Solución de recurrencias

En los algoritmos de búsqueda y ordenamiento sobre un arreglo, existen dos operaciones que generalmente dominan el tiempo de ejecución total: las comparaciones entre elementos y las asignaciones entre elementos. A su vez, muchos de estos algoritmos son recursivos pero fáciles de análizar en su peor caso, por lo que se prestan para calcular exactamente cuántas de estas operaciones realizan.

---

subrutina BúsquedaBinaria(natural n, arreglo $a[n]$, natural b)
  si $n = 0$
    regresa falso
  sino si $b < a[\lfloor \frac{n}{2} \rfloor]$
    regresa BúsquedaBinaria($\lfloor \frac{n}{2} \rfloor, a[0 \ldots \lfloor \frac{n}{2} \rfloor - 1], b$)
  sino si $b > a[\lfloor \frac{n}{2} \rfloor]$
    regresa BúsquedaBinaria($n - \lfloor \frac{n}{2} \rfloor - 1, a[\lfloor \frac{n}{2} \rfloor + 1 \ldots n - 1], b$)
  sino
    regresa verdadero

El primer ejemplo que trabajaremos es el de búsqueda binaria. Para este ejemplo, nos concentraremos en calcular la cantidad de comparaciones realizadas en el peor caso (cuando lo que buscamos nunca lo encontramos) sobre un arreglo de $n = 2^k$ elementos. Recordemos que el algoritmo de búsqueda binaria primero compara el elemento buscado con el de enmedio del arreglo, repitiendo la búsqueda sobre la mitad izquierda o sobre la mitad derecha dependiendo del resultado de la comparación. Cuando n es par, no hay un elemento que esté exactamente a la mitad del arreglo, así que al tomar un elemento que esté aproximadamente enmedio y luego descartarlo, una de las dos mitades queda con un elemento menos que la otra. En el peor caso tendríamos hacer recursión sobre el lado más grande, el cual tiene exactamente $\frac{n}{2}$ elementos. Si $n = 2^k$ entonces el valor de n enviado en la recursión siempre es par hasta llegar a $n = 1$.

Definiremos $T(n)$ como la cantidad de comparaciones que búsqueda binaria realiza sobre un arreglo de n elementos. Entonces:

$$T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 + T(\lfloor \frac{n}{2} \rfloor) & \text{si } n > 0 \end{cases}$$

Una de las técnicas más empleadas para calcular una fórmula no recursiva para $T(n)$ es la técnica de sustitución repetida. En esta técnica, el valor de $T(n)$ se desarrolla recursivamente de forma repetida hasta que aparezca un patrón no recursivo. Después se deberá demostrar que el patrón en efecto es correcto, usualmente mediante inducción matemática. Desarrollando

$T(n)$ para búsqueda binaria obtenemos:

$$\begin{aligned} T(n) &= 1 + T\left(\frac{n}{2}\right) \\ &= 1 + 1 + T\left(\frac{n}{4}\right) = 2 + T\left(\frac{n}{4}\right) \\ &= 1 + 1 + 1 + T\left(\frac{n}{8}\right) = 3 + T\left(\frac{n}{8}\right) \\ &= 1 + 1 + 1 + 1 + T\left(\frac{n}{16}\right) = 4 + T\left(\frac{n}{16}\right) \\ &\dots \end{aligned}$$

El patrón que aparece es $T(k) = \log_2(k) + T(\frac{n}{k})$ y la fracción se vuelve 1 para $T(n) = \log_2(n) + T(\frac{n}{n}) = \log_2(n) + T(1)$, donde al desarrollar tenemos $T(n) = \log_2(n) + 1 + T(0) = \log_2(n) + 1$. Con esto podemos proponer la siguiente definición alternativa de $T(n)$:

$$T(n) = \begin{cases} 0 & \text{si } n = 0 \\ \log_2(n) + 1 & \text{si } n > 0 \end{cases}$$

Aunque el patrón es correcto, debemos demostrarlo formalmente y lo haremos por inducción.

Demostración. Claramente, las definiciones coinciden para $T(0)$. Para el caso recursivo, demostraremos primero el caso $T(1)$. Tenemos que $T(1) = 1 + T(0) = 1$ según la definición original, mientras $T(1) = \log_2(1) + 1 = 1$ según la definición propuesta. Ahora supondremos que $T(\frac{n}{2})$ es correcta y demostraremos que $T(n)$ también lo es. Tenemos que $T(n) = 1 + T(\frac{n}{2})$ y eso es igual a $1 + 1 + \log_2(\frac{n}{2})$ por la hipótesis de inducción. Desarrollando tenemos $T(n) = 1 + \log_2(2) + \log_2(\frac{n}{2}) = 1 + \log_2(\frac{2n}{2}) = 1 + \log_2(n)$. □

