

Algoritmos y Complejidad

Técnicas y Herramientas

Pablo R. Fillottrani

Depto. Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Primer Cuatrimestre 2019



Técnicas y Herramientas

Técnicas de Demostración

Herramientas Matemáticas Básicas

Notación Asintótica

Estructuras de Datos

Análisis de Algoritmos por Estructuras de Control

Resolución de Recurrencias



- ▶ pruebas por contradicción
- ▶ pruebas por inducción
 - ▶ principio de inducción
 - ▶ inducción matemática generalizada
 - ▶ inducción constructiva

Herramientas ya conocidas. Leer apunte disponible en la página web.



- ▶ **Lógica:** cálculo proposicional y de predicados.
- ▶ **Teoría de Conjuntos:** operaciones básicas, producto cartesiano, cardinalidad.
- ▶ **Teoría de Números:** módulo, intervalos, techo y piso.
- ▶ **Elementos básicos de álgebra y análisis:** funciones, relaciones, series, sumatorias y productos, límites, módulos, logaritmos.
- ▶ **Probabilidades:** probabilidad condicional, esperanza, varianza.
- ▶ **Combinatoria:** permutaciones, combinaciones.

En el final del apunte se presenta un compendio de fórmulas útiles sobre estos temas.



Objetivos

- ▶ no interesa conocer los valores absolutos de las funciones
- ▶ permitir una caracterización simple de la eficiencia de un algoritmo y comparar las performances relativas de distintos algoritmos
- ▶ independizar el análisis de los algoritmos de condiciones específicas de implementación: lenguaje de programación, compilador, equipo, etc.

- ▶ se aplica a funciones de tiempo de ejecución o de espacio de memoria de algoritmos en base a la longitud de la entrada:
 $f(n) : \mathbf{N} \longrightarrow \mathbf{R}^+$
- ▶ se denomina **asintótica** porque analiza el comportamiento de las funciones en el *límite*, es decir su **tasa de crecimiento**

Notación $O(\cdot)$

$$O(g(n)) = \{f(n) : \exists c \in \mathbf{R}^+, \exists n_0 \in \mathbf{N}, \text{ tal que} \\ f(n) \leq cg(n) \text{ para todo } n \geq n_0\}$$

- ▶ determina una cota superior en la tasa de crecimiento de una función, dentro de un factor constante
- ▶ ejemplos:
 - ▶ $6n^3 \in O(n^3)$ ya que se cumple la definición con $c = 6, n_0 = 1$
 - ▶ $3 \log n \in O(n)$ ya que se cumple la definición con $c = 1, n_0 = 4$



notacion-O

Ejemplos:

- ▶ $300n^2 \in O(n^2)$
- ▶ $5n^4 - 4n^3 + 10n^2 + 39 \in O(n^4)$
- ▶ $\log_b n \in O(\log_a n), \forall a, b$
- ▶ $2^n \in O(n!)$
- ▶ $500000n \in O(0,00001n^2)$
- ▶ $0,000001n^2 \notin O(500000n)$
- ▶ $n! \notin O(2^n)$

Notación $\Omega(\cdot)$

$$\Omega(g(n)) = \{f(n) : \exists c \in \mathbf{R}^+, \exists n_0 \in \mathbf{N}, \text{ tal que} \\ f(n) \geq cg(n) \text{ para todo } n \geq n_0\}$$

- ▶ determina una cota inferior en la tasa de crecimiento de una función, dentro de un factor constante
- ▶ ejemplos:
 - ▶ $6n^3 \in \Omega(n^3)$ ya que se cumple la definición con $c = 1, n_0 = 1$
 - ▶ $1/3n \in \Omega(\log n)$ ya que se cumple la definición con $c = 1/3, n_0 = 1$



notacion-Omega

Ejemplos:

- ▶ $3n^5 + 4n^3 - 8n^2 + 10n \in \Omega(n^4)$
- ▶ $\log_b n \in \Omega(\log_a n), \forall a, b$
- ▶ $n! \in \Omega(2^n)$
- ▶ $0,00001n^2 \in \Omega(50000n)$
- ▶ $50000n \notin \Omega(0,00001n^2)$
- ▶ $2^n \notin \Omega(n!)$

Notación $\Theta(\cdot)$

$$\Theta(g(n)) = \{f(n) : \exists c, d \in \mathbf{R}^+, \exists n_0 \in \mathbf{N}, \text{ tal que} \\ cg(n) \leq f(n) \leq dg(n) \text{ para todo } n \geq n_0\}$$

