Diseño de algoritmos

Jesús Bermúdez de Andrés. UPV-EHU
Guías para la solución de ejercicios: Montículos

Curso 2008-09

1. Escriba un algoritmo eficiente para la operación RETIRAR_DE_MONTÍCULO(M, i), que retire el elemento del nodo número i del montículo M.

   Solución:

   proc RETIRAR_DE_MONTÍCULO(M, i)
   $M(i) \leftarrow M(M.fin)$
   $M.fin \leftarrow M.fin - 1$
   HUNDIR(M, $M.fin$, i)

   Obviamente, este algoritmo es del mismo orden que HUNDIR(M, $M.fin$, i). En el peor caso $O(\lg n)$, siendo n el número de elementos de M.

2. Considere la ejecución del algoritmo ORDEN_POR_MONTÍCULO($T(1..n)$) sobre la tabla de entrada $T = [1, 3, 2, 5, 6, 7, 4, 10, 9, 8]$.

   a) ¿Cual es el estado de T al finalizar CREAR_MONTÍCULO(T)?

   b) ¿Cual es el estado de T después de que $8, 9$ y 10 hayan quedado colocados en su posición de orden y se haya restituido la propiedad de montículo?

   Solución:

   a) Siendo $T = [1, 3, 2, 5, 6, 7, 4, 10, 9, 8]$, al finalizar CREAR_MONTÍCULO(T), el estado de T es $[10, 9, 7, 5, 8, 2, 4, 1, 3, 6]$.

   b) Después de que $8, 9$ y 10 han quedado colocados en su posición de orden y se ha restituido la propiedad de montículo, el estado de T es $[7, 6, 4, 5, 3, 2, 1, 8, 9, 10]$.

3. Escriba un algoritmo que mezcle k listas ordenadas (de menor a mayor) en una sola lista ordenada que finalmente tendrá n elementos (la reunión de todos los elementos de las k listas). El algoritmo debe ejecutarse en tiempo $O(n \lg k)$.

   Solución:
   Utilizaremos un montículo-min para ayudar en la mezcla de las k listas.

La idea básica del algoritmo es la siguiente: Tomamos el primer elemento de cada una de las listas y construimos un montículo-min M de k elementos. Cada elemento llevará asociado un número que indica el número de lista de la que proviene.

Mientras haya elementos en el montículo M:

- retiramos la raíz del montículo, anotando la lista de la que proviene ese elemento
- añadimos ese elemento al final de la lista que será el resultado
- ponemos en la posición de la raíz el siguiente elemento (si lo hay) de la lista anotada. En caso de que no queden elementos en esa lista, ponemos en la posición de la raíz el 'último' elemento del montículo
- hundimos el valor situado en la raíz

Este algoritmo necesita $\Theta(k)$ para construir el montículo M. Después realiza un bucle de n iteraciones en el que, en cada iteración, realiza un trabajo de $O(\lg k)$ para manipular el montículo y colocar el siguiente elemento en la lista resultado. Por lo tanto, resultará un algoritmo de $O(n \lg k)$.

4. La operación NEXT() obtiene, sucesivamente, un número de una colección de números enteros positivos menores que 10^6.

   Complete el programa siguiente, para que calcule en el montículo $M(1.,10)$ los diez menores números de una colección de n números que van siendo obtenidos sucesivamente por la operación NEXT(). Debe decidir si interesa un montículo-max o un montículo-min.

```
for i in 1...10 loop M(i) ← ...
for i in 1...n loop
  x ← NEXT()
  ...
  ...
end loop
return M(1..10)
```

   Solución:
   Interesa que $M(1.,10)$ sea un montículo-max, en el que iremos guardando los 10 menores números de la colección que hayan sido ofrecidos por NEXT() hasta el momento.

```
for i in 1.,10 loop M(i) ← 1 + 10^6
for i in 1..n loop
  x ← NEXT()
  if x < RAÍZ(M) then
    M(1) ← x
    HUNDIR(M, 10, 1)
end loop
return M(1..10)
```

5. Analice la siguiente versión del algoritmo HUNDIR para montículos-max binarios

```
proc HUNDIR(M, i)
    elMayor ← i
    l ← 2i
    r ← 2i + 1
    if l ≤ M.fin ∧ M(l) > M(i) then elMayor ← l
    if r ≤ M.fin ∧ M(r) > M(elMayor) then elMayor ← r
    if elMayor ≠ i then
        INTERCAMBIAR(M(i), M(elMayor))
        HUNDIR(M, elMayor)
```

Solución:
Obsérvese que, en el peor caso, el subárbol izquierdo de un montículo tiene (casi) $2n/3$ de los n elementos del montículo. Por lo tanto, en el caso peor, la función de coste de esa versión de HUNDIR(M, i) tiene la forma de la siguiente recurrencia: $f(n) = f(2n/3) + \Theta(1)$, que resolviendo resulta $f(n) = \Theta(\lg n)$.

6. Escriba y analice un algoritmo que construya un montículo-max utilizando el procedimiento FLOTAR (y sin utilizar el procedimiento HUNDIR). Analice su algoritmo.

Solución:

```
proc CONSTRUIR_MONT(M)
    for i in 2 . . . n loop
        FLOTAR(M, i)
```

Sabemos que FLOTAR(M, i) es de $O(\lg i)$. Por lo tanto, el coste de CONSTRUIR_MONT(M), con un montículo de n elementos será $\sum_{i=2}^{n} \lg i = O(n \lg n)$.