Ecuaciones de recurrencia

Aparecen naturalmente al analizar algoritmos recursivos.

Por ejemplo:

```
Fib(n : natural)
  if n ≤ 1 then
      return n
  else
      return Fib(n-1) + Fib(n-2)
```

El valor entregado por $Fib(n)$ puede expresarse mediante la ecuación de recurrencia:

$$\underbrace{T(n)}_{\text{valor entregado por } Fib(n)} = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ \underbrace{T(n-1)}_{\text{valor entregado por } Fib(n-1)} + \underbrace{T(n-2)}_{\text{valor entregado por } Fib(n-2)}, & n > 1. \end{cases}$$

Queremos saber exactamente qué función es $T(n)$.

Nos gustaría encontrar la "forma cerrada" de T, (2)
para poder estudiarla de mejor manera
(por ejemplo, para poder entender su velocidad de crecimiento).

Para este caso, ¿Cuál será la función que acota $T(n)$?

Vamos a demostrar que $T(n) \in O(2^n)$

Eso significa que hay que demostrar que $\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}$, tal que
$$T(n) \leq c \cdot 2^n, \quad \forall n \geq n_0.$$

Vamos a demostrarlo por inducción:

Caso base) Primero probamos que la propiedad se cumple para los casos base.

$n=0 \Rightarrow T(0) = 0 \Rightarrow$ se debe cumplir $T(0)=0 \leq c \cdot 2^0$, lo cual es cierto para $c \geq 0$.

$n=1 \Rightarrow T(1) = 1, \Rightarrow$ se debe cumplir $T(1)=1 \leq c \cdot 2^1$, lo cual es cierto para $c \geq 1/2$.

Esto significa que la propiedad es cierta para los casos base. Planteamos ahora la hipótesis inductiva.

H.I.) Asumimos que $T(n') \leq c \cdot 2^{n'}$, $\underbrace{\forall n' \in \{0, 1, \ldots, n-1\}}_{\text{inducción fuerte, por eso era necesario chequear todos los casos base.}}$

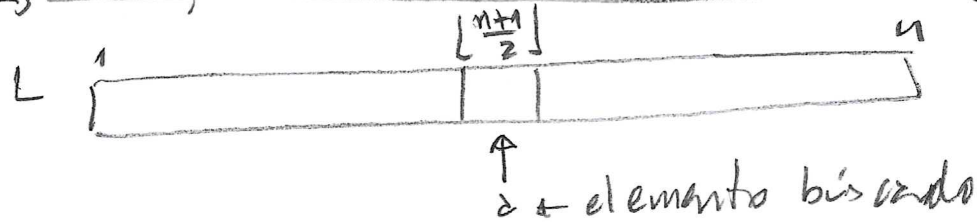
Paso inductivo) Sea $n \geq 2$. Entonces $T(n)$ corresponde al caso recurrente, $T(n) = T(n-1) + T(n-2)$. Por H.I., $T(n-1) \leq c \cdot 2^{n-1}$ y $T(n-2) \leq c \cdot 2^{n-2}$. Entonces

$$T(n) \leq c \cdot 2^{n-1} + c \cdot 2^{n-2} < c \cdot 2^{n-1} + c \cdot 2^{n-1} = c \cdot 2^{n},$$

con lo que demostramos la propiedad.

Estudiamos a continuación un algoritmo muy importante.

↳ <u>Algoritmo de búsqueda binaria en un arreglo ordenado</u>



Comparando el elemento buscado a con $L[\lfloor \frac{n+1}{2} \rfloor]$ y usando la transitividad del orden, podemos descartar la mitad del arreglo con una única comparación.

El algoritmo es:

```
BBinaria (a, L[i..j])
    if i > j then return no
    else if i = j then
        if L[i] = a then return i
        else return no
    else
        p ← ⌊(i+j)/2⌋
        if L[p] < a then return BBinaria(a, L[p+1..j])
        else if L[p] > a then return BBinaria(a, L[i..p-1])
        else return p.
```

Denotemos con $T(n)$ el tiempo de ejecución de este algoritmo para la invocación inicial

$$BBinaria(a, L[1..n])$$

Vamos a considerar sólo comparaciones entre elementos de la lista L y el valor buscado a.

