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

Recuperatorio Segundo Parcial de Estructuras de Datos y Algoritmos

25/06/2010

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.**

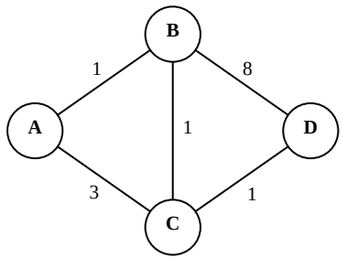
Ejercicio 1

Se cuenta con la siguiente implementación parcial de un grafo que almacena pesos enteros positivos 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, int weight) {
		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(weight, dest));
			dest.adj.add(new Arc(weight, 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 {
		int weight;
		Node neighbor;

		public Arc(int weight, Node neighbor) {
			this.weight = weight;
			this.neighbor = neighbor;
		}
	}
}
```

Agregarle a esta implementación el método **maxDistance**, que dado un nodo origen y un nodo destino retorna el peso del camino **más largo** entre ambos (sin ciclos).

En el siguiente grafo, el camino máximo entre los nodos A y D está formado por los nodos {A, C, B, D} y tiene peso 12.



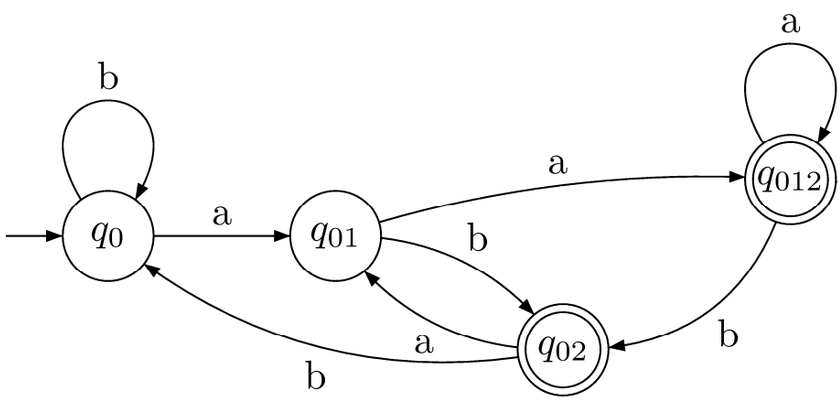
Ejercicio 2

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 (en el ejemplo siguiente desde el estado inicial q_0 si se lee una b se pasa al estado q_{01}, si se lee una b seguimos en q_0). 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.

El siguiente autómata acepta palabras formadas con las letras 'a' y 'b', y cuya penúltima letra sea una 'a'



Se pide crear una clase que implemente un autómata finito 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.

Ejercicio 3

La clase EDAUtils contiene un método estático **evaluate** que recibe un vector de variables booleanas y –aplicando relaciones lógicas- retorna **true** o **false**. Dicho vector puede tener cualquier dimensión.

Escribir una clase que implemente un método que reciba un único parámetro entero de nombre **dim** (que debería ser mayor que cero) e imprima todas las combinaciones de vectores booleanos de dimensión **dim** tal que, al ser evaluados por el método evaluate de EDAUtils retornen **true**.