UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica

etsinf

Tema 2. Análisis

Programación (PRG)

Jorge González Mollá

Departamento de Sistemas Informáticos y Computación

Índice

Motivación

- Cuando se está analizando el coste temporal de un algoritmo, lo que nos interesa es ver el tipo de crecimiento que presenta.
- Si se identifican los tipos de crecimiento o funciones típicas, entonces podemos comparar diversos algoritmos, y así, elegir.
- Por tanto, el análisis a priori de un algoritmo consistirá en medir el tipo o tasa de crecimiento de sus funciones de coste y expresarlo de forma adecuada mediante notación asintótica.

Notación asintótica

- Sea $f : \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$. El conjunto de las funciones del orden de f(n), denotado por O(f(n)), se define como:

$$O(f(n)) = \{ g : \mathbb{N} \rightarrow \mathbb{R}^{\geq 0} \mid \exists\, c \in \mathbb{R}^{>0}, n_0 \in \mathbb{N} \text{ tales que, } \forall n \geq n_0, g(n) \leq c \cdot f(n) \}$$

- $g(n) \in O(f(n))$ ⇨ se dice que "g(n) es del orden de f(n)" y que "f(n) es asintóticamente una cota superior de g(n)".

- Sea $f : \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$. El conjunto de las funciones al menos del orden de f(n), denotado por $\Omega(f(n))$, se define como:

$$\Omega(f(n)) = \{ g : \mathbb{N} \rightarrow \mathbb{R}^{\geq 0} \mid \exists\, c \in \mathbb{R}^{>0}, n_0 \in \mathbb{N} \text{ tales que, } \forall n \geq n_0, g(n) \geq c \cdot f(n) \}$$

- $g(n) \in \Omega(f(n))$ ⇨ se dice que "g(n) es al menos del orden de f(n)" y que "f(n) es asintóticamente una cota inferior de g(n)".

Notación asintótica

- Sea $f: \mathbb{N} \to \mathbb{R}^{\geq 0}$. El conjunto de las funciones exactamente del orden de f(n), denotado por $\Theta(f(n))$, se define como:

$$\Theta(f(n)) = \{ g : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \exists\, c_1, c_2 \in \mathbb{R}^{>0} \text{ y } n_0 \in \mathbb{N} \text{ tales que, } \forall n \geq n_0,$$
$$0 \leq c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n) \}$$

- $g(n) \in \Theta(f(n))$ ⇨ se dice que "g(n) es del orden exacto de f(n)" y que "f(n) es asintóticamente una cota superior e inferior de g(n)"

- El conjunto de la funciones exactamente del orden de f(n) está formado por todas aquellas funciones g(n) que, simultáneamente, están acotadas tanto superior como inferiormente por la función f(n)

$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

Notación asintótica

- Las razones que apoyan este tipo de análisis son:

1. Para valores de n grandes, la función se encuentra completamente determinada por su término dominante.

2. El valor exacto del coeficiente del término dominante de la función suele variar de un entorno de programación a otro.

3. La notación asintótica establece una relación de orden entre funciones de coste en función del término dominante

$$O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(n^3) \subset O(2^n)$$

Notación asintótica

- Tabla de funciones típicas usando notación asintótica

Función	Nombre	Notación Asintótica
c	constante	$\Theta(1)$
$\log n$	logarítmica	$\Theta(\log n)$
$\log^2 n$	logarítmica al cuadrado	$\Theta(\log^2 n)$
n	lineal	$\Theta(n)$
$n \log n$	n log n	$\Theta(n \log n)$
n^2	cuadrática	$\Theta(n^2)$
n^3	cúbica	$\Theta(n^3)$
2^n	exponencial	$\Theta(2^n)$

Ejemplo: Coste Exponencial

- A medida que los ordenadores se vuelven más y más rápidos puede parecer que apenas merece la pena invertir nuestro tiempo en diseñar algoritmos más eficientes. ¿Y si esperamos a la siguiente generación de ordenadores? ¿Por qué hay que buscar la eficiencia?.

- Supongamos que para resolver un problema concreto se dispone de un algoritmo exponencial y de un ordenador que ejecuta dicho algoritmo para tamaño n en $10^{-4} \cdot 2^n$ segundos.

n	Tiempo
10	$10^{-4} \times 2^{10}$ s., aprox. 1 décima de seg.
20	$10^{-4} \times 2^{20}$ s., aprox. 2 minutos
30	$10^{-4} \times 2^{30}$ s., más de 1 día de cálculo
¡38! ⇦	1 año

- Supongamos que se compra un ordenador nuevo cien veces más rápido que el anterior. Ahora se puede resolver el mismo problema para tamaño n en $10^{-6} \cdot 2^n$ segundos. En 1 año ⇨ $n < ¡45!$

- En general, si antes se resolvía un ejemplar de tamaño n en un tiempo dado, la nueva máquina resolverá tamaños como mucho de $n+\log_2 100$, aprox. $n+7$, en el mismo tiempo.

Ejemplo: Coste Cúbico

- Supongamos que se invierte en algoritmia y, por la misma cantidad de dinero, se encuentra un algoritmo cúbico que en la máquina original resuelve un ejemplar de tamaño n en $10^{-2} \cdot n^3$ segundos.

n		Tiempo
10		$10^{-2}x10^3$ = 10 seg.
20		$10^{-2}x20^3$ = entre 1 y 2 minutos
30		$10^{-2}x30^3$ = 4 minutos y medio
>200		1 día
¡1500!	⇦	1 año

- El nuevo algoritmo no sólo ofrece una mejora mucho mayor que la adquisición del nuevo hardware, sino que además, suponiendo que uno pueda permitirse ambas cosas, hará que esta adquisición sea todavía más rentable.

- En general, las mejoras en un algoritmo por cambio de lenguaje de programación, de ordenador, ahorro de variables y/o de instrucciones, etc. son mucho menos importantes que las mejoras por una estrategia de diseño que implique una tasa de crecimiento inferior.