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

#### Segundo Parcial de Estructuras de Datos y Algoritmos

#### Primer Cuatrimestre de 2016

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

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

## Ejercicio 1

Se cuenta con la siguiente implementación parcial de un grafo no dirigido:

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) {

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));

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

}

}

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;

public Arc(Node neighbor) {

this.neighbor = neighbor;

}

}

}

Agregar a la implementación de grafos un método que reciba una lista de objetos de tipo V y determine si dicha lista se corresponde con un recorrido BFS del grafo.

**public boolean isBFS(List<V> values)**

Ejemplo: dado el siguiente grafo, para estas entradas el algoritmo retornaría true: {A, B, C, D, E, F}, {A, B, C, E, F, D}, {A, C, B, E, F, D}, {C, A, B, F, E, D}. Para las siguientes entradas retornaría false: {A, B, D, C, E, F}, {A, B, C, T}, {A, A, B, C, D, E, F}, {B, C, F, A, D, E}.

## Ejercicio 2

Agregarle a la implementación de grafos del ejercicio anterior el método booleano **hasCenter**, que reciba un número entero que representa un **radio** y determine si el grafo contiene un “centro” para ese radio.

Un nodo es “centro” para un determinado radio si el resto de los nodos está a una distancia menor o igual al radio.

Para el siguiente grafo A y D son centro con radio 2 pero no con radio 1. B y C son centro con radio 1.

## Ejercicio 3

Se desea implementar la siguiente función: Dado una lista de números positivos L y un número N dar todas las posibles combinaciones posibles de sumas de números de la lista L (admitiendo repetidos) que sumen N.

**public Set<List<Integer> possibleSums(List<Integer> numbers, int n)**

Ejemplo: possibleSums(L, N) con L = [1, 2, 3, 4, 7, 10] y N = 5 debería retornar: {[1, 1, 1, 1, 1], [1, 1, 1, 2], [1, 2, 2], [1, 1, 3], [1, 4], [2, 3]}