Examen de Estructuras de Datos y Algoritmos
(Ingeniería Informática)

Febrero 2010

Primera parte (50% nota del examen)

1) Se desea hacer una clase llamada `ListaClientes` que permita almacenar un conjunto de objetos de la clase `Cliente` organizados de manera que se agrupen juntos los clientes de la misma provincia. La clase cliente dispone de los métodos siguientes que retornan el número de una provincia (entre 1 y 51) y el nombre del cliente:

```
int numProvincia()
String nombre()
```

La clase `ListaClientes` deberá tener una lista de 51 listas de clientes, una para cada provincia. Se pide escribir la clase con un constructor al que se le pasa una lista de clientes, y los organiza metiendo a cada uno en su provincia. También escribir un método que busca a un cliente y lo retorna.

```
ListaClientes(List<Cliente> lista)
Cliente busca(String nombre)
```

Indicar también la eficiencia del constructor y del método de búsqueda en función de *n*, número total de clientes, y *m*, número máximo de clientes por provincia.

2) Se desea crear una clase llamada `Apuestas` para almacenar las quinielas de un grupo de personas. Cada quiniela es un objeto que contiene un array de 15 caracteres (que serán '1', 'X' ó '2') y la jornada de la liga que le corresponde. La clase `Quiniela` dispone del método `equals` que permite comprobar si dos quinielas son iguales.

Cada persona se identifica por su nombre y tiene una lista de quinielas que ha ido haciendo desde la primera jornada de la liga. Estos datos se guardan en un mapa, en el que la clave es el nombre de la persona y el valor es una `LinkedList` de objetos de la clase `Quiniela`. Se pide escribir un método que añade una quiniela al final de la lista de una persona. Asimismo, otro método que lista en pantalla los nombres de las personas que tengan una quiniela que coincida con la que se pasa como parámetro.

```
void insertaQuiniela(String nombrePersona, Quiniela q)
void listaCoincidencias(Quiniela q)
```

Indicar la eficiencia de estos métodos.

3) Escribir el pseudocódigo de un método al que se le pasa un grafo que utiliza la interfaz de grafos vista en clase, el contenido de un vértice inicial, y un número entero `n`, y que retorne una lista de los contenidos de aquellos vértices que están a distancia mínima `n` del vértice inicial.

```
método <V,A> distN(Grafo<V,A> g, V vInicial, entero n)
    retorna List<V>
```

Examen de Estructuras de Datos y Algoritmos
(Ingeniería Informática)

Febrero 2010

Segunda parte (50% nota del examen)

4) Se dispone de una implementación de un árbol genealógico realizada con cursores. La clase almacena un `ArrayList` de personas y cada persona tiene como atributos su nombre, y cursores a su padre y su madre (cada cursor indica la casilla del `ArrayList` donde se encuentra el padre o madre, respectivamente). Escribir un método con la cabecera que se indica abajo que retorne una lista de los hermanos de una persona. Retornará `null` si la persona no existe en el árbol genealógico. Se entiende que una persona es hermana de otra si tiene el mismo padre o la misma madre.

```
import java.util.*;

public class ArbolGenealogico
{
    // atributos del arbol genealógico
    ArrayList<Persona> lista=new ArrayList<Persona> ();

    // clase interna Persona
    private static class Persona {
        String nombre;
        int padre;
        int madre;
    }

    public List<String> hermanos(String nombre) {...}
    .. resto de los métodos

}
```

5) Escribir el pseudocódigo de un método que permita en un árbol que sigue la interfaz de los árboles vista en clase retornar un iterador colocado en el primer antecesor común de dos nudos (`nudoA` y `nudoB`) referidos por dos iteradores que se pasan como parámetros. Para ello, a partir del `nudoA` se obtiene la lista de sus antecesores en orden ascendente. Posteriormente se recorren los antecesores de `nudoB`, también en orden ascendente, y el primero de ellos que se encuentre en la lista es el que corresponde al resultado. La cabecera del método será:

```
método <E> primerAntecesorComun
  (Arbol<E> a, IteradorDeArbol<E> nudoA,
   IteradorDeArbol<E> nudoB) retorna IteradorDeArbol<E>
```

6) Se dispone de la clase `ColaEnlazada` que implementa una cola usando una lista enlazada simple. Los elementos se insertan por el final y se eliminan por el principio. Se muestra parte de la implementación abajo. Se pide implementar el método estático `concatena`, que permite añadir al final de la cola `c1` todos los elementos de la cola `c2`, respetando su orden. Al finalizar el método la cola `c2` se dejará vacía. Hacer una implementación que sea O(1).

```
public class ColaEnlazada<E> extends AbstractQueue<E>
```

```
{

    // atributos de la cola
    private Celda<E> principio,fin;
    private int num;

    // clase privada que define la celda
    private static class Celda<E> {
        E contenido;
        Celda<E> siguiente;

        Celda(E cont) {
            contenido=cont;
        }
    }

       public static <E> void concatena(ColaEnlazada<E> c1,
ColaEnlazada<E> c2) {...}

   ... resto de los métodos
}
```