- ▶ determina una cota superior e inferior en la tasa de crecimiento de una función, dentro de un factor constante
- ▶ ejemplos:
 - ▶ $6n^3 \in \Theta(n^3)$ ya que se cumple la definición con $c = 6, d = 6, n_0 = 1$.
 - ▶ $1/3n \in \Theta(n)$ ya que se cumple la definición con $c = 1/5, d = 1, n_0 = 1$.



notacion-Theta

Ejemplos:

- ▶ $3n^2 \in \Theta(n^2)$
- ▶ $\log n \notin \Theta(n)$
- ▶ $2^{n+1} \in \Theta(2^n)$
- ▶ $500000n^2 \in \Theta(0,00001n^2)$
- ▶ $\log_b n \in \Theta(\log_a n)$ para todo $a, b > 0$

Uso en Ecuaciones

- ▶ por ejemplo, $f(n) = 2n^2 + \Theta(n)$ significa que $f(n)$ es igual a $2n^2$ más alguna función cualquiera perteneciente a $\Theta(n)$
- ▶ $2n^2 + O(n) = O(n^2)$ significa que *no importando que función perteneciente a $O(n)$ se sume a $2n^2$, siempre el resultado es una función en $O(n^2)$*
- ▶ $f(n) = O(g(n)) + O(h(n))$ significa que $f(n)$ es una función que se puede obtener *sumando punto a punto una función de $O(g(n))$ con una función de $O(h(n))$*
- ▶ se evita hacer referencia a detalles que no afectan el comportamiento general de la función

Algunas Propiedades útiles

- ▶ $O(f_1(n) + f_2(n)) = O(\max(f_1(n), f_2(n)))$
- ▶ $f(n) \in \Theta(g(n))$ si y solo si $g(n) \in \Theta(f(n))$
- ▶ $f(n) \in O(g(n))$ si y solo si $g(n) \in \Omega(f(n))$
- ▶ $f(n) \in \Theta(g(n))$ si y solo si $f(n) \in O(g(n)) \wedge f(n) \in \Omega(g(n))$
- ▶ si $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \in \mathbf{R}^+$ entonces $f(n) \in O(g(n))$ y $g(n) \in O(f(n))$
- ▶ si $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$ entonces $f(n) \in O(g(n))$ pero $g(n) \notin O(f(n))$

ORDENAR UN ARREGLO

Algoritmo: Ordenamiento por Inserción

- Costo de la ejecución del algoritmo:

$$\begin{aligned}T_I(n) &= c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=1}^{n-1} t_j + c_5 \sum_{j=1}^{n-1} (t_j - 1) + \\&\quad + c_6 \sum_{j=1}^{n-1} (t_j - 1) + c_8(n-1) \\&= (c_1 + c_2 + c_3 + c_8)n - (c_2 + c_3 + c_8) + (c_4 + c_5 + c_6) \sum_{j=1}^{n-1} t_j - (c_5 + c_6) \sum_{j=1}^{n-1} 1\end{aligned}$$

► para analizar el $O(\cdot)$ se tiene:

$$\begin{aligned}T_I(n) &= d_1 n - d_2 + d_3 \sum_{j=1}^{n-1} t_j - d_4 \sum_{j=1}^{n-1} 1 \\&\leq d_1 n + d_3 \sum_{j=1}^{n-1} t_j \\&\leq d_1 n + d_3 \sum_{j=1}^{n-1} n \\&= d_1 n + d_3 n(n-1) \\&\leq d_1 n + d_3 n^2\end{aligned}$$

► luego $T(n) \in O(n^2)$.



- para analizar el $\Omega(\cdot)$ se tiene:

$$\begin{aligned}T_I(n) &= d_1 n - d_2 + d_3 \sum_{j=1}^{n-1} t_j - d_4 \sum_{j=1}^{n-1} 1 \\&\geq d_1 n - d_2 + d_3 \sum_{j=1}^{n-1} j - d_4(n-1) \\&= d_1 n - d_2 + d_3(n-1)n/2 - d_4(n-1) \\&\geq \frac{d_3}{2}n^2 + d_1 n - \left(\frac{d_3}{2} + d_4\right)n - (d_2 + d_4) \in \Omega(n^2)\end{aligned}$$

