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

Segundo Parcial de Estructuras de Datos y Algoritmos

Primer Cuatrimestre de 2013 – 18/06/2013

Ejercicio 1	Ejercicio 2	Ejercicio 3	Nota

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

Ejercicio 1

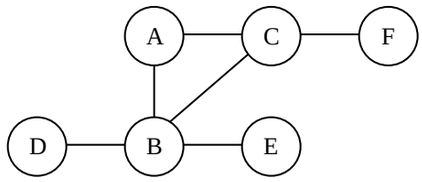
Se cuenta con la siguiente implementación parcial de un grafo no dirigido:

```
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

Un autómata finito puede utilizarse para determinar si una secuencia de caracteres pertenece o no a un lenguaje regular, o –lo que es lo mismo- determinar si una expresión "matchea" con una expresión regular. Cada estado recibe un identificador y entre dos estados puede haber una transición asociada a un carácter. Además los estados pueden ser de aceptación o no.

Cada autómata tiene distinguido un único estado inicial y uno o más estados de aceptación. Cuando se lee una secuencia de caracteres se comienza desde el estado inicial, si del estado inicial hay una transición asociada al carácter ingresado se pasa al estado que "apunta" la transición. A medida que se leen caracteres se va pasando de un estado a otro hasta que:

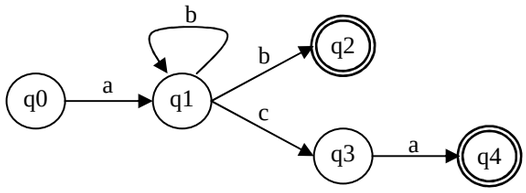
- no hay transición asociada al carácter leído. En este caso se dice que la secuencia de caracteres no es aceptada
- no hay más caracteres para leer. Si el estado en el que estamos es de aceptación, se acepta la secuencia, caso contrario se rechaza.

En un autómata finito no determinista, puede haber más de una transición posible desde un estado, dada cierta entrada. En estos casos se deben analizar todas las opciones posibles, y basta con que una finalice en un estado de aceptación para considerar a la entrada aceptada.

Se pide crear una clase que implemente un autómata finito no determinista basado en un grafo dirigido con listas de adyacencia. Los métodos que deben implementarse son:

- Crear el autómata, que recibe como dato adicional cuál es el estado inicial (cada estado se identifica con un string) y si el mismo es de aceptación o no
- Agregar un estado (un nodo que se identifica con un string y además se debe saber si es de aceptación o no)
- Agregar una transición (un eje entre dos nodos con un char como información asociada)
- Un método que reciba una cadena de caracteres y retorne **true** si la misma es aceptada por el autómata o **false** en caso contrario.

Ejemplo: En el siguiente autómata, q0 es el estado inicial, q2 y q4 son estados de aceptación. Las siguientes cadenas son aceptadas: "ab", "abb", "abbbb", "abbbca". Las siguientes cadenas no son aceptadas: "a", "ad", "abbba", "abbcab", "abbbc".



Ejercicio 3

Se denomina Vertex Cover a un conjunto de nodos, tal que todas las aristas del grafo son incidentes en al menos un nodo de este conjunto. Agregarle a la implementación de grafos del ejercicio 1 un método que encuentre el vertex cover mínimo (es decir, con la menor cantidad de nodos) y retorne los nodos que lo conforman.

Para el grafo de ejemplo del ejercicio 1, el vertex cover mínimo está dado por los nodos {B, C}.

Ejercicio 3 (variante)

Dado un grafo no dirigido G con N vértices (donde N <= 1000) con peso no negativo en sus ejes, y cada nodo está identificado con un número entre 1 y 1000, escribir un método que retorne el camino mínimo entre dos nodos, si no existe que retorne null. Para resolverlo se debe usar Programación Dinámica (no se aceptará una solución que no cumpla con lo pedido).