Algoritmos y Complejidad

Repaso de Técnicas y Herramientas

Pablo R. Fillottrani

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

Primer Cuatrimestre 2024

Técnicas y Herramientas

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

Herramientas ya conocidas. Leer apunte y referencias disponible en el curso Moodle-UNS.

- 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

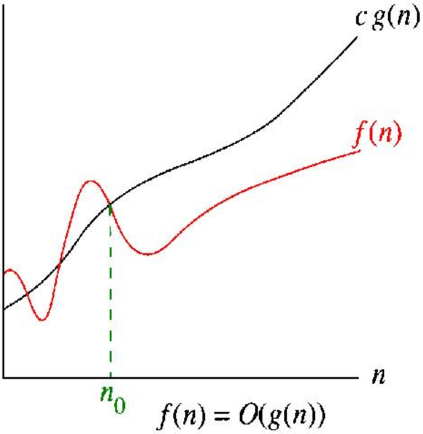
- no interesa conocer los valores absolutos de las funciones
- permitir una caracterización simple de la eficiencia de un algoritmo y comparar las performances relativas de distintos algoritmos
- independizar el análisis de los algoritmos de condiciones específicas de implementación: lenguaje de programación, 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 $O(\cdot)$

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

- 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 $c = 1, n_0 = 4$



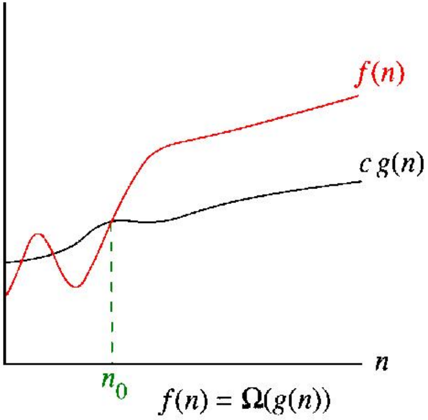
Ejemplos:

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

Notación $\Omega(\cdot)$

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

- 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 $c = 1/3, n_0 = 1$



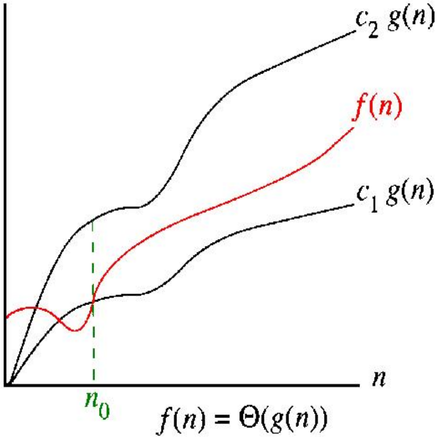
Ejemplos:

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

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 $c = 1/5, d = 1, n_0 = 1$.



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 $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in \mathbf{R}^+$ entonces $f(n) \in O(g(n))$ y $g(n) \in O(f(n))$
- si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ 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:

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

- luego $T(n) \in O(n^2)$.

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

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

- 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 $T(n) \in \Theta(n^2)$

- 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

ORDENAR UN ARREGLO

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

	costo	veces
`FUNCTION Heapsort(A)`		
`Construir Heap(A)`	$\Theta(n)$	1
`FOR i ::= n DOWNTO 2`	c_1	$\sum_{i=2}^{n} 1$
`    A[1] <=> A[i]`	c_2	$\sum_{i=2}^{n} 1$
`   A.tamaño- -`	c_3	$\sum_{i=2}^{n} 1$
`   A.heapify(1)`	$\Theta(\log n)$	$\sum_{i=2}^{n} 1$
`ENDFOR`		

- 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

Código	costo	veces
`FOR i ::= 1 TO n-1`	a	n
`   ind ::= i;  min ::= A[i]`	b	$n-1$
`   FOR j ::= i+1 TO n`	c	$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n+1} 1$
`      IF (A[j]<min)`	c	$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 1$
`         min ::= A[j]`	c	$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 1$
`         ind ::= j`	c	$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 1$
`      ENDIF`		
`   ENDFOR`		
`   A[ind] ::= A[i]; A[i] ::= min`	b	$n-1$
`ENDFOR`		

- calculando la cantidad de recursos se tiene

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

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

Recursividad

- 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
```

- $t(n) \in O(max(t_{P1}(n), t_{P2}(n) + t(m)))$
- 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:

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