- recordemos que $T(n)$ es el tiempo de ejecución en el peor caso para instancias de tamaño n
- luego $T(n) \in \Omega(n^2)$ y por lo tanto también $T(n) \in \Theta(n^2)$



NÚMERO DE FIBONACCI

Algoritmo: algoritmo iterativo simple

- ▶ para analizar el $O(\cdot)$ se tiene:

$$T_{FIB2}(n) = b + \sum_{k=1}^n (c_1 + c_2 + c_3) + d \leq fn$$

- ▶ luego $T_{FIB2}(n) \in O(n)$
- ▶ para analizar el $\Omega(\cdot)$ se tiene:

$$T_{FIB2}(n) = b + \sum_{k=1}^n (c_1 + c_2 + c_3) + d \geq \sum_{k=1}^n (c_1 + c_2 + c_3) \geq n$$

- ▶ luego $T_{FIB2}(n) \in \Omega(n)$, y por lo tanto también $T_{FIB2}(n) \in \Theta(n)$



Notación Asintótica Condicional

- ▶ muchos algoritmos son más fáciles de analizar si se restringe la atención a instancias cuyos tamaños satisfacen determinadas condiciones

$$O(g(n) \mid P(n)) = \{ f(n) : \exists c \in \mathbf{R}^+ \exists n_0 \in \mathbf{N}, \text{ tal que} \\ f(n) \leq cg(n) \text{ para todo } n \geq n_0 \\ \text{siempre que } P(n) \}$$

- ▶ análogamente, se definen $\Omega(g(n) \mid P(n))$ notación omega condicional, y $\Theta(g(n) \mid P(n))$ notación Θ condicional



- ▶ por ejemplo, $t(n) \in \Theta(n^2 \mid n = 2^k)$ significa que si n es potencia de 2 entonces $t(n) \in \Theta(n^2)$
- ▶ nada se está afirmando sobre $t(n)$ si n no es potencia de 2

NÚMERO DE FIBONACCI

Algoritmo: algoritmo iterativo complejo

$$T_{FIB3}(n) = c1 + \sum_{k=1}^{\log n} c2 + \sum_{k=1}^{\log n} c3$$

► si $n = 2^k$ entonces

$$T_{FIB3}(n) = c1 + \sum_{k=1}^{\log n} c3 \leq d \log n$$

► y entonces $T_{FIB3}(n) \in O(\log n \mid n = 2^k)$



- ▶ si $n = 2^k - 1$ entonces

$$T_{FIB3}(n) = c1 + \sum_{k=1}^{\log n} (c2 + c3) \leq e \log n$$

- ▶ y entonces $T_{FIB3}(n) \in O(\log n \mid n = 2^k - 1)$
- ▶ analizando que este último caso es el peor de los casos posible se puede concluir que $T_{FIB3}(n) \in O(\log n)$

Regla de las Funciones de Crecimiento Suave

- sirve para extender lo analizado condicionalmente a todos los tamaños de entrada

Teorema 1

Sea $f : \mathbf{N} \longrightarrow \mathbf{R}^+$ una función de crecimiento suave, y $t : \mathbf{N} \longrightarrow \mathbf{R}^+$ una función eventualmente no decreciente. Luego siempre que $t(n) \in \Theta(f(n) \mid n = b^k)$ para algún entero $b \geq 2$, entonces $t(n) \in \Theta(f(n))$



Definición

una función $f(n) : \mathbf{N} \longrightarrow \mathbf{R}^+$ es *eventualmente no decreciente* si existe $n_0 \in \mathbf{N}$ tal que para todo $n \geq n_0$ vale $f(n) \leq f(n+1)$

Definición

una función $f(n) : \mathbf{N} \longrightarrow \mathbf{R}^+$ es *de crecimiento suave* si existe $b \in \mathbf{N}, b \geq 2$ tal que $f(n)$ es eventualmente no decreciente y $f(bn) \in O(f(n))$



- ▶ la mayoría de las funciones que se encuentran son de crecimiento suave: $\log n$, n , $n \log n$, n^2 , o cualquier polinomio con coeficiente principal positivo
- ▶ funciones tales como $n^{\log n}$, 2^n o $n!$ no son de crecimiento suave
- ▶ reglas análogas también son válidas para $O(\cdot)$ y $\Omega(\cdot)$

Ejemplo

- ▶ si $t(n)$ es

$$t(n) = \begin{cases} a & \text{si } n = 1 \\ 4t(\lceil n/2 \rceil) + bn & \text{sino} \end{cases}$$