El algoritmo de ordenamiento por mezcla es un algoritmo que primero ordena cada mitad de un arreglo de n elementos de forma recursiva y luego los mezcla de forma ordenada. Todas las comparaciones y asignaciones de elementos se realizan durante la mezcla, pero no siempre se hacen la misma cantidad de comparaciones. Supongamos que $n = 2^k$ y que la mezcla toma dos arreglos de $\frac{n}{2}$ elementos cada uno:

```
subrutina OrdenamientoPorMezcla(natural n, arreglo a[n])
    si n ≤ 1
        regresa a
    sino
        //t1 y t2 son arreglos y |t1| y |t2| son sus tamaños
        t1 = OrdenamientoPorMezcla(⌊n/2⌋, a[0 ... ⌊n/2⌋ − 1])
        t2 = OrdenamientoPorMezcla(n − ⌊n/2⌋, a[⌊n/2⌋ ... n − 1])
        regresa MezclaOrdenada(|t1|, t1, |t2|, t2)
```

```
subrutina MezclaOrdenada(natural n, arreglo a[n], natural m, arreglo b[m])
    r ← [ ], i ← 0, j ← 0
    mientras i < n y j < m
        si a[i] < b[j]
            r ← (r)(a[i]) //pegamos a r el elemento a[i]
            i ← i + 1
        sino
            r ← (r)(b[j]) //pegamos a r el elemento b[j]
            j ← j + 1
    regresa (r)(a[i ... n − 1])(b[j ... m − 1]) //pegamos a r lo que sobre de a o b
```

- Cuando todos los elementos del primer arreglo son menores que los del segundo (o viceversa), entonces se necesitan $\frac{n}{2}$ comparaciones para darnos cuenta que el siguiente elemento a colocar en el resultado siempre proviene del mismo arreglo. Al acabarse este arreglo, los $\frac{n}{2}$ elementos del arreglo restante pueden colocarse al final sin usar comparaciones adicionales entre elementos. La cuenta de comparaciones para el algoritmo de ordenamiento por mezcla cuando la mezcla siempre entra al mejor caso es:

$$T_1(n) = \begin{cases} 0 & \text{si } n \leq 1 \\ \frac{n}{2} + 2T_1(\frac{n}{2}) & \text{si } n > 1 \end{cases}$$

- Cuando la mezcla se obtiene intercalando los elementos de ambos arreglos, entonces cada comparacion sirve para colocar alguno de los elementos en el resultado. Cuando sólo falta un elemento por considerar, éste ya no tiene con quién compararse. La situación descrita anteriormente realiza $n - 1$ comparaciones y es el peor caso del algoritmo de mezcla. La cuenta de comparaciones para el algoritmo de ordenamiento por mezcla cuando la mezcla siempre entra al peor caso es:

$$T_2(n) = \begin{cases} 0 & \text{si } n \leq 1 \\ n - 1 + 2T_2(\frac{n}{2}) & \text{si } n > 1 \end{cases}$$

- La mezcla siempre hace $\frac{n}{2} + \frac{n}{2} = n$ asignaciones porque todos los elementos de ambos arreglos deben colocarse en el resultado. La cuenta de asignaciones para el algoritmo de ordenamiento por mezcla es:

$$T_3(n) = \begin{cases} 0 & \text{si } n \leq 1 \\ n + 2T_3(\frac{n}{2}) & \text{si } n > 1 \end{cases}$$

Dado que $T_3(n)$ es la función que más crece de las tres, resolveremos ésta de forma exacta.

Sea $T = T_3$, obtenemos lo siguiente mediante la técnica de sustitución repetida:

$$\begin{aligned} T(n) &= n + 2T\left(\frac{n}{2}\right) \\ &= n + \frac{2n}{2} + 4T\left(\frac{n}{4}\right) = 2n + 4T\left(\frac{n}{4}\right) \\ &= 2n + \frac{4n}{4} + 8T\left(\frac{n}{8}\right) = 3n + 8T\left(\frac{n}{8}\right) \\ &= 3n + \frac{8n}{8} + 16T\left(\frac{n}{16}\right) = 4n + 16T\left(\frac{n}{16}\right) \\ &\dots \end{aligned}$$

El patrón que aparece es $T(k) = \log_2(k)n + kT(\frac{n}{k})$ y la fracción se vuelve 1 para $T(n) = \log_2(n)n + nT(\frac{n}{n})$, donde al desarrollar tenemos $T(n) = \log_2(n)n + nT(1) = \log_2(n)n$. Con esto podemos proponer la siguiente definición alternativa de $T(n)$:

$$T(n) = \begin{cases} 0 & \text{si } n \leq 1 \\ n\log_2(n) & \text{si } n > 1 \end{cases}$$

Aunque el patrón es correcto, debemos demostrarlo formalmente y lo haremos por inducción.

Demostración. Claramente, ambas definiciones coinciden para $T(0)$ y $T(1)$. Para el caso recursivo, demostraremos primero el caso $T(2)$. Tenemos que $T(2) = 2 + 2T(\frac{2}{2}) = 2$ según la definición original, mientras $T(2) = 2\log_2(2) = 2$ según la definición propuesta. Ahora supondremos que $T(\frac{n}{2})$ es correcta y demostraremos que $T(n)$ también lo es. Tenemos que $T(n) = n+2T(\frac{n}{2})$ y eso es igual a $n+2(\frac{n}{2}\log_2(\frac{n}{2}))$ por la hipótesis de inducción. Desarrollando tenemos $T(n) = n + n\log_2(\frac{n}{2}) = n(1 + \log_2(\frac{n}{2})) = n(\log_2(2) + \log_2(\frac{n}{2})) = n(\log_2(\frac{2n}{2})) = n\log_2(n)$ que es lo que queríamos demostrar. □

Ejercicios sugeridos

1. Resuelve exactamente la siguiente recurrencia:

$$T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 + T(n-1) & \text{si } n > 0 \end{cases}$$

2. Resuelve exactamente la siguiente recurrencia:

$$T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 + 2T(n-1) & \text{si } n > 0 \end{cases}$$

3. Resuelve exactamente la recurrencia T_2 de ordenamiento por mezcla para $n = 2^k$.