Así, el tiempo de ejecución puede expresarse con la ecuación de recurrencia: (tiempo para comparar por "=")

$$T(n) = \begin{cases} b, & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + d, & n > 1 \end{cases}$$

(tiempo de las comparaciones < y >)

con $b \in \mathbb{N}$, $d \in \mathbb{N}$, son constantes, $b \geq 1$, $d \geq 1$.

Antes de usar inducción, vamos a suponer que n es una potencia de 2, para tratar de "adivinar" una cota para $T(n)$, la que luego probamos por inducción (para todo n)

O sea $n = 2^k$, para $k \geq 0$. Entonces, $\boxed{k = \lg_2 n}$

Reemplazamos esto en la ecuación y nos queda:

$$T(2^k) = \begin{cases} b, & k = 0 \ [2^k = 1] \\ T(2^{k-1}) + d, & k > 0 \ [2^k > 1] \end{cases}$$

Extendiendo la expresión anterior tenemos:

$$T(2^K) = \underbrace{T(2^{K-1})} + d$$

$$= (T(2^{K-2}) + d) + d = T(2^{K-2}) + 2d$$

$$= T(2^{K-3}) + d + 2d = T(2^{K-3}) + 3d$$

$$\vdots$$

$$= T(\underbrace{2^{K-i}}) + id$$

Esto será un caso base cuando $2^{K-i} = 1$, es decir, $\boxed{K - i = 0}$, con lo cual $\boxed{i = K.}$

$$\Rightarrow T(2^K) = T(2^{\overset{K-K}{0}}) + Kd$$

$$= b + K \cdot d.$$

dado que $K = \log_2 n$, $\boxed{T(n) = d \cdot \log_2 n + b}$

para n potencia de 2.

A continuación, demostraremos que para todo n, se cumple $T(n) \in O(\log_2 n)$.

Esto es, hay que probar que $\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}$ tal que

$$T(n) \leq c \cdot \log_2 n, \quad \forall n \geq n_0$$

Lo demostramos por inducción constructiva: (6)

Casos base) para $n=1$, $T(1)=b$ y $\lg 1=0$, entonces debería cumplirse que $b \leq c \cdot 0$, con lo cual no es posible ya que $b \geq 1$. Descartamos este caso base. Probamos ahora $n=2$.

Para $n=2$, $T(2)=b+d$ y $\lg_2 2=1$, por lo que $b+d \leq c \cdot 1$, y por lo que cualquier $c \geq b+d$ nos sirve.

Dado que usaremos inducción fuerte, tenemos que verificar si no hay más casos base.

Para $n=3$, $T(3)= \underbrace{T(1)} + d = b+d$

Dado que para $T(1)$ no se cumple la propiedad, agregamos $T(3)$ como caso base y chequeamos:

$$b+d \leq c \cdot \lg_2 3$$

Notar que $T(4)=T(2)+d$, y $T(2)$ es caso base para el que se cumple la propiedad $\Rightarrow$ Los casos base son $n=2$ y $n=3$.

Hipótesis Inductiva) Asumimos que se cumple

$$T(n') \leq c \cdot \lg_2 n', \quad \forall n' \in \{2, \ldots, n-1\}$$

Paso Inductivo) Para $n \geq 4$ tenemos:

$$
\begin{aligned}
T(n) &= T\left(\left\lfloor \tfrac{n}{2} \right\rfloor\right) + d \\
&\leq c \lg \left\lfloor \tfrac{n}{2} \right\rfloor + d \leq c \lg\left(\tfrac{n}{2}\right) + d \\
&= c\lg n - c\lg 2 + d \leq c \lg n - (b+d) + d \\
&= c\lg n - b < c \lg n \quad \#
\end{aligned}
$$

dado que $c \geq b+d$, estamos restando algo más chico que c.