- ▶ entonces es fácil probar (usando los métodos de resolución de recurrencias que se verán) que $t(n) = (a + b)n^2 - bn$ si $n = 2^k$, ie $t(n) \in \Theta(n^2 \mid n = 2^k)$
- ▶ luego, como n^2 es una función de crecimiento suave y $t(n)$ es eventualmente no decreciente (**¿porqué?**) se puede aplicar la regla de las funciones de crecimiento suave y concluir que $t(n) \in \Theta(n^2)$ para todo n



- ▶ es necesario un manejo fluído de las siguientes estructuras de datos:
 - ▶ Arreglos y Matrices
 - ▶ Listas simplemente enlazadas, Pilas y Colas
 - ▶ Grafos, implementados mediante matriz o lista de adyacencia
 - ▶ árboles
 - ▶ Tablas Asociativas (*Hash*)
 - ▶ Colas con Prioridad (*Heaps*), implementados por árboles binarios completos
 - ▶ Conjuntos Disjuntos



ORDENAR UN ARREGLO

Algoritmo: **Heapsort** (ordenamiento por construcción de un *heap*)

```

FUNCTION Heapsort (A)
  Construir Heap(A)
  FOR i ::= n DOWNTO 2
    A[1] <=> A[i]
    A.tamaño- -
    A.heapify(1)
  ENDFOR

```

costo	veces
$\Theta(n)$	1
c_1	$\sum_{i=2}^n 1$
c_2	$\sum_{i=2}^n 1$
c_3	$\sum_{i=2}^n 1$
$\Theta(\log n)$	$\sum_{i=2}^n 1$

- ▶ calculando el tiempo de ejecución se tiene:

$$\begin{aligned}T_H(n) &= \Theta(n) + \sum_{i=2}^n (c_1 + c_2 + c_3 + \Theta(\log n)) = \\&= \Theta(n) + \Theta(n \log n) \in \Theta(n \log n)\end{aligned}$$

- ▶ es fundamental en este ejemplo usar la implementación más eficiente para las operaciones de la estructura de datos

Secuencia

Algoritmo A

P1

P2

- ▶ sea $t_A(n)$ la cantidad de recursos a analizar.
- ▶ si $P1$ insume $\Theta(f_1(n))$ recursos y $P2$ insume $\Theta(f_2(n))$ recursos, entonces

$$\begin{aligned} t_A(n) &= \Theta(f_1(n)) + \Theta(f_2(n)) = \Theta(f_1(n) + f_2(n)) = \\ &= \Theta(\max(f_1(n), f_2(n))) \end{aligned}$$



Condicional

```
Algoritmo  A
  IF  (X)
    P1
  ELSE
    P2
  ENDIF
```

- el tiempo en el peor de los casos es

$$\begin{aligned}t_A(n) &= t_X(n) + \max(t_{P1}(n), t_{P2}(n)) = \\ &= O(\max(t_X(n), t_{P1}(n), t_{P2}(n)))\end{aligned}$$



► y también

$$\begin{aligned} t_A(n) &\geq c + \max(\Theta(f_1(n)), \Theta(f_2(n))) = \\ &= \Omega(\max(c, f_1(n), f_2(n))) \end{aligned}$$

► si el $O(\cdot)$ y el $\Omega(\cdot)$ coinciden, entonces se puede definir el $\Theta(\cdot)$



Ciclo FOR

Algoritmo B

```
FOR i ::= 1 TO m
```

```
    P(i)
```

```
ENDFOR
```

- ▶ sean c_1 , c_2 , c_3 los costos de las operaciones elementales
- ▶ si $P(i)$ insume t recursos (no depende de i ni de m) entonces

$$\begin{aligned}t_B(n) &= c_1 + (m+1)c_3 + mt + mc_2 = \\ &= (c_2 + c_3 + t)m + (c_1 + c_3) \in \Theta(mt)\end{aligned}$$



- ▶ si $P(i)$ insume $t(i)$ recursos (dependiendo de i , del tamaño n de la instancia, o de cada instancia en particular) entonces

$$t_B(n) = \sum_{i=1}^m t(i)$$

- ▶ para obtener el $O(\cdot)$ o el $\Omega(\cdot)$ de esta función se pueden usar las distintas propiedades vistas para obtener la notación asintótica

