Estructuras de Datos y Algoritmos – IIC2133

I1 - pauta

31 agosto 2011

1. Supón que tenemos k fuentes de datos. Los datos de cada fuente vienen ordenados, p.ej., de mayor a menor. Queremos enviar la totalidad de los datos recibidos por un único canal de salida, también ordenados de mayor a menor. El dispositivo electrónico que debe hacer esta *mezcla ordenada de datos* tiene una capacidad limitada de memoria.

[**4 pts.**] Da un algoritmo para este dispositivo, que le permita hacer su tarea empleando un arreglo a de k casilleros, en que cada casillero tiene dos campos: en $a[j].data$ se puede almacenar un dato, y en a[j].num se puede almacenar un número entero entre 1 y k. Para recibir un dato desde la fuente i, se ejecuta $receive[i]()$, y para enviar un dato x por el canal de salida, se ejecuta $send(x)$.

La idea es armar un (max-)heap binario de tamaño k, inicialmente con el primer dato (el mayor) de cada fuente, de modo que en la raíz quede el mayor de todos los datos:

```
for (i = 0; i < k; i = i+1)
    x.data = receive[i]()
    x.num = i
    insertObject(x)
```

A continuación, hay que ir sacando el dato que está en la raíz, enviándolo por el canal de salida, y reemplazándolo en el heap por el próximo dato recibido de la misma fuente de donde provenía el que salió:

```
while ( true )
    x = xMax()
    i = x.num
    send(x.data)
    x.data = receive[i]()
    insertObject(x)
```

[**2 pts.**] Si la cantidad total de datos en las k fuentes es n, ¿cuál es la cantidad total de pasos que ejecuta tu algoritmo, en notación $O(\,)$?

$O(n \log k)$, ya que el `for` ejecuta k operaciones `insertObject(x)`, c/u de las cuales ejecuta un número de pasos proporcional a log k, lo que da un subtotal de $k \log k$); y el `while` ejecuta $n–k$ operaciones *insertObject(x)*, c/u de las cuales nuevamente ejecuta un número de pasos proporcional a log k, lo que da otro subtotal de $(n–k)\log k$.

2. En el caso de hashing con direccionamiento abierto, supón que tienes una tabla de tamaño $T = 10$ y las claves A_5, A_2, A_3, B_5, A_9, B_2, B_9, C_2, en que K_i significa que al aplicar la función de hash h a la clave, obtenemos la posición i. Muestra qué ocurre al insertar las claves anteriores, en el orden en que aparecen, en los siguientes casos:

a) [**3 pts.**] Si usamos revisión lineal, de modo que la posición intentada es $(h(K) + i) \bmod T$.

b) [**3 pts.**] Si usamos revisión cuadrática, de modo que la posición intentada es $h(K)$, $h(K)+1$, $h(K)–1$, $h(K)+4$, $h(K)–4$, $h(K)+9$, ..., $h(K)+(T–1)^2/4$, $h(K)–(T–1)^2/4$.

	a)	**b**)
0	B_9	B_9
1		B_2
2	A_2	A_2
3	A_3	A_3
4	B_2	
5	A_5	A_5
6	B_5	B_5
7	C_2	
8		C_2
9	A_9	A_9

3. Un árbol binario de búsqueda (ABB) se dice **balanceado** si la diferencia en altura de ambos subárboles de cualquier nodo es cero o uno (p.ej., un árbol AVL). Un ABB se dice **perfectamente balanceado** si es balanceado y todas sus hojas están en un mismo nivel o, a lo más, en dos niveles (p.ej., un *heap* binario). Da un algoritmo para convertir cualquier ABB en un ABB perfectamente balanceado.

Para esto, considera un ABB totalmente desbalanceado, convertido en una lista ligada hacia la derecha, con 2^k-1 nodos, para algún entero k. Partiendo desde la raíz, baja por la lista y haz una rotación simple a la izquierda cada dos nodos (la primera rotación es del hijo derecho de la raíz con respecto a la raíz; la segunda rotación involucra a los dos nodos siguientes en la lista; y así sucesivamente).

¿Qué ha pasado con los desbalances cuando llegas abajo? ¿Qué tendrías que hacer a continuación para convertir finalmente el árbol en uno perfectamente balanceado y completo? ¿Cómo manejarías el caso de un árbol que no puede ser completo, es decir, que su número de nodos no puede escribirse como 2^k-1?

La sugerencia que parte en el segundo párrafo es como la 'segunda' pare del algoritmo pedido. La primera parte consiste en convertir el árbol original en una lista ligada hacia la derecha; esto lo vimos en la ayudantía de 25/8:

`while (` haya un nodo x con un hijo izquierdo y `)`
    hacer una rotación simple a la derecha, de y con respecto a x

Como cada rotación simple a la derecha reduce en uno el número de hijos izquierdos en el árbol, a lo más $n-1$ rotaciones son necesarias para convertir todo el árbol en una lista ligada hacia la derecha.

Una vez hecho esto, se aplica el algoritmo sugerido. Las rotaciones sugeridas, a lo largo de la lista y 'nodo por medio', van reduciendo paulatinamente el desbalance: las primera rotación reduce el desbalance de la raíz en dos; las siguientes, en uno cada una. Así, al llegar abajo, el desbalance de la raíz se ha reducido de $n-1$ a $n-3-\lfloor n/2 - 1 \rfloor$. También se han reducido los desbalances de los subárboles. Lo que hay que hacer a continuación es repetir el proceso, desde la raíz, por la rama más a la derecha del árbol; el número rotaciones a realizar en esta segunda pasada va a ser la mitad de las de la primera pasada, y el árbol va a quedar aún un poco mejor balanceado. Y así sucesivamente, hasta formar un árbol completo (ya que n es de la forma 2^k-1) perfectamente balanceado.

Si el árbol original no es completo, hay que tratar el número de nodos 'sobrantes' como casos especiales, p.ej., haciendo ese número de rotaciones a la izquierda justo antes de aplicar el algoritmo sugerido.

4. Uno de los ejercicios propuestos en el texto de Cormen et al. afirma que si un árbol rojo-negro de más de un nodo ha sido construido sólo a través de operaciones de inserción (sin eliminaciones), entonces el árbol tendrá al menos un nodo rojo. Muestra con un ejemplo que un árbol rojo-negro de más de un nodo puede tener sólo nodos negros si su construcción ha incluido eliminaciones.

Supongamos un árbol inicialmente vacío, al que insertamos las claves 3, 2 y 4, en este orden: 3 queda en la raíz, y es negro, y 2 y 4 son sus hijos izquierdo y derecho, respectivamente, ambos rojos. Si ahora insertamos la clave 1, esta va a parar como hijo izquierdo de 2, también rojo. Al ser rojos 2 y su hijo 1, hay que hacer algo; en este caso, como el hermano de 2, 4, también es rojo, intercambiamos colores por niveles: pintamos negros 2 y 4 y pintamos rojo 3. Pero como 3 es la raíz, lo volvemos a pintar negro. En este momento, sólo 1 es rojo.

Si ahora eliminamos 1, simplemente eliminamos una hoja roja, por lo que no es necesario hacer ninguna otra operación en el árbol, que así queda sólo con tres nodos negros.