] matrix;

/\*\*

\* Instantiates a new graph util.

\*

\* @param matrix the matrix

\* @param start the start

\* @param finish the finish

\*/

@SuppressWarnings("static-access")

public GraphUtil(int [][] matrix, int start, int finish) {

this.matrix = matrix;

nodes = new Node[matrix.length];

for (int i = 0; i < matrix.length; i++) {

nodes[i] = new Node(i);

}

this.start = start;

this.finish = finish;

}

/\*\*

\* Gets the node.

\*

\* @param index the index

\* @return the node

\*/

public Node getNode(int index){

return nodes[index];

}

/\*\*

\* Builds the graf.

\*/

public void buildGraf(){

for (int i = 0; i < matrix.length; i++) {

Node buf = getNode(i);

for (int j = 0; j < matrix[i].length; j++) {

if (matrix[i][j]!=0){

buf.addLinkFromThisNode(matrix[i][j], getNode(j));

getNode(j).addLinkToThisNode(matrix[i][j], buf);

}

}

}

System.out.println("\nBuilding completed");

}

/\*\*

\* Calc prices nodes.

\*/

public void calcPricesNodes(){

nodes[start].price=0;

Level buf = new Level(this);

buf.addNode(nodes[start]);

buf =buf.buildNextLevel();

while (endFlag) {

buf.calcLevel();

buf = buf.buildNextLevel();

}

System.out.println("\nCalc node prices completed");

}

/\*\*

\* Calc return track.

\*/

public void calcReturnTrack(){

System.out.println("\nResult:");

if (nodes[finish].price==-1){

System.err.println("Top unattainable");

}else{

nodes[finish].printTrack();

System.out.println("\nPrice Track "+nodes[finish].price);

}

}

/\*\*

\* Prints the graph.

\*/

public void printGraph(){

for (Node node : nodes) {

System.out.println(node.toString());

}

}

}