ORDENAR UN ARREGLO

Algoritmo: Ordenamiento por Selección

```

FOR i ::= 1 TO n-1
  ind ::= i;  min ::= A[i]
  FOR j ::= i+1 TO n
    IF (A[j]<min)
      min ::= A[j]
      ind ::= j
    ENDIF
  ENDFOR
  A[ind] ::= A[i]; A[i] ::= min
ENDFOR

```

costo

veces

*a**n**b**n* - 1*c* $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n+1} 1$ *c* $\sum_{i=1}^{n-1} \sum_{j=i+1}^n 1$ *c* $\sum_{i=1}^{n-1} \sum_{j=i+1}^n 1$ *c* $\sum_{i=1}^{n-1} \sum_{j=i+1}^n 1$ *b**n* - 1

- calculando la cantidad de recursos se tiene

$$\begin{aligned}T_S(n) &= a + \sum_{i=1}^{n-1} (a + b + (n-i)c) \\&= a + \sum_{i=1}^{n-1} (a + b + cn) - c \sum_{i=1}^{n-1} i = \\&= a + (n-1)(a + b + cn) - cn(n-1)/2 \\&= a + \frac{cn^2}{2} + (a + b - \frac{c}{2})n - (a + b) \in \Theta(n^2)\end{aligned}$$

NÚMERO DE FIBONACCI

Algoritmo: primer algoritmo iterativo (sumas y restas son operaciones elementales)

```
Function FIB2 (n)
```

```
    i ::= 1; j ::= 0
```

```
    FOR k ::= 1 TO n
```

```
        j ::= i+j
```

```
        i ::= j-i
```

```
    ENDFOR
```

```
    RETURN j
```

	costo	veces
b		1
c_1		$\sum_{k=1}^{n+1} 1$
c_2		$\sum_{i=1}^n 1$
c_3		$\sum_{i=1}^n 1$
d		1

- calculando el tiempo de ejecución se tiene

$$\begin{aligned}T_{FIB2}(n) &= b + c_1 + \sum_{k=1}^n (c_1 + c_2 + c_3) + d = \\&= (b + d) + (c_1 + c_2 + c_3)n \in \Theta(n)\end{aligned}$$

- si la suma y la resta no son operaciones elementales (operan sobre números muy grandes) entonces

```
Function FIB2 (n)
```

```
    i ::= 1; j ::= 0
```

```
    FOR k ::= 1 TO n
```

```
        j ::= i+j
```

```
        i ::= j-i
```

```
    ENDFOR
```

```
    RETURN j
```

costo

veces

 b

1

 c_1 $\sum_{k=1}^n 1$ $c_2 * \text{tamaño}(j)$ $\sum_{i=1}^n 1$ $c_3 * \text{tamaño}(j)$ $\sum_{i=1}^n 1$ d

1

► resultando

$$\begin{aligned}T_{FIB2}(n) &= b + \sum_{k=1}^n (c_1 + (c_2 + c_3) * \text{tamaño}(j)) + d = \\&\leq (b + d) + \sum_{k=1}^n (c_1 + (c_2 + c_3) * \text{tamaño}(F_k)) \\&\leq (b + d) + nc_1 + \sum_{k=1}^n dk \text{ por } \text{tamaño}(F_k) \in \Theta(k) \\&= (b + d) + nc_1 + d \frac{n(n+1)}{2} = \\&= (b + d) + nc_1 + \frac{d}{2}n^2 + \frac{d}{2}n \in O(n^2)\end{aligned}$$

Ciclos WHILE y REPEAT

- ▶ no es tan fácil para los casos de ciclos **repeat** o **while**, no se sabe cuántas veces serán ejecutados.
- ▶ algunas de las técnicas a aplicar pueden ser:
 1. encontrar una función en las variables involucradas cuyo valor decrezca en cada iteración, y que sea siempre positiva
 2. tratar la iteración como si fuese un procedimiento recursivo, y aplicar el método para recursividad
 3. elegir como cota del cuerpo del bucle el tiempo de ejecución de una de sus sentencias, la cual se denomina **barómetro**. Luego se debe contar cuántas veces se ejecuta el barómetro
- ▶ ningún método es aplicable para todos los casos, y solo a través de la experiencia se puede detectar cuál usar



MÁXIMO COMÚN DIVISOR

Algoritmo: Algoritmo de Euclides

```
Function EUCLIDES (m, n)
    WHILE m > 0
        temp ::= m
        m ::= n mod m
        n ::= temp
    ENDWHILE
    RETURN n
```

