4. Ecuaciones de recurrencia

4.1 Introducción

En el análisis de las ecuaciones de recurrencia tenemos desarrollos de expresiones matemáticas de recurrencia mediante métodos de resolución.
Cuando divido el problema y cada problema viene con la mitad de los datos, cada nivel tendra un árbol con altura (en esa ecuación es dividir y vencer)

Si en un resta y vencerás pasa una sola vez (10, 9, 8, ... 1), pero si es una ecuación del tipo resta y serás vencido entonces serán los n hijos y tardará demasiado.

Un mismo problema como el factorial tiene su versión iterativa o recursiva:

Una **condición** es cuando para momento específico hace algo, no como un **caso** cuando en función a un dato realiza una acción.
En un algoritmo es óptimo hacer únicamente un punto de retorno (canalizar).

$$\prod_{i=1}^{n} i$$

$$fact(n) = n \times fact(n-1)$$

4.1.1 Ecuaciones

Consideramos constantes $a \geq 1$; $b > 1$; $c, d > 0$.

Divide y Vencerás (DyV)

En DyV aplican funciones como
$F_0 : T(n) = T(n/b) + f(n)$
$F_1 : T(n) = aT(n/b) + f(n)$
$F_2 : T(n) = T(n/b) + T(n/c) + f(n)$
$F_3 : T(n) = T(n/b) + T(n/c) + \cdots + T(n/z) + f(n)$

Resta y Vencerás (RyV)

Ecuación común en RyV:
$F_4 : T(n) = T(n-b) + f(n)$

Resta y serás Vencido (RysV)

Ecuaciones comunes en RysV:
$F_5 : T(n) = aT(n-b) + f(n)$
$F_6 : T(n) = aT(n-b) + cT(n-d) + f(n)$

4.1.2 Clasificación

A continuación, se muestra cómo se clasifican las ecuaciones anteriores según el método de resolución más y no adecuado en orden de conveniencia:

Iteración

$I \in \{F_4, F_5, F_0, F_1\} \notin \{F_2, F_3, F_6\}$

Árbol de Recursión

$A \in \{F_2, F_3, F_6, F_5, F_1, F_0\} \notin \{F_4\}$

Teorema Maestro

$M \in \{F_1, F_0\} \notin \{F_2, F_3, F_4, F_5, F_6\}$

Sustitución Inteligente

$S \in \{F_5, F_6, F_4, F_2, F_3, F_1, F_0\}$

Ecuación Característica

$E \in \{F_5, F_6, F_4\} \notin \{F_0, F_1, F_2, F_3\}$
No obstante F_0 puede pertenecer si se representa como $t_k = 2t_{k-1} - \lg k$.
(lineal no homogenea, donde)

4.2 Métodos

Los métodos mencionados antes serán explicados más a detalle.

4.2.1 Método de la iteración

Aplicable cuando existe a lo más un punto de iteración (si la complejidad es apreciable en múltiples puntos).

■ **Example 4.1** Supongamos dan:

$$T(n) = T(n/2) + 1; \quad T(1) = 1$$

Hay 02 formas de analizar la función recursiva.

Definition 4.2.1 — Evaluación directa.

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

$$T(n/2) = T(n/4) + 1$$

$$\dots$$

$$T(1) = 1$$

Sea x el número de iteraciones para llegar a $n = 1$ regida por $n = 2^x$. Con esto $x = \lg n + 1$, el 1 viene tras la primera iteración cuál no es una partición. Complejidad de $T(n) = \Theta(\lg n), O(\lg n)$

Definition 4.2.2 — Evaluación indirecta. Función reemplazada en tiempo real.

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

$$T(n) = T(n/4) + 1 + 1$$

$$T(n) = T(n/8) + 1 + 1 + 1$$

$$T(n) = T(n/16) + 1 + 1 + 1 + 1$$

$$\cdots$$

$$T(1) = 1 + \cdots + 1$$

Cantidad de unos en $T(n)$ es $\lg n$, podemos describirlo como:

$$\sum_{1}^{\lg n+1} 1$$

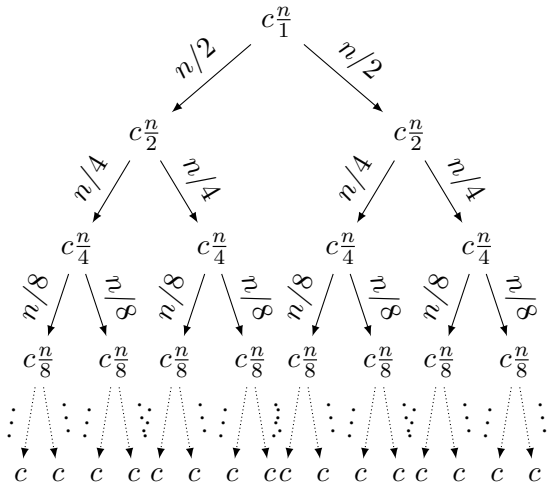
Resultante en $\lg n + 1$, el 1 proveniente de la primera partición.

