Estructuras de Datos y Algoritmos

Grados en Ingeniería Informática, de Computadores y del Software

Segundo Examen Parcial, 11 de septiembre de 2013.

1. (4 puntos)

Te han contratado para implementar un sistema de gestión de barcos de pesca. Cada barco tiene una bodega de carga donde los pescadores que van en el barco van depositando las capturas que realizan, anotando siempre la especie del pez y su peso. Cuando el barco llega a puerto, cada pescador se lleva a casa lo que ha pescado de cada especie.

Las operaciones públicas del TAD *BarcoMatic* son:

- *nuevo*: Crea una nueva instancia de la estructura *BarcoMatic*, recibiendo como argumento un el peso máximo (en kilos) admitido en la bodega.
- *altaPescador*: Da de alta a un pescador, identificado por su nombre. No devuelve nada.
- *nuevaCaptura*: Registra que un pescador (que debe estar registrado) ha pescado un ejemplar de tantos kilos de una especie concreta. Las especies y los pescadores se indican mediante sus nombres, y el peso en kilos se especifica mediante un número. Si el peso de la captura, añadido a la bodega, haría que la bodega excediese su capacidad, esta operación debe fallar.
- *capturasPescador*: Recibe el nombre de un pescador, y devuelve una lista de parejas especie-kilos. Si, para una especie dada, el pescador no ha pescado nada, no la debes incluir en la lista devuelta. Puedes asumir la existencia de un TAD *Pareja<A, B>* similar al visto en clase.
- *kilosEspecie*: Recibe el nombre de una especie, y devuelve el número total de kilos de esa especie pescados, sumando las capturas de todos los pescadores.
- *kilosPescador*: Recibe el nombre de un pescador, y devuelve el número total de kilos que ha pescado, sumando todas las especies.
- *bodegaRestante*: Devuelve el número de kilos restantes en la bodega.

Se pide: prototipos de las operaciones públicas, representación eficiente del TAD (basada en tipos vistos durante el curso), coste de cada operación (*especificando qué representa la N en cada caso concreto*) e implementación en C++ de todas las operaciones.

2. (3 puntos)

Dado un árbol binario, se llama *altura mínima* del árbol a la menor de sus distancias desde la raíz a un subárbol vacío. Se puede especificar formalmente mediante las ecuaciones:

$$\begin{aligned} hmin(avacio) &= 0 \\ hmin(cons(i, x, d)) &= 1 + min(hmin(i), hmin(d)) \end{aligned}$$

Decimos que un árbol binario es *zurdo* si o, bien es vacío, o si no lo es, ambos hijos son zurdos y la altura mínima del hijo izquierdo nunca es menor que la del hijo derecho.
Se pide:

1. Dibujar algunos ejemplos no triviales de árboles zurdos y no zurdos
2. Implementar en C++ una función, de coste lineal con el número de nodos, que dado un árbol binario, determine si es o no zurdo. Justificar dicho coste.

3. (3 puntos)

Podemos representar la discretización de una función $f(x)$ mediante una tabla (*Tabla<int,float>*) que asocia a ciertos valores enteros de abscisa $(x_1, x_2, \ldots)$ sus correspondientes valores reales de ordenada $(f(x_1), f(x_2), \ldots)$. Suponiendo que:

- La tabla sólo contiene valores de abscisa positivos y consecutivos comenzando en $x = 1$.
- La función $f(x)$ discretizada es estrictamente creciente $(x_1 < x_2 \rightarrow f(x_1) < f(x_2))$.

Se pide implementar una función con coste $\mathcal{O}(\log n)$ que, dada una tabla como la anterior y un valor de ordenada y, devuelva la abscisa entera x asociada a y si el punto (x,y) existe en la tabla, y -1 si no existe.

```
int buscaX(const Tabla<int,float> &f, float y)
```