Divide & Conquer

Elemento mayoría

Enunciado

Sea A un arreglo de números naturales de tamaño n, se dice que x es un elemento mayoría de A si aparece más de la mitad de las veces en el arreglo. Formalmente,

$$em(A, x) = \#apariciones(A, x) > \frac{n}{2} \tag{1}$$

$$apariciones(A, x) = \{i | A_i = x\} \tag{2}$$

Se pide dar un algoritmo que dado un arreglo A devuelva, en caso de existir, su elemento mayoría.

Propiedad

Antes de resolver el problema vamos a demostrar una propiedad:

$$em(A, x) \Rightarrow (em(A_{1..\lfloor n/2 \rfloor}, x) \vee em(A_{\lfloor n/2 \rfloor + 1..n}, x)). \tag{3}$$

En otras palabras, si un elemento x es mayoría en A entonces es mayoría en alguna de las mitades de A.

Demostración

Sea x un elemento mayoría en A (arreglo de tamaño n), k la cantidad de apariciones de x en A, k_{izq} y k_{der} la cantidad de apariciones de x en la primera y la segunda mitad de A. Formalmente,

$$k = \#apariciones(A, x) \tag{4}$$

$$m = \lfloor n/2 \rfloor \tag{5}$$

$$k_{izq} = \#apariciones(A_{1..m}, x) \tag{6}$$

$$k_{der} = \#apariciones(A_{m+1..n}, x) \tag{7}$$

Supongamos que x no es mayoría en ninguna de las dos mitades, entonces $k_{izq} \leq \frac{m}{2}$ y $k_{der} \leq \frac{n-m}{2}$. Entonces $k = k_{izq} + k_{der} \leq \frac{m}{2} + \frac{n-m}{2} \leq \frac{n}{2}$, pero k era elemento mayoría en A, por lo tanto vale $k > \frac{n}{2}$. Absurdo, viene de suponer que x no es mayoría en ninguna de las dos mitades.

Algoritmo

Vamos a resolver este problema utilizando la técnica divide & conquer.
Como vimos antes, si un arreglo tiene elemento mayoría entonces tiene que serlo también de alguna mitad, por lo tanto, primero vamos a resolver las dos mitades y obtener los elementos mayoría de las mismas (si es que existen). Luego, vamos a contar cuántas veces aparece cada uno en el arreglo original. Si alguno de ellos es mayoría también allí, encontramos la respuesta, y si ninguno lo es entonces concluímos que no existe tal elemento.

```
EM(A, out x):
  n = |A|

  // Caso base
  Si n = 1:
    x = A[1]
    devolver true

  // Caso recursivo
  Si n > 1:
    // Resuelvo las dos mitades.
```

```
    m = floor(n/2)
    hayEmIzq = EM(A[1..m], xIzq)
    hayEmDer = EM(A[m+1..n], xDer)

    // Chequeo si alguno de los resultados es mayoría en A.
    Si hayEmIzq & CantAp(A, xIzq) > n/2:
      x = xIzq
      devolver true
    Si hayEmDer & CantAp(A, xDer) > n/2:
      x = xDer
      devolver true
    // No hay elemento mayoría.
    Si no:
      devolver false

CantAp(A, x):
 contador = 0
  Para i = 1 hasta |A|:
    si A[i] = x:
      contador = contador + 1
  devolver contador
```

Complejidad

Por simplicidad, vamos a calcular la complejidad del algoritmo asumiendo que n es potencia de dos.
Como podemos ver en el código, CantAp es una función que recorre una vez cada elemento del arreglo A por lo tanto tiene orden O(n).
La función principal, EM, es recursiva por lo tanto tiene una ecuación de recurrencia:

$$T(n) = 2\ T(n/2)\ +\ O(n) \quad si\ n\ >\ 1 \tag{8}$$

$$T(n) = O(1) \quad si\ n\ =\ 1 \tag{9}$$

El caso base es de orden constante porque simplemente devuelve el primer elemento. El paso recursivo, llama a la función con cada una de sus mitades (de tamaño $n/2$) y luego hace a lo sumo dos llamados a la función CantAp que tiene orden lineal.
La ecuación de recurrencia es igual a la de mergesort, por lo tanto, podemos concluir que este algoritmo también es O(n log n).