■

4.2.2 Método del árbol de recursión

■ **Example 4.2** Son análogos los conceptos de (Nodos:Ambiente) y (Arista:Entrada). Así mismo la relación entre Altura (h) y Nivel (l): $l = h + 1$.

Aplicado sobre Merge Sort : $msort = 2T(n/2) + cn$; $\quad T(1) = c$, donde 2 son los hijos, $T(n/2)$ el tamaño de la entrada y cn el dato del nodo.



1. Llamado sin coste asociado.
2. $\rightarrow$ cost : $2^0 c\frac{n}{2^0}$
3. $\rightarrow$ cost : $2^1 c\frac{n}{2^1}$
4. $\rightarrow$ cost : $2^2 c\frac{n}{2^2}$
5. $\rightarrow$ cost : $2^3 c\frac{n}{2^3}$
6. $\vdots$
7. $\rightarrow$ cost : $\log n$

El coste $h = cn + cn + \cdots + cn = \lg n$, por $l = \lg n + 1$. Hallando su complejidad tenemos:

$$cost : \sum_{0}^{\lg n+1} cn = cn(\lg n + 1)$$

$$T(n) = cn \lg n + cn = \Theta(n \lg n)$$

■

4.2.3 Método del teorema maestro

Aplicado sobre funciones DyV *(ex: $aT(n/b) + f(n)$)*, donde los problemas generan subproblemas independientes.

Theorem 4.2.1 — Casos. Son mutuamente excluyentes.

1. $f(n) \in O(n^{\lg_b a - \epsilon}), \epsilon \in \mathbb{R}^+ - \{0\} \implies T(n) = \Theta(n^{\lg_b a})$
2. $f(n) \in \Theta(n^{\lg_b a}) \implies T(n) = \Theta(n^{\lg_b a} \lg n)$
3. $f(n) \in \Omega(n^{\lg_b a + \epsilon}), \epsilon \in \mathbb{R}^+ - \{0\}$ *si* $af(n/b) \leq cf(n) \implies T(n) = \Theta(f(n))$, es aquí particular la **condición de regularidad** *(La función tiene que ser más grande que la función en sus partes o en lo sumo igual).*

Las constantes definidas son $b > a \geq 1$ y en $1 > c > 0$.

■ **Example 4.3** Tiene muchas versiones (veremos el maestro principal), algunos autores dividen las clases (e.g. teorema maestro)
Aplicado sobre *msort*: Definimos $a \geq 2, b > 1$.

Caso 01: Tenemos $cn \in O(n^{\lg_2 2 - 0,1}) \implies cn \stackrel{?}{=} O(n^{0,9}) \implies cn \neq cn^{0,9}$. Por ende, queda descartado.

Caso 02: Tenemos $cn \in \Theta(n^{\lg_2 2}) \implies cn \stackrel{?}{=} \Theta(n \lg n)$. Se cumple, hallamos la complejidad directamente. ■

■ **Example 4.4** Trabajaremos con la ecuación de QuickSort en su mejor escenario:

$$T(n) = 2T(n) + cn$$

Entonces verificamos que cumpla la forma y a, b funcionan. Pero el hecho que cumpla con la forma no implica siempre de una solución. // Determinamos $f(n) = cn$ y aplicamos por caso:

1. $Cn \stackrel{?}{=} O(n^{\lg_2 2 - \epsilon})$ En estas polinomiales el exponente es fundamental Si tenemos que $\epsilon = 0{,}0 \cdots 01$. Como notamos esta función no satisface puesto
2. Notamos que acá si se cumple, podemos sacar el límite y esto debe darnos una constante (no 1).

Vemos que nos dió el orden, no la función de eficiencia. Lo que nos interesa siempre, eventualmente es el orden.

■

4.2.4 Método de la sustitución inteligente

Consiste de los pasos:

1. Suponer solución.

2. Demostrar con PIM que es funcional (hallar ctes).

■ **Example 4.5 — Similar a msort.** Se tiene la función $T(n) = 2T\lfloor n/2 \rfloor + n$
Notamos como $msort = 2T(n/2) + cn \wedge T(n) \in O(n \lg n)$
Suponemos $k = \frac{n}{2}$, por lo que $k < n$.
Sustituímos en la ecuación de recurrencia:

$$T(n) \leq 2(c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor) + n$$
$$\leq cn \lg \lfloor n/2 \rfloor + n$$
$$\leq cn \lg \lfloor n \rfloor - cn + n$$
$$\leq cn \lg n; \quad c \geq 1$$

Debemos mostrar solución cumple: Condición límite $\wedge$ Casos base.
Notamos como no cumple $n = 1$ puesto

$$T(1) \leq c\ 1 \log 1$$

No obstante cumple para constantes superiores.