- ▶ se puede observar las propiedades:
 - ▶ $n_i = m_{i-1}$, $m_i = n_{i-1} \bmod m_{i-1}$ siempre que $i \geq 1$
 - ▶ $n_i \geq m_i$ siempre que $i > 1$
 - ▶ para todo n, m tal que $n \geq m$ vale $n \bmod m < n/2$
 - ▶ $n_i = m_{i-1} = n_{i-2} \bmod m_{i-2} < n_{i-2}/2$ si $i > 2$
- ▶ luego, en dos iteraciones n_0 se reduce a menos de la mitad; en cuatro a menos del cuarto; etc. Como $m_i > 0$ entonces no puede haber más de $2 \log_2 n_0$ iteraciones. Y $T(n) \in O(\log n)$.

ORDENAR UN ARREGLO

Algoritmo: Ordenamiento por Cubículos (enteros hasta s)

array U[1..s] ::= 0 $\Theta(n)$

FOR i ::= 1 TO n

 k ::= T[i]; U[k]++ $\Theta(1)$

ENDFOR

i ::= 0

FOR k ::= 1 TO s

 WHILE U[k] != 0 barómetro

 T[i++] ::= k; U[k]--

 ENDWHILE

ENDFOR



- ▶ el barómetro se ejecuta $U[k]_0 + 1$ veces por cada k
- ▶ luego el tiempo total es: $\sum_{k=1}^s (U[k]_0 + 1)\Theta(1)$
- ▶ y vale

$$\begin{aligned}T(n) &= \Theta(n) + \Theta(1) + \sum_{k=1}^s (U[k]_0 + 1)\Theta(1) = \\&= \Theta(n) + \sum_{k=1}^s U[k]_0\Theta(1) + \sum_{k=1}^s \Theta(1) \\&= \Theta(n) + \Theta(n) + \Theta(s) \in \Theta(\max(n, s))\end{aligned}$$

- ▶ el problema de este algoritmo es el límite máximo de los números a utilizar, y el espacio de memoria auxiliar



- ▶ una simple inspección del algoritmo da origen a una **recurrencia**, que “simula” el flujo de control del algoritmo

```
function F(n)
    IF (x)
        P1(n)
    ELSE
        P2(n)
        F(m);    % con m<n
    ENDIF
```

- ▶ $t(n) \in O(\max(t_X(n), t_{P1}(n), t_{P2}(n) + t(m)))$
- ▶ luego se debe aplicar algún método para resolver la recurrencia

ELEMENTO MAYOR

```
Function MAXIMO(T)
  IF n=1
    RETURN T[1]
  ELSE
    x ::= MAXIMO(T[1..n-1])
    IF (x>T[n])
      RETURN x
    ELSE
      RETURN T[n]
    ENDIF
  ENDIF
```

- genera la siguiente recurrencia:

$$T_{MAX}(n) = \begin{cases} a & \text{si } n = 1. \\ b + T_{MAX}(n-1) & \text{si } n > 1 \end{cases}$$

NÚMERO DE FIBONACCI

Algoritmo: algoritmo recursivo

```
function FIB1 (n)
  IF n < 2
    RETURN n
  ELSE
    RETURN (FIB1 (n-1) + FIB1 (n-2) )
  ENDIF
```

► que genera la recurrencia

$$T_{FIB1}(n) = \begin{cases} a & \text{si } n < 2 \\ T_{FIB1}(n-1) + T_{FIB1}(n-2) + b & \text{si } n \geq 2 \end{cases}$$



- ▶ veremos dos técnicas básicas y una auxiliar que se aplican a diferentes clases de recurrencias:

Técnicas de
Resolución de
Recurrencias

{ método del teorema maestro
método de la ecuación característica
cambio de variables

- ▶ no analizaremos su demostración formal, sólo consideraremos su aplicación para las recurrencias generadas a partir del análisis de algoritmos



Método del Teorema Maestro

- ▶ se aplica en casos como:

$$T(n) = \begin{cases} 5 & \text{si } n = 0 \\ 9T(n/3) + n & \text{si } n \neq 0 \end{cases}$$

- ▶ es importante identificar:
 - ▶ la cantidad de llamadas recursivas
 - ▶ el cociente en el que se divide el tamaño de las instancias
 - ▶ la sobrecarga extra a las llamadas recursivas

