Guía complejidad algorítmica

Algoritmos y Estructuras de Datos (CB100) - FIUBA
Martin Klöckner - mklockner@fi.uba.ar

1. Un algoritmo divide un problema de tamaño n en dos subproblemas de tamaño $n/2$ con un costo constante en cada paso adicional, obtener la complejidad sabiendo que el costo real $T(n)$ es:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$$

Como la función de costo real $T(n)$ depende de la misma función $T(n)$ pero con una entrada menor $n/2$ se trata de un algoritmo recursivo o iterativo. Por el método de expansion, el orden se puede calcular como se muestra a continuación:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1) \tag{1}$$

Pero $T\left(\frac{n}{2}\right)$ resulta:

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + O(1) \tag{2}$$

Por lo tanto reemplazando (2) en (1) resulta:

$$\begin{aligned} T(n) &= 2 \cdot \left[2T\left(\frac{n}{4}\right) + O(1)\right] + O(1) \\ &= 4T\left(\frac{n}{4}\right) + 2O(1) + O(1) \\ &= 4T\left(\frac{n}{4}\right) + 3O(1) \end{aligned} \tag{3}$$

Pero de (1) se puede obtener $T\left(\frac{n}{4}\right)$:

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + O(1) \tag{4}$$

Y reemplazando en la ecuación (3):

$$\begin{aligned} T(n) &= 4 \cdot \left[2T\left(\frac{n}{8}\right) + O(1)\right] + 2O(1) + O(1) \\ &= 8T\left(\frac{n}{8}\right) + 4O(1) + 2O(1) + O(1) \\ &= 8T\left(\frac{n}{8}\right) + 7O(1) \end{aligned} \tag{5}$$

Se puede ver que luego de realizar k veces el mismo procedimiento resulta:

$$T(n) = 2^k \cdot T\left(\frac{n}{2^k}\right) + (2^k - 1) \cdot O(1) \tag{6}$$

Pero las iteraciones se terminan cuando el numero de entradas es 1, entonces en (6):

$$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow \boxed{log_2(n) = k} \tag{7}$$

Con (7) en (6) resulta

$$T(n) = n \cdot T(1) + (n - 1) \cdot O(1)$$

De la expresión anterior, suponiendo $T(1) = O(1)$ y aproximando, resulta

$$\begin{aligned} T(n) &= n \cdot O(1) + (n - 1) \cdot O(1) \\ &= O(n) + O(n) \Rightarrow \boxed{T(n) = O(n)} \end{aligned}$$

Es decir la complejidad resulta $O(n)$. El mismo problema se podría haber resuelto utilizando el teorema maestro para la reducción por division, el cual en primer lugar dice que dado un algoritmo con función de costo real $T(n)$ de la siguiente forma:

$$T(n) = \begin{cases} c \cdot n^k & \text{si } 1 \leq n < b \\ a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k & \text{si } n \geq b \end{cases}$$

La complejidad algorítmica, o la solución de la ecuación de recurrencia $T(n)$ resulta:

$$T(n) = \begin{cases} O(n^k) & \text{si } a < b^k \\ O(n^k \cdot log(n)) & \text{si } a = b^k \\ O(n^{log_b(a)}) & \text{si } a > b^k \end{cases}$$

En este caso se tiene que $T(n)$ es

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$$

Y es de la forma

$$a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k$$

Con $a = 2$, $b = 2$, $c = 1$ y $k = 0$. Por lo tanto, la solución de la ecuación de recurrencia, o la complejidad sale directamente, siendo esta:

$$T(n) = O(n^{log_b(a)}) = O(n^{log_2(2)}) \Rightarrow \boxed{T(n) = O(n)}$$

Resultando de igual forma que para la complejidad hallada mediante el método de expansión.

2. Un algoritmo busca un valor en un array ordenado reduciendo el problema a la mitad en cada paso, con un costo constante para la comparación

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

Para obtener la complejidad algorítmica, utilizo el método de expansión:

$$T(n) = T\left(\frac{n}{2}\right) + O(1) \tag{8}$$

Pero $T\left(\frac{n}{2}\right)$ resulta de reemplazar n por $\frac{n}{2}$ en $T(n)$, entonces:

$$T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + O(1) \tag{9}$$

Reemplazando (9) en (8) resulta

$$\begin{aligned} T(n) &= T\left(\frac{n}{4}\right) + O(1) + O(1) \\ &= T\left(\frac{n}{4}\right) + 2O(1) \end{aligned} \tag{10}$$

Realizando una iteración mas reemplazando $n = \frac{n}{4}$ en (8) resulta

$$T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + O(1) \tag{11}$$

Y con (11) en (10):

$$\begin{aligned} T(n) &= T\left(\frac{n}{8}\right) + O(1) + O(1) + O(1) \\ &= T\left(\frac{n}{8}\right) + 3O(1) \end{aligned} \tag{12}$$

Se puede ver que luego de k veces (o iteraciones) de repetir el procedimiento resulta

$$T(n) = T\left(\frac{n}{2^k}\right) + k \cdot O(1) \tag{13}$$

Pero las iteraciones se terminan cuando se llega al caso base, esto es, cuando $\frac{n}{2^k} = 1$ entonces:

$$\frac{n}{2^k} = 1 \Rightarrow 2^k = n \Rightarrow \boxed{k = log_2(n)} \tag{14}$$

Reemplazando (14) en (13)

$$T(n) = T(1) + log_2(n) \cdot O(1) \tag{15}$$

Suponiendo $T(1) = O(1)$ resulta que

$$T(n) = O(1) + log_2(n) \cdot O(1) \tag{16}$$

Por lo tanto la complejidad algorítmica resulta:

$$\boxed{T(n) = O(log(n))}$$

De manera análoga utilizando el teorema maestro, la expresión de $T(n)$ es de la forma

$$a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k$$

Con $a = 1$, $b = 2$, $c = 1$ y $k = 0$, por lo tanto aplicando el teorema maestro para la reducción por division resulta que la complejidad es:

$$T(n) = O(n^k \cdot log(n)) = O(n^0 \cdot log(n))$$

$$\Rightarrow \boxed{T(n) = O(log(n))}$$

3. Un algoritmo suma los elementos de una lista de n números dividiendo la lista en dos partes de igual tamaño.

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

En este caso aplico el teorema maestro en caso de poder aplicarse primero y luego comparo con el método de expansión. Se puede ver que $T(n)$ es de la forma

$$a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k$$

Con $a = 2$, $b = 2$, $c = 1$ y $k = 1$, por lo tanto se puede aplicar el teorema maestro y la complejidad algorítmica resulta;

$$T(n) = O(n^k \cdot log(n)) = O(n^1 \cdot log(n))$$

$$\Rightarrow \boxed{T(n) = O(n \cdot log(n))}$$

De manera análoga, utilizando el método de expansion:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \tag{17}$$

Pero $T\left(\frac{n}{2}\right)$ resulta de reemplazar $n = \frac{n}{2}$ en (17), de lo cual resulta

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + O\left(\frac{n}{2}\right) \tag{18}$$

Entonces de (18) en (17)

$$\begin{aligned} T(n) &= 2 \cdot \left[2T\left(\frac{n}{4}\right) + O\left(\frac{n}{2}\right)\right] + O(n) \\ &= 4 \cdot T\left(\frac{n}{4}\right) + 2 \cdot O\left(\frac{n}{2}\right) + O(n) \end{aligned} \tag{19}$$

(20)

Realizando una iteración más $T\left(\frac{n}{4}\right)$ resulta de reemplazar $n = \frac{n}{4}$ en (17), de lo cual resulta

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + O\left(\frac{n}{4}\right) \tag{21}$$

Y de (20) en (19)

$$\begin{aligned} T(n) &= 4 \cdot \left[2T\left(\frac{n}{8}\right) + O\left(\frac{n}{4}\right)\right] + 2 \cdot O\left(\frac{n}{2}\right) + O(n) \\ &= 8 \cdot T\left(\frac{n}{8}\right) + 4 \cdot O\left(\frac{n}{4}\right) + 2 \cdot O\left(\frac{n}{2}\right) + O(n) \end{aligned} \tag{22}$$

(23)

Es decir, luego de k iteraciones se llega a:

$$\begin{aligned} T(n) &= 2^k \cdot T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} 2^i \cdot O\left(\frac{n}{2^i}\right) \\ &= 2^k \cdot T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} O(n) \end{aligned} \tag{24}$$

(25)

Pero cuando se llega al caso base se cumple:

$$\frac{n}{2^k} = 1 \Rightarrow 2^k = n \Rightarrow \boxed{k = log_2(n)} \tag{26}$$

De (23) en (22):

$$\begin{aligned} T(n) &= n \cdot T(1) + \sum_{i=0}^{log_2(n-1)} O(n) \\ &= n \cdot T(1) + log_2(n-1) \cdot O(n) \\ &= O(n) + O(n \cdot log(n)) \end{aligned}$$

El termino dominante de la expresión anterior es $n \cdot log(n)$ entonces:

$$\Rightarrow \boxed{T(n) = O(n \cdot log(n))}$$

La expresión anterior coincide con la complejidad algorítmica hallada mediante el método de aplicar el teorema maestro, pero se es evidente que el método de expansion en algunos casos se puede volver mucho mas tedioso en comparación a aplicar el teorema.