UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE CIENCIAS

Escuela Profesional de Ciencias de la Computación

Algoritmos y Estructuras de Datos

Examen Final – Parte II

Fecha: 14.07.2023

Hora: 8:00hrs-9:00hrs

Lugar: Sala 3

Código : 20211405E Apellidos y Nombres: Sanchez Malaspina Cesar

1. Realice la demostración del Código utilizando el caso del cartero chino (cap 8 -drozdek), Ref. Código Github https://github.com/inwe-research-group/DrozdekCap8Graphs (5pts)

package CarteroChinoHamiltoniano;

import java.util.ArrayList;

import java.util.List;

public class CarteroChinoHamiltoniano {

//Se declara la clase CarteroChinoHamiltoniano

private static final int INF = Integer.MAX\_VALUE / 2;

//Se declara INF

public static void main(String[] args) {

int[][] grafo = {

{0, 10, 0, 30, 0},

{10, 0, 20, 5, 10},

{0, 20, 0, 0, 15},

{30, 5, 0, 0, 20},

{0, 10, 15, 20, 0}

};

//Se define la matriz de adyacencia GRAFO la cual representa la grafo propuesto

List<Integer> rutaOptima = encontrarRutaOptima(grafo);

//Se llama al metodo encontrarRutaOptima

System.out.println("Ruta óptima: " + rutaOptima);

//Se imprime la ruta optima resultante

}

public static List<Integer> encontrarRutaOptima(int[][] grafo) {

int n = grafo.length;

int[][] matrizRecorrido = new int[n][n];

//Se inicializa la matriz recorrido como una copia del grafo original

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

matrizRecorrido[i][j] = grafo[i][j];

if (grafo[i][j] == 0 && i != j) {

matrizRecorrido[i][j] = INF;

}

}

}

//Se recorre cada elemento de la matriz "grafo y se procede a copiar los valores correspondientes a matrizRecorrido

//Adicional a ello se le agrego un "if" el cual se encarga de verificar si el elemento dado del grafo es 0 y si 1 no

//es igual a j, cuando sucede esos 2 casos, se le asignara el valor "INF" a la posicion correspondiente de la matrizRecorrido

for (int k = 0; k < n; k++) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (matrizRecorrido[i][k] + matrizRecorrido[k][j] < matrizRecorrido[i][j]) {

matrizRecorrido[i][j] = matrizRecorrido[i][k] + matrizRecorrido[k][j];

}

}

}

}

//Este algoritmo se encarga de encontrar los caminos mas cortos del grafo

int[] grados = new int[n];

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (grafo[i][j] != 0 && grafo[i][j] != INF) {

grados[i]++;

grados[j]++;

}

}

}

//Aqui se procede a calcular el grafo de cada nodo del grafo original

List<Integer> verticesImpares = new ArrayList<>();

for (int i = 0; i < n; i++) {

if (grados[i] % 2 != 0) {

verticesImpares.add(i);

}

}

//Se crea la lista verticesImpares donde se almacenara los indices de los nodos con grado impar

List<Integer> rutaOptima = new ArrayList<>();

if (!verticesImpares.isEmpty()) {

int inicio = verticesImpares.get(0);

encontrarRutaHamiltoniana(matrizRecorrido, inicio, rutaOptima);

} else {

encontrarRutaHamiltoniana(matrizRecorrido, 0, rutaOptima);

}

return rutaOptima;

}

//Se crea la lista "rutaOptima" donde se almacenará la ruta resultante

//Tambien se verifica la existencia de nodos con grado impar en "verticesImpares"

//Se selecciona el primer nodo de "verticesImpares" como nodo inicial y se llama al metodo "encontrarRutaHamiltoniana

private static boolean encontrarRutaHamiltoniana(int[][] matrizRecorrido, int v, List<Integer> rutaOptima) {

rutaOptima.add(v);

//Se agrega el nodo v a rutaOptima

if (rutaOptima.size() == matrizRecorrido.length) {

return true;

}

//Si rutaOptima contiene todos los nodos del grafo retorna true, lo que quiere decir es que se encontro una ruta hamiltoniana completa

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

if (matrizRecorrido[v][i] != 0 && matrizRecorrido[v][i] != INF) {

matrizRecorrido[v][i] = 0;

matrizRecorrido[i][v] = 0;

//Se modifico el if añadiendo la condicion de que el elemento de la matrizRecorrido no sea INF

if (encontrarRutaHamiltoniana(matrizRecorrido, i, rutaOptima)) {

return true;

}

matrizRecorrido[v][i] = 1;

matrizRecorrido[i][v] = 1;

}

}

rutaOptima.remove(rutaOptima.size() - 1);

return false;

//Si no se encuentra ruta hamiltoniana, retorna false

}

}

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Realice la demostración del Código utilizando el caso del bloqueo mortal o puntos muertos (cap 8 -drozdek), Ref. Código Github https://github.com/inwe-research-group/DrozdekCap8Graphs (5pts)

package DeadlockDetection1;

import java.util.\*;

public class DeadlockDetection1 {

private final Map<String, List<String>> graph;

private final Set<String> visited;

private final Set<String> cycleNodes;

public DeadlockDetection1() {

graph = new HashMap<>();

visited = new HashSet<>();

cycleNodes = new HashSet<>();

}

public void addDependency(String from, String to) {

graph.putIfAbsent(from, new ArrayList<>());

graph.get(from).add(to);

}

public boolean hasCycle() {

for (String node : graph.keySet()) {

if (dfs(node)) {

return true;

}

}

return false;

}

private boolean dfs(String node) {

if (cycleNodes.contains(node)) {

return true; // Retorna true cuando encuentra un ciclo

}

//Se agrego el if para corroborar que contenga el nodo, para evitar errores

if (!visited.add(node)) {

return false; // Retorna falso cuando ya visito ese nodo

}

cycleNodes.add(node);

if (graph.containsKey(node)) {

for (String neighbor : graph.get(node)) {

if (dfs(neighbor)) {

return true; // Ciclo detectado

}

}

}

cycleNodes.remove(node);

return false;

}

public static void main(String[] args) {

DeadlockDetection1 detector = new DeadlockDetection1();

// Agregar dependencias

detector.addDependency("T1", "T2");

detector.addDependency("T2", "T1");

detector.addDependency("T3", "T2");

// Detectar deadlock

if (detector.hasCycle()) {

System.out.println("Se detectó un deadlock");

} else {

System.out.println("No se detectó un deadlock");

}

}

}

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. En ambos casos complementariamente debe documentar el Código paso a paso, determinar si se ha omitido alguna lógica para que estos resuelvan los problemas o casos planteados. Asimismo, en el link Github indicado ubicara varias implementaciones de cada caso, puede elegir cualquiera para su demostración y documentación. La presentacion de la prueba sera directamente desde github o el drive de google. El cual debe adjuntar como respuesta en el classroom.