Teorema 2

Sean $a \geq 1$, $b > 1$ constantes, $f(n)$ una función y $T(n)$ una recurrencia definida sobre los enteros no negativos de la forma $T(n) = aT(n/b) + f(n)$, donde n/b puede interpretarse como $\lfloor n/b \rfloor$ o $\lceil n/b \rceil$. Entonces valen:

1. si $f(n) \in O(n^{\log_b a - \varepsilon})$ para algún $\varepsilon > 0$ entonces $T(n) \in \Theta(n^{\log_b a})$.
2. si $f(n) \in \Theta(n^{\log_b a})$ entonces $T(n) \in \Theta(n^{\log_b a} \lg n)$.
3. si $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ para algún $\varepsilon > 0$, y satisface $af(n/b) \leq cf(n)$ para alguna constante $c < 1$, entonces $T(n) \in \Theta(f(n))$.

Ejemplos:

1. si $T(n) = 9T(n/3) + n$ entonces $a = 9$, $b = 3$, se aplica el caso 1 con $\varepsilon = 1$ y $T(n) \in \Theta(n^2)$
2. si $T(n) = T(2n/3) + 1$ entonces $a = 1$, $b = 3/2$, se aplica el caso 2 y $T(n) = \Theta(\lg n)$
3. si $T(n) = 3T(n/4) + n \lg n$ entonces $a = 3$, $b = 4$, $f(n) \in \Omega(n^{\log_4 3 + 0.2})$ y $3(n/4) \lg(n/4) \leq 3/4 n \lg n$, por lo que se aplica el caso 3 y $T(n) \in \Theta(n \lg n)$
4. si $T(n) = 2T(n/2) + n \lg n$, no se puede aplicar el caso 3 porque $f(n) = n \lg n \notin \Omega(n^{1+\varepsilon})$ para cualquier $\varepsilon > 0$

Método de la Ecuación Característica

- ▶ se aplica a ciertas recurrencias lineales con coeficientes constantes como:

$$T(n) = \begin{cases} 5 & \text{si } n = 0 \\ 10 & \text{si } n = 1 \\ 5T(n-1) + 8T(n-2) + 2n & \text{si } n > 1 \end{cases}$$

- ▶ en general, para recurrencias de la forma:

$$T(n) = a_1 T(n-1) + a_2 T(n-2) + \cdots + a_k T(n-k) + b^n p(n)$$

donde a_i , $1 \leq i \leq k$, b son constantes y $p(n)$ es un polinomio en n de grado s



Ejemplos:

► en $t(n) = 2t(n-1) + 3^n$,
 $s = 0$

$$a_1 = 2, b = 3, p(n) = 1,$$

► en $t(n) = t(n-1) + t(n-2) + n$,
 $b = 1, p(n) = n, s = 1$

$$a_1 = 1, a_2 = 1,$$

- para resolver la recurrencia

$$T(n) = a_1 T(n-1) + a_2 T(n-2) + \cdots + a_k T(n-k) + b^n p(n):$$

1. encontrar las raíces no nulas de la ecuación característica:

$$(x^k - a_1 x^{k-1} - a_2 x^{k-2} - \cdots - a_k)(x - b)^{s+1} = 0$$

Raíces: $r_i, 1 \leq i \leq l \leq k$, cada una con multiplicidad m_i .

2. las soluciones son de la forma de combinaciones lineales de estas raíces de acuerdo a su multiplicidad

$$T(n) = \sum_{i=1}^l \sum_{j=1}^{m_i} c_{ij} n^{j-1} r_i^n$$

3. si se necesita, se encuentran valores para las constantes $c_{ij}, 1 \leq i \leq l, 0 \leq j \leq m_i - 1$ y $d_i, 0 \leq i \leq s - 1$ según la recurrencia original y las condiciones iniciales (valores de la recurrencia para $n = 0, 1, \dots$)



Ejemplo:

$$T(n) = \begin{cases} 0 & \text{si } n=0 \\ 2T(n-1) + 1 & \text{si } n > 0 \end{cases}$$

1. si $b = 1$ y $p(n) = 1$ de grado 0, la **ecuación característica** $(x - 2)(x - 1)^{0+1} = 0$, con $r_1 = 2$, $m_1 = 1$ y $r_2 = 1$, $m_2 = 1$
2. la **solución general** es de la forma $T(n) = c_{11}2^n + c_{21}1^n$.
3. a partir de las condiciones iniciales se encuentra:

$$\begin{aligned} c_{11} + c_{21} &= 0 & \text{de } n = 0 \\ 2c_{11} + c_{21} &= 1 & \text{de } n = 1 \end{aligned}$$

de donde $c_{11} = 1$ y $c_{21} = -1$.

