Examen de Estructuras de Datos y Algoritmos
(Ingeniería Informática)

Septiembre 2008

Primera parte (50% nota del examen)

1) Se dispone de la clase `Articulo` indicada abajo, que representa artículos en venta en una tienda. Cada artículo tiene un nombre y un precio. Queremos hacer un método Java que tenga como parámetros:

   almacen: una lista de objetos de la clase Articulo, con todos los artículos de la tienda
   pedido: un conjunto de Strings, con nombres de artículos de un pedido

   El método debe obtener y retornar una cola de prioridad con aquellos artículos de la lista `almacen` cuyo nombre coincida con alguno de los almacenados en el conjunto `pedidos`. El motivo de usar una cola de prioridad es que así aparecerán los artículos ordenados por su precio.

```
import java.util.*;
/**
 * Clase que representa un articulo en venta
 */
public class Articulo implements Comparable {

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

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


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

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

}
```

2) Suponer que usamos un mapa para guardar un listín telefónico. En este mapa las claves son Strings que contienen nombres de personas y los valores son Strings que contienen números de teléfono.

   Se desea hacer un método Java cuya interfaz se muestra abajo y al que se le pasan como parámetros dos listines telefónicos. El método debe buscar cada persona del listín `nuevo` en el listín `viejo`, y si la encuentra cambiarle el numero de teléfono asignado, poniendo en el listín `viejo` el número que aparece en el `nuevo`. Además, debe retornar un conjunto que contenga las personas del listín `nuevo` que no estaban en el `viejo`.

   Indicar la eficiencia del método creado.

```
public static Set<String> actualiza
  (Map<String,String> viejo, Map<String,String> nuevo)
{
      return null;
}
```

3) Un grafo dirigido representa la conectividad entre computadores de una red. Cada vértice contiene un `String` que identifica el nombre de un computador. Cada arista de un computador a otro indica que hay una conexión directa del primero al segundo. Las aristas no tienen pesos.

   Se desea hacer un método Java con la interfaz indicada abajo para añadir al grafo las conexiones de un conjunto de parejas de computadores, si no existen previamente. Al método se le pasa el grafo y un conjunto de objetos de la clase `Pareja`. Lo que hace es añadir al grafo para cada pareja de ese conjunto una arista del primer elemento de la pareja al segundo, siempre que no exista ya en el grafo.

```
public class Pareja {
     // atributos públicos
     public String primero;
     public String segundo;
}
```

```
public static void insertaConexiones (Grafo<String> g, Set<Pareja> nueva)
```

Examen de Estructuras de Datos y Algoritmos
(Ingeniería Informática)

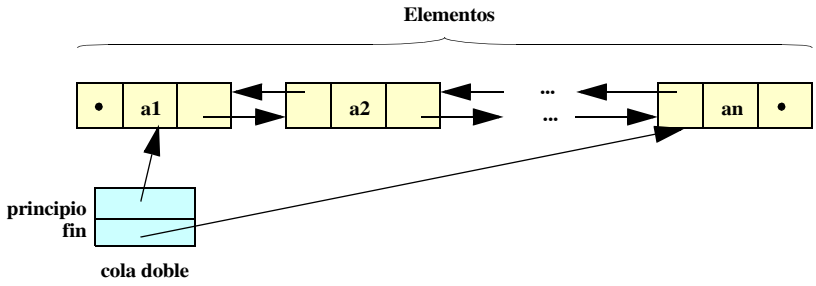
Septiembre 2008

Segunda parte (50% nota del examen)

4) Para una implementación de mapas de troceado abierto diseñar en pseudocódigo un método interno a la clase (con acceso por tanto a la implementación de la estructura de datos) que permita obtener el indice de ocupación medio de las casillas ocupadas. Este índice se define como la media de las longitudes de las listas del mapa que no estén vacías.

   Asimismo, razonar sobre la eficiencia de este método, según el tipo de implementación de la lista.

5) Suponer una cola doble, que es una estructura de datos como la de la figura, implementada mediante una lista doblemente enlazada, y en la que se pueden insertar y eliminar elementos por cualquiera de sus dos extremos (pero no por el medio).



   Se pide diseñar en pseudocódigo un método interno a la cola doble que responda a la interfaz indicada abajo y que permita introducir un elemento por el principio o el final de la cola doble (si `alPrincipio` vale `true` se mete por el principio, y si no por el final).

   método inserta(Elemento e, booleano alPrincipio)

6) Hacer un método Java que responda a la interfaz que se muestra abajo y permita comprobar si dos nudos `a` y `b` de un árbol de Strings diferentes tienen o no un antecesor común entre los descendientes del nudo `origen`. Para ello se puede ir recorriendo el árbol hacia la raíz desde cada uno de los dos nudos `a` y `b`, y meter cada nudo por el que pasemos en un conjunto. Si en alguna ocasión metemos el mismo nudo dos veces, quiere decir que hay un antecesor común, y en ese caso hay que comprobar que éste es descendiente del nudo `origen`. Si llegamos al nudo `origen` o a la raíz cesamos el recorrido por ese camino. El método acaba si encontramos un antecesor común, o si por ambos caminos llegamos al nudo `origen` o a la raíz sin encontrar el antecesor. Si `a` y `b` son el mismo nudo, retornar `false`. Los nudos están representados por un iterador de árbol.

```
public static boolean mismoAntencesor
    (Arbol<String> arbol,
     IteradorDeArbol<String> a,
     IteradorDeArbol<String> b,
     IteradorDeArbol<String> origen)
```