Algorítmica: Análisis de Algoritmos

Conrado Martínez
U. Politècnica Catalunya

Q1-2011-2012

- Eficiencia de un algoritmo = consumo de recursos de cómputo: tiempo de ejecución y espacio de memoria
- Análisis de algoritmos $\rightarrow$ Propiedades sobre la eficiencia de algoritmos
  - Comparar soluