Examen de Estructuras de Datos y Algoritmos
(Ingeniería Informática)

Septiembre 2007

Primera parte (50% nota del examen)

1) Se dispone de una clase `Articulo` cuyos objetos representan artículos en venta en una tienda. Cada artículo tiene como atributos su nombre, referencia, y precio. Disponen de una operación de comparación (`compareTo()`) que compara los artículos por precio. La interfaz de la clase se muestra a continuación.

```
import java.util.*;
public class Articulo implements Comparable
{

     /**
      * Constructor al que se le pasa el nombre,
      * la referencia y el precio;
      */
     public Articulo(String nombre, int referencia, double precio)
     {...}

     /**
      * Retorna el nombre
      */
     public String nombre(){...}

     /**
      * Retorna la referencia
      */
     public int referencia(){...}

     /**
      * Retorna el precio
      */
     public double precio(){...}

     /**
      * Comparación de clientes por su precio
      */
     public int compareTo(Object otro) {...}

}
```

Se pide escribir un método java para cambiar los precios de una lista de artículos que se almacena en una cola de prioridad (ordenados por tanto por su precio). Al método se le pasa la cola de prioridad con la lista inicial de artículos, y un mapa que contiene como claves referencias de artículos y como valores sus nuevos precios. El método debe retornar una nueva cola de prioridad con todos los artículos, con el nuevo precio en aquellos artículos cuyo precio haya cambiado. Para ello, el método va extrayendo de la cola inicial cada artículo. Si la referencia del artículo está en el mapa crea un nuevo artículo con el mismo nombre y referencia pero el precio nuevo (obtenido del mapa) y lo inserta en la nueva cola. En caso contrario, inserta en la nueva cola el artículo original. Al finalizar, retorna la nueva cola. La cabecera del método debe ser:

```
public static PriorityQueue<Articulo> cambiaPrecio(
   PriorityQueue<Articulo> cola, Map<Integer,Double> mapa)
```

2) Para la clase `Articulo` del ejercicio anterior, se desea hacer un método que compruebe si una lista de artículos tiene nombres o referencias repetidos. Para cada uno de aquellos elementos que tengan la referencia repetida, debe mostrarse en pantalla un mensaje de error con la referencia del artículo. Para cada uno de aquellos elementos que tengan el nombre repetido, debe mostrarse en pantalla un mensaje de error con el nombre y referencia del artículo. Al finalizar, el método retornará un booleano que será `true` si no se ha encontrado ningún nombre ni ninguna referencia repetidos, y `false` en caso contrario. La eficiencia del método debe ser $O(n)$ en promedio, para lo cual pueden utilizarse, por ejemplo, conjuntos de la clase `HashSet` para detectar elementos repetidos. La cabecera del método será:

```
public static boolean compruebaLista(List<Articulo> lista)
```

3) Se dispone de un grafo dirigido acíclico cuyos vértices contienen nombres de asignaturas de un plan de estudios. Las aristas del grafo representan dependencias entre asignaturas, de modo que si hay una arista desde la asignatura A a la B esto representa que para poder matricularse de B hay que haber completado la asignatura A. El grafo sigue la interfaz de grafos vista en clase.

   Se pide escribir el pseudocódigo de un método al que se le pasa el grafo, una lista de las asignaturas aprobadas por un alumno, y una asignatura de la que el alumno se quiere matricular. El método debe retornar un booleano que indique si el alumno puede matricularse o no de esa asignatura. Razonar asimismo sobre la eficiencia de este método en función del número de aristas del grafo y del número de asignaturas aprobadas, suponiendo que el grafo se implementa mediante listas de adyacencia y que la lista de asignaturas aprobadas es una lista enlazada no ordenada.

Examen de Estructuras de Datos y Algoritmos
(Ingeniería Informática)

Septiembre 2007

Segunda parte (50% nota del examen)

4) Escribir el pseudocódigo de una operación para una tabla hash de troceado cerrado que retorne el índice de colisiones de la tabla. Este índice se define como la cantidad total de celdas no nulas consecutivas de la tabla (tanto si están borradas como si no), dividida entre la cantidad total de celdas no nulas. Se considera para determinar el orden consecutivo, que la celda anterior a la primera de la tabla es la última de la misma tabla. La operación es interna a la clase, por lo que tiene acceso a la parte privada de la implementación.

5) Escribir para la clase `ColaEnlazada` vista en clase (que representa una cola implementada mediante una lista simplemente enlazada) una operación `remove`() que elimine un elemento dado de la cola. La operación tiene la cabecera que se muestra abajo. Funciona recorriendo la cola en busca del elemento `element`, y borrándolo de la cola si lo encuentra. En ese caso retorna `true`, mientras que si el elemento no se encuentra en la cola retorna `false`.

```
boolean remove(Object element);
```

6) Escribir el pseudocódigo de una operación a la que se le pasa un árbol binario que obedece a la interfaz del árbol binario y un iterador que corresponde al iterador de árbol binario vistos en clase. El árbol binario contiene números enteros. La operación debe retornar un entero que sea la suma del valor del nudo actual y de todos sus descendientes. Retornará cero si el nudo actual no es válido.