A continuación estudiaremos un caso más complicado (7)

Sea:

$$T(n) = \begin{cases} 0, & n = 0 \\ n^2 + nT(n-1), & n > 0 \end{cases}$$

Notar que $T(n) = n^2 + \underbrace{nT(n-1)}$

tiene la forma del factorial $n!$ (aunque no es exactamente eso por el término n^2).

Demostraremos que $T(n) \in O(n!)$.

Debemos encontrar $c \in \mathbb{R}^+$, $n_0 \in \mathbb{N}$, tal que

$$T(n) \leq c \cdot n!, \quad \forall n \geq n_0$$

Lo haremos por inducción.

Caso base) $n = 0 \Rightarrow T(0) = 0$, por lo tanto $0 \leq c \cdot 0! = c$, por lo que la propiedad se cumple para $n = 0$, para cualquier $c > 0$.

Hipótesis inductiva) $T(n) \leq c \cdot n!$

Paso inductivo) (para $n \geq 1$)

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

$$\overset{H.I.}{\leq} (n+1)^2 + (n+1)(c \cdot n!)$$

$$= (n+1)^2 + c \cdot (n+1)!$$

Sin embargo, el término $(n+1)^2$ no es constante y $n \geq 0$, por lo que NO podremos acotar $(n+1)^2 + c(n+1)! \leq c \cdot (n+1)!$ [para ningún c]

Para poder demostrar que $T(n) \in O(n!)$ vamos a demostrar algo equivalente (en este caso):

$$\boxed{\exists c \in \mathbb{R}^+, \exists d \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \text{ tal que } T(n) \leq c \cdot n! - d \cdot n, \quad \forall n \geq n_0}$$

Caso base) $n=0$, $T(0)=0$, $\Rightarrow$ $0 \leq c \cdot 0! - d \cdot 0 = c$. Se cumple para cualquier $c, d > 0$.

Hipótesis inductiva) Para n se cumple $T(n) \leq c \cdot n! - dn$.

Paso inductivo) Para $n+1$ tenemos

$$
\begin{aligned}
T(n+1) &= (n+1)^2 + (n+1)T(n) \\
\overset{HI}{\longrightarrow} &\leq (n+1)^2 + (n+1)(c \cdot n! - d \cdot n) \\
&= c \cdot (n+1)! + (n+1)^2 - d \cdot n \cdot (n+1) \\
&= c(n+1)! + (n+1)\left((n+1) - dn\right)
\end{aligned}
$$

Para poder demostrar que

$$\cancel{c(n+1)! + (n+1)}\left((n+1) - dn\right) \leq \cancel{c(n+1)! - (n+1)}d$$

hay que demostrar que $(n+1) - dn \leq -d$,
lo que es equivalente a

$$(n+1) \leq d(n-1)$$

$$\Leftrightarrow \quad \frac{(n+1)}{(n-1)} \leq d.$$

si $n \geq 2$ (porque tenemos $(n-1)$ dividiendo)

$\Rightarrow d \geq 3$.

Consideramos entonces $n_0 = 2$ y $d = 3$.

Nos falta asegurar ahora que la propiedad se cumple para el caso base $n_0 = 2$.

Tenemos:

$$T(2) = 2^2 + 2\underbrace{T(1)}_{\underbrace{1^2 + 1\underbrace{T(0)}_{0}}_{1}} = 6$$

$$\Rightarrow 6 \leq c \cdot 2! - d \cdot 2$$

$$= c \cdot 2 - 3 \cdot 2 = 2c - 6$$

$$\Rightarrow 6 \leq 2 \cdot c - 6, \Rightarrow \boxed{c \geq 6.}$$

$$\Rightarrow \forall n \geq 2,\ T(n) \leq 6\,n! - \underbrace{3 \cdot n}$$

está restando, no afecta.

$$\Rightarrow T(n) \in O(n!)$$