$$T(2) \leq c\ 2 \lg 2$$
$$T(2) \leq c\ 3 \lg 3$$

Tomando ventaja de la Notación asintótica, reemplazamos casos base (*c.b.*) en la prueba inductiva.

$$T(n) = 2T\lfloor n/2 \rfloor + n$$
$$\therefore T(2) = 2(1) + 2 = 4$$
$$\therefore T(3) = 2(1) + 3 = 5$$

La prueba inductiva dicta $T(n) \leq cn \lg n$ y un $c \geq 2$, tenemos:

$$T(2) \leq c\ 2 \lg 2$$
$$T(3) \leq c\ 3 \lg 3$$

Es importante considerar que la **prueba inductiva** $\neq$ **ec. recurrencia**. ■

4.2.5 Método de ecuación característica

Es aplicado sobre ecuaciones de recurrencia lineales homogeneas (ER-LH) de la forma

$$a_0 T(n) = a_1 T(n-1) \pm \cdots \pm a_k T(n-k)$$
$$x^n = T(n); \quad T_0 = I_0; \quad T_1 = I_1; \quad \cdots; \quad T_{k-1} = I_{k-1}.$$

Definition 4.2.3 — Ecuación característica.

$$a_n T_n + a_{n-1} + T_{n-1} + \cdots + a_{n-k} T_{n-k} = 0$$
$$= a_n T_k + a_{n-1} T_{k-1} + \cdots + a_k T_0 = 0$$
$$\implies a_0 x^k + a_1 x^{k-1} + \cdots + a_k x^0 = 1$$

Con raíces $r : r_1, r_2, \cdots, r_k$. Sean distintas y de multiplicidad 1 ó múltiples de multiplicidad.

Theorem 4.2.2 — Mult. = 1.

$$T(n) = \sum_{i=1}^{n} c_i r_i^n$$

■ **Example 4.6 — Sobre Fibonacci.** Defínase $T(n) = T(n-1) + T(n-2);\ T(0) = 0;\ T(1) = 1.\ n \in \mathbb{Z}^+$.
Primero obtenemos la ecuación característica:

$$1T(n) - 1T(n-1) - 1T(n-2) = 0; \quad T(n-k) = x^{n-k}$$

$$1x^{2-0} - 1x^{2-1} - 1x^{2-2} = 0$$

$$x^2 - x - 1 = 0$$

Mediante la cuadrática hallamos las raíces, son $r_{1,2} = \frac{1}{2}(1 \pm \sqrt{5})$. Notamos que son distintas, aplicamos el teorema para mult. $= 1$.

$$T(n) = c_1[\frac{1}{2}(1+\sqrt{5})]^n + c_2[\frac{1}{2}(1-\sqrt{5})]^n$$

Aplicado sobre los casos base T(0) y T(1):

$$T(0) = c_1[\frac{1}{2}(1+\sqrt{5})]^0 + c_2[\frac{1}{2}(1-\sqrt{5})]^0$$

$$T(1) = c_1[\frac{1}{2}(1+\sqrt{5})]^1 + c_2[\frac{1}{2}(1-\sqrt{5})]^1$$

Resta resolver el sistema 2×2 y entonces obtendremos la complejidad del algoritmo. ■

5. Algoritmos voraces

5.1 Introducción

LA entrada son los candidatoss, se trabaja por etapas (y se hace el mismo proceso), escoge candidatos y si sire lo lleva al conjunto solución, es como seguir la ramita de un árbol.

Metodología simple, aplicada especialmente sobre optimización *(máximos y mínimos)*.
Se obtiene un subconjunto *sln* que satisfaga restricciones asociadas al problema de n entradas. Si *sln* satisfacce las restricciones decimos es 'prometedora'; Una *sln* prometedora que optimiza la función objetivo (FO) z es una *sln* 'óptima'.
Son una forma ágil de obtener una solución algorítmica. Son una forma ágil de obtener una solución algorítmica.
Consiste en identificar en la entrada/s de datos quién puede servir para resolver el problema.

Definition 5.1.1 — Elementos. Contiene las etapas:

1. Conjunto de candidatos → n entradas.
2. Función de selección → Candidatos idóneos.
3. Función de validación *(factibilidad)* → Subconjunto prometedor *(Permite adición de candidatos)*.
4. Una z a evaluar y optimizar.
5. Una función comprobante → subconjunto es solución (óptima o no).

Digamos que se compra un terreno porque se ve muy vacano pero así si esté muy bien ubicado, el mejor costo, el mejor clima pero sobre arenas movedizas, la factibilidad es nula (aplicada sobre candidatos).
Suponiendo que es factible el candidato miramos la función objetivo para evaluar.
La manera como vemos el problema es determinante a su solución.

Trabaja sin pensar en el futuro, por etapas escoge el óptimo local suponiendo será global al problema.
Previo adicionar candidatos, comprueba sea prometedor añadirlo, si no, se descarta para