4. la **solución** es $T(n) = 2^n - 1$



Ejemplo:

$$T(n) = \begin{cases} n & \text{si } n=0,1,2 \\ 5T(n-1) - 8T(n-2) + 4T(n-3) & \text{si } n > 2 \end{cases}$$

1. si $b = 0$ y $p(n) = 1$, la **ecuación característica** es $x^3 - 5x^2 + 8x - 4 = 0$, con $r_1 = 1, m_1 = 1$ y $r_2 = 2, m_2 = 2$
2. la **solución general** es de la forma $T(n) = c_{11}1^n + c_{21}2^n + c_{22}n2^n$.
3. a partir de las condiciones iniciales se obtienen:

$$\begin{aligned} c_{11} + c_{21} &= 0 & \text{de } n=0 \\ c_{11} + 2c_{21} + 2c_{22} &= 1 & \text{de } n=1 \\ c_{11} + 4c_{21} + 8c_{22} &= 2 & \text{de } n=2 \end{aligned}$$

de donde $c_{11} = -2$, $c_{21} = 2$ y $c_{22} = -1/2$

4. la **solución** es entonces la función $T(n) = 2^{n+1} - n2^{n-1} - 2$



Ejemplo: número de Fibonacci

$$F(n) = \begin{cases} n & \text{si } n=0,1 \\ F(n-1) + F(n-2) & \text{si } n \geq 2 \end{cases}$$

1. si $b = 0$ y $p(n) = 1$, la **ecuación característica** es $x^2 - x - 1 = 0$, con raíces $\phi = \frac{1+\sqrt{5}}{2}$ y $\hat{\phi} = \frac{1-\sqrt{5}}{2}$
2. la **solución general** es $F(n) = c_{11}\phi^n + c_{21}\hat{\phi}^n$.
3. a partir de las condiciones iniciales se obtienen:

$$\begin{aligned} c_{11} + c_{21} &= 0 & \text{de } n=0 \\ c_{11}\phi + c_{21}\hat{\phi} &= 1 & \text{de } n=1 \end{aligned}$$

cuyas soluciones son $c_{11} = 1/\sqrt{5}$ y $c_{21} = -1/\sqrt{5}$

4. la **solución** es $F(n) = \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n)$.



Cambio de Variable

- ▶ por ejemplo para la recurrencia

$$T(n) = \begin{cases} a & \text{si } n=1 \\ 2T(n/2) + n \log_2 n & \text{sino} \end{cases}$$

no se puede ninguno de los dos métodos anteriores

- ▶ se define una nueva recurrencia $S(i) = T(2^i)$, con el objetivo de llevarla a una forma en la que se pueda resolver siguiendo algún método anterior
- ▶ el caso general queda

$$S(i) = T(2^i) = 2T(2^i/2) + 2^i i = 2T(2^{i-1}) + i2^i = 2S(i-1) + i2^i$$

con $b = 2$ y $p(i) = i$ de grado 1



- ▶ la ecuación característica de esta recurrencia es $(x - 2)(x - 2)^{1+1} = 0$ con raíz 2 de grado 3
- ▶ la solución es entonces $S(i) = c_{11}2^i + c_{12}i2^i + c_{13}i^22^i$
- ▶ volviendo a la variable original queda $T(n) = c_{11}n + c_{12}(\log_2 n)n + c_{13}(\log_2 n)^2n$.
- ▶ se pueden obtener los valores de las constantes sustituyendo esta solución en la recurrencia original:

$$T(n) - 2T(n/2) = n\log_2 n = (c_{12} - c_{13})n + 2c_{13}n(\log_2 n)$$

de donde $c_{12} = c_{13}$ y $2c_{12} = 1$

- ▶ por lo tanto $T(n) \in \Theta(n \log^2 n \mid n \text{ es potencia de } 2)$
- ▶ si se puede probar que $T(n)$ es eventualmente no decreciente, por la **regla de las funciones de crecimiento suave** se puede extender el resultado a todos los n (dado que $n \log^2 n$ es de crecimiento suave). En este caso $T(n) \in \Theta(n \log^2 n)$