Estructuras de Datos

Grado en Ingeniería en Informática

Examen de Febrero de 2012

Nombre: Grupo:

Pregunta 1:

Contestar V/F las siguientes cuestiones, teniendo en cuenta que una respuesta incorrecta invalida una correcta.

[V] Todo árbol binario puede representarse mediante un vector. Si el vector está compacto, entonces el árbol es completo.

[V] Dos árboles ABB equivalentes pueden tener diferente altura, diferente raíz, y diferentes hojas.

[F] La implentación del problema de Josefo con una lista doblemente enlazada circular posee ciertas ventajas con respecto a una lista simplemente enlazada y circular.

[F] Si el conjunto de enteros A={1,2,4,8,16,32,64,128,256,512,1024,2048} es representado mediante un conjunto de bits necesitamos un vector con mas de 2000 bytes para almacenarlo.

[F] Un heap permite obtener el dato con menor prioridad en O(1).

[F] La dispersión abierta tiene una implementación más sencilla y un rendimiento más predecible que la dispersión cerrada.

[F] Si la posición de una estrella en el firmamento queda determinada por su ascensión y declinación, un árbol AVL es una estructura de datos adecuada para localizar aquellas situadas en una ventana rectángular de la bóveda celeste.

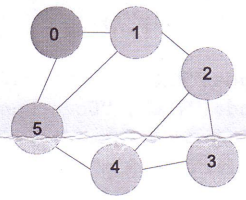
[V] Una tabla de dispersión construida correctamente permite localizar un dato por su clave de manera más eficiente que un árbol AVL.

[F] Cuando un índice simple es modificado, los cambios son inmediatamente reflejados en el fichero donde se guarda el índice.

[F] Un árbol B utiliza rotaciones para mantener el equilibrio en altura.

Pregunta 2:

Listar el conjunto de nodos cuando en el siguiente grafo se realiza un recorrido en profundidad (DFS) y en anchura (BFS), partiendo siempre del nodo 0.



Pregunta 3:

a) Implementad la función int ABB<T>::altura( ); que devuelva la altura de un árbol binario de búsqueda.

b) La clase ListaSECircular<T> es una lista simplemente enlazada y circular. Implementad el constructor ListaSECircular<T>::ListaSECircular (const ListaEnlazada<T> &l); para que construya dicha lista a partir de los datos de una lista simplemente enlazada (no circular).

Pregunta 4:

Después de años de resistirse, los ingenieros de Microsoft han reconocido la utilidad de un sistema de instalación/desinstalación de módulos similar a los existentes en Linux (apt-get/rpm). Finalmente han implementado un prototipo con la estructura mostrada por diagrama UML adjunto.

La clase *Modulo* representa cualquier modulo/aplicación instalable en el sistema operativo: VLC, Winzip, GIMP, DriverNVidia, CodecMpeg4, etc. Cada módulo tiene asociado un identificador único y puede estar instalado o no en el sistema (atributo *instalado*). Por otro lado es frecuente que un módulo dependa de otros como queda reflejado en el UML, es decir, que la instalación de un módulo requiera la instalación de otros módulos para su correcto funcionamiento. Por ejemplo: VLC depende de DriverNVidia y de CodecMpeg4. El atributo *contadorUso* indica el número de módulos dependientes instalados que están usando a este módulo actualmente. Las operaciones de la clase más reseñables son:

- *instalar()*: imprime el mensaje "Instalando módulo: <identificador>..." y pone el atributo instalado a true.
- *desinstalar()*: imprime el mensaje "Desinstalando módulo: <identificador>..." y pone el atributo instalado a false.
- *incrementarUso()* y *decrementarUso()* incrementan o decrementan el valor del atributo *contadorUso*.

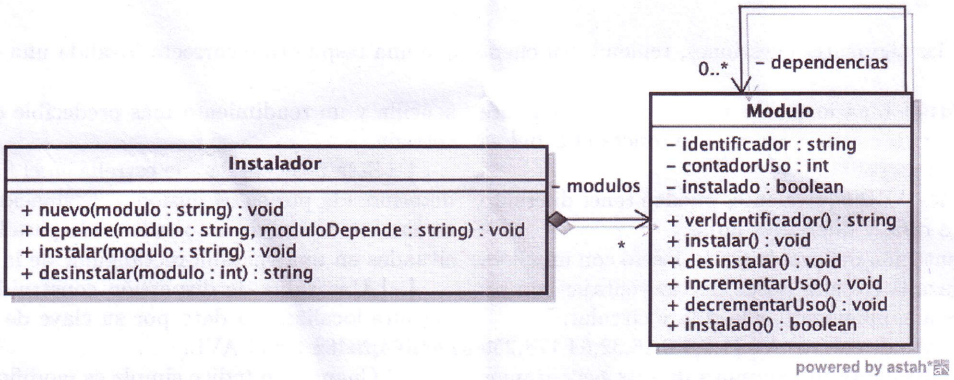
La clase *Instalador* es la responsable de gestionar los módulos del sistema. Las operaciones *nuevo()* y *depende()* permiten definir nuevos módulos en el sistema, y añadir dependencias entre módulos. La operación instalar de un módulo funciona de la siguiente manera:

1. Si el módulo ya estaba instalado se lanzaría una excepción.
2. Se instalan todos los módulos de los que depende el módulo que no estén instalados. Tanto si estaban instalados como si no, se incrementa su contador de uso.
3. Se instala el módulo.

Y la operación desinstalar de un módulo funcionaría como sigue:

1. Si el módulo está siendo usado por otros, no se puede desinstalar, y por tanto se lanza una excepción.
2. Se decrementa el contador de uso de todos los módulos de los que depende. Si el contador de uso de alguno es cero (nadie lo usa ya) entonces se desinstala.
3. Se desinstala el módulo.

Se pide la implementación de las clases descritas, eligiendo las estructuras de datos más adecuadas. Es posible añadir alguna operación adicional no contemplada en el diagrama.



A continuación se muestra un ejemplo de uso y la salida generada:

```
Instalador instalador;
instalador.nuevo("DriverNVidia");
instalador.nuevo("CodecMpeg4");
instalador.nuevo("VLC");
instalador.nuevo("Winzip");

instalador.depende("VLC", "DriverNVidia");
instalador.depende("VLC", "CodecMpeg4");

instalador.instalar("Winzip");
instalador.instalar("VLC");
instalador.desinstalar("Winzip");
```

Salida:

```
Instalando módulo Winzip...
Instalando módulo DriverNvidia...
Instalando módulo CodecMpeg4...
Instalando módulo VLC...
Desinstalando módulo Winzip...
```