Complejidad de algoritmos recursivos

Recurrencia

Introducción

Cuando se intenta analizar la complejidad de una función o un procedimiento recursivo es necesario conocer la complejidad de las llamadas recursivas. La definición natural de la función de complejidad de un algoritmo recursivo también es recursiva, y viene dada por una o más ecuaciones de recurrencia.

Definición

Una recurrencia es una ecuación o una desigualdad que describe una función en términos de su propio valor sobre entradas más pequeñas.

Una recurrencia es lineal si cada llamada recursiva genera, como mucho, otra llamada recursiva. En caso contrario, es no lineal.

Los recursos empleados por algoritmos recursivos se pueden describir con la recurrencia T(n), la cual tiene una estructura inductiva:

a) Complejidad del caso base.
b) Complejidad de los casos progresivos.

En general, las ecuaciones de recurrencia tienen la forma:

t(n) = b **para 0 ≤ n ≤ n 0 Casos base**

t(n) = f (t(n), t(n-1), ..., t(n-k), n) **En otro caso**

Resolver este tipo de ecuaciones consiste en encontrar una expresión no recursiva de T(n). ¿Cómo se pueden resolver las ecuaciones en recurrencia? Los métodos son:

- **Método de Sustitución e Iteración.**
- **Métodos de Resolución de Recurrencias (homogéneas y no homogéneas).**
- **Teorema Maestro.**

Dada una función recursiva, ¿cómo se halla la ecuación de recurrencia?

La siguiente función recursiva permite hallar la factorial de un número:

```
int factorial (int n){
  if (n<=1)
     return 1;
 else
     return factorial (n-1)*n;
```

Recordar que T(n) es el tiempo de ejecución en el peor de los casos. La ecuación de recurrencia que acota superiormente a T(n) es:

$$T(n) = \begin{cases} c_1 & si\ n \leq 1 \\ T(n-1) + c_2 & si\ n > 1 \end{cases}$$

Donde c_1 es el tiempo de ejecución del caso base.

En el caso de n mayor a 1, el tiempo de ejecución se puede dividir en dos partes:

T(n-1) la llamada recursiva a la función factorial con n-1

c_2: el tiempo de evaluar la condición n>1 y la multiplicación factorial(n-1)*n

¿Cómo se resuelve? Aplicando el **método de sustitución:**

Se sabe que $T(n)=T(n-1)+c_2$

¿Cómo se calcula T(n-1)?

$T(n-1)=T(n-2)+c_2$

$T(n-2)=T(n-3)+c_2$

$T(n-3)=T(n-4)+c_2$

$T(n)= T(n-1)+c_2$

$=(T(n-2)+c_2)+c_2 = T(n-2)+2*c_2$

$=(T(n-3)+c_2)+2*c_2 = T(n-3)+3*c_2$

$= T(n-k) + k * c2$

Cuando k=n-1 se tiene que $T(n)=T(1)+ (n-1)*c_2$

Entonces T(n) tiene una complejidad lineal, o sea, O(n)

Otro ejemplo:

La siguiente función recursiva:

```
int Recursiva1 (int n) {

 if (n <= 1)
   return(1);
 else
   return(Recursiva1(n-1)+ Recursiva1(n-1));
}
```

La ecuación de recurrencia es:

$$T(n) = \begin{cases} 1, & si\ n \leq 1 \\ 2 \cdot T(n-1)+1, & si\ n > 1 \end{cases}$$

Aplicando el método de sustitución e interacción:

$T(n) = 2*T(n-1) + 1 =$

$= 2*(2*T(n-2) + 1) + 1 = 2^2*T(n-2) + (2^2-1) =$

....

$= 2^k*T(n-k) + (2^k-1)$

Para k = n-1, $T(n) = 2^{n-1}*T(1) + 2^{n-1}-1$, y por tanto T(n) es $O(2^n)$.

Otro ejemplo:

```
int Recursiva3 (int n) {

 if (n <= 1)
   return (1);
 else
   return (2 * Recursiva3(n /2));
}
```

La ecuación de recurrencia:

$$T(n) = \begin{cases} 1, & si\ n \leq 1 \\ T(n/2) + 1, & si\ n > 1 \end{cases}$$

Aplicando el método de sustitución:

$$\begin{aligned} T(n) &= T(n/2) + 1 = \\ &= T(n/2^2) + 2 = \\ &\ldots. \\ &= T(n/2^k) + k \end{aligned}$$

Como la ecuación $n/2^k = 1$ se resuelve para $k = \log_2 n$, tenemos que

$T(n) = T(1) + \log_2 n = 1 + \log_2 n$, y por tanto $T(n)$ es $O(\log n)$.