Nombre y Apellido: ...................................... N° Legajo: .............

#### Recuperatorio del Segundo Parcial de Estructuras de Datos y Algoritmos 72.34

#### Primer Cuatrimestre de 2016

|  |  |  |  |
| --- | --- | --- | --- |
| Ejercicio 1 | ***Ejercicio 2*** | ***Ejercicio 3*** | Nota |
|  |  |  |  |

Condición Mínima de Aprobación: Tener por lo menos dos ejercicios con B-

###### Consideraciones a tener en cuenta. MUY IMPORTANTE

###### El ejercicio que no respete estrictamente el enunciado será anulado.

* **Se puede entregar el examen escrito en lápiz.**

###### Se tendrán en cuenta la eficiencia y el estilo de programación.

###### Entregar los ejercicios en hojas separadas

## Ejercicio 1

Se cuenta con la siguiente implementación parcial de un grafo no dirigido con peso en las aristas:

public class Graph<V> {

private HashMap<V, Node> nodes = new HashMap<V, Node>();

private List<Node> nodeList = new ArrayList<Node>();

public void addVertex(V vertex) {

if (!nodes.containsKey(vertex)) {

Node node = new Node(vertex);

nodes.put(vertex, node);

nodeList.add(node);

}

}

public void addArc(V v, V w, Double d) {

Node origin = nodes.get(v);

Node dest = nodes.get(w);

if (origin != null && dest != null && !origin.equals(dest)) {

for (Arc arc : origin.adj) {

if (arc.neighbor.info.equals(w)) {

return;

}

}

origin.adj.add(new Arc(dest, d));

dest.adj.add(new Arc(origin, d));

}

}

private class Node {

V info;

boolean visited = false;

int tag = 0;

List<Arc> adj = new ArrayList<Arc>();

public Node(V info) {

this.info = info;

}

public int hashCode() {

return info.hashCode();

}

public boolean equals(Object obj) {

if (obj == null || (obj.getClass() != getClass())) {

return false;

}

return info.equals(((Node)obj).info);

}

}

private class Arc {

Node neighbor;

Double weight;

public Arc(Node neighbor, Double weight) {

this.neighbor = neighbor;

this.weight = weight;

}

}

}

Implementar un algoritmo que dado un numero postivo N diga si el grafo dado es N-coloreable.

**public boolean isColoreable(int n)**

## Ejercicio 2

Agregarle a la implementación de grafos del ejercicio anterior el método **countCycles**, que recibe por parámetro un valor **n**, y cuenta la **cantidad total de ciclos simples de longitud n** que existen en el grafo.

Ejemplo: para el siguiente grafo, si se invoca con 3 retorna 4. Si se invoca con 4 retorna 3. Si se invoca con 6 retorna 1.

***Ejercicio 3***

Se tiene una matriz en donde las celdas pueden o no contener una moneda. La matriz se recorre desde la celda superior izquierda hasta la celda inferior derecha, y sólo se pueden hacer movimientos hacia la derecha o hacia abajo.

Implementar un algoritmo que encuentre el camino que maximice la cantidad de monedas encontradas y retorne **la secuencia de movimientos que conforman dicho camino**. La entrada es una matriz de tipo boolean, donde **true** representa una moneda.

El algoritmo debe implementar al menos una poda.

public enum Direction {DOWN, RIGHT};

public static List<Direction> maxCoins(boolean[][] board)

Ejemplo: Para la siguiente matriz, el algoritmo debería retornar {DOWN, RIGHT, RIGHT, DOWN, RIGHT, DOWN}.

|  |  |  |  |
| --- | --- | --- | --- |
| X |  |  | X |
| X |  | X |  |
|  |  | X | X |
| X | X |  | X |