

Tema 10. Resumen de las ecuaciones.

La regla del límite: Nos permite comparar dos funciones en cuanto a la notación asintótica se refiere. Tendremos que calcular el siguiente límite:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}.$$

Al resolver el límite se nos darán 3 posibles resultados:

$$1. \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c \in \mathbb{R} \Rightarrow$$

$$\Rightarrow \left\{ \begin{array}{lll} f(n) \in O(g(n)) & f(n) \in \Omega(g(n)) & f(n) \in \theta(g(n)) \\ g(n) \in O(f(n)) & g(n) \in \Omega(f(n)) & g(n) \in \theta(f(n)) \end{array} \right\}.$$

Estas funciones se comportan igual, diferenciándose en una constante multiplicativa.

$$2. \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty \Rightarrow$$

$$\Rightarrow \left\{ \begin{array}{lll} f(n) \notin O(g(n)) & f(n) \in \Omega(g(n)) & f(n) \notin \theta(g(n)) \\ g(n) \in O(f(n)) & g(n) \notin \Omega(f(n)) & g(n) \notin \theta(f(n)) \end{array} \right\}.$$

Por muy alta que sea la constante multiplicativa de $g(n)$ nunca superará a $f(n)$.

$$3. \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \Rightarrow$$

$$\Rightarrow \left\{ \begin{array}{lll} f(n) \in O(g(n)) & f(n) \notin \Omega(g(n)) & f(n) \notin \theta(g(n)) \\ g(n) \notin O(f(n)) & g(n) \in \Omega(f(n)) & g(n) \notin \theta(f(n)) \end{array} \right\}.$$

$g(n)$ crece más exponencialmente que $f(n)$, por lo que sería su cota superior.

Tendremos dos tipos:

- Reducción por sustracción:

La ecuación de la recurrencia es la siguiente:

$$T(n) = \begin{cases} c * n^k & \text{si } 0 \leq n < b \\ a * T(n - b) + c * n^k & \text{si } n \geq b \end{cases}$$

La resolución de la ecuación de recurrencia es:

$$T(n) = \begin{cases} \theta(n^k) & \text{si } a < 1 \\ \theta(n^{k+1}) & \text{si } a = 1 \\ \theta(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$$

- Reducción por división:

La ecuación de la recurrencia es la siguiente:

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

La resolución de la ecuación de recurrencia es:

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

siendo:

a: Número de llamadas recursivas.

b: Reducción del problema en cada llamada.

$c * n^k$: Todas aquellas operaciones que hacen falta además de las de recursividad.

El esquema voraz es el siguiente:

```
funcion voraz (C: Conjunto): conjunto
{ C es el conjunto de candidatos }
   $S \leftarrow \emptyset$       { Construimos la solución en el conjunto S }
  mientras  $C \neq \emptyset$  y  $\neg$ solución (S) hacer
     $x \leftarrow \text{seleccionar}(C)$ 
     $C \leftarrow C \setminus \{x\}$ ;
    si factible ( $S \cup \{x\}$ ) entonces  $S \leftarrow S \cup \{x\}$ 
    si solución (S) entonces devolver S
  si no devolver “no hay solución”
```

Los esquemas de vuelta atrás:

```
fun vuelta-atrás (ensayo)
si valido (ensayo) entonces
  devolver ensayo
si no
  para cada hijo en compleciones (ensayo) hacer
    si condiciones-de-poda (hijo) entonces
      vuelta-atrás (hijo)
    fsi
  fpara
fsi
ffun
```

```
fun vueltaatrás ( $v[1..k]$ )
{ v es un vector k-prometedor }
si v es una solución entonces escribir v
si no
  para cada vector (k+1)-prometedor w
    tal que  $w[1..k] = v[1..k]$  hacer
      vueltaatrás ( $w[1..k + 1]$ )
```

El de ramificación y poda (problemas de maximización):

```
fun ramificación-y-poda (ensayo)
  m ← montículo-vacío
  cota-superior ← inicializar-cota-superior
  solución ← solución-vacía
  añadir-nodo (ensayo, m)
  mientras no vacío (m) hacer
    nodo ← extraer-raíz (m)
    si valido (nodo) entonces
      si coste (nodo) < cota-superior entonces
        solución ← nodo
        cota-superior ← coste (nodo)
      fsi
    si no { Nodo no es válido (solución) }
      si cota-inferior (nodo) ≥ cota-superior entonces
        devolver solución
      si no { cota-inferior (nodo) < cota-superior }
        para cada hijo en compleciones (nodo) hacer
          si condiciones-de-poda (hijo) y
            cota-inferior (hijo) < cota-superior entonces
            añadir-nodo (hijo, m)
          fsi
        fsi
      fpara
    fsi
  fmientras
ffun
```

El esquema de divide y vencerás:

```
fun divide-y-vencerás (problema)
  si suficientemente-simple (problema) entonces
    dev solucion-simple (problema)
  si no { No es solución suficientemente simple }
    { $p_1 \dots p_k$ } ← decomposicion (problema)
    para cada  $p_i$  hacer
       $s_i$  ← divide-y-vencerás ( $p_i$ )
    fpara
    dev combinacion ( $s_i \dots s_k$ )
  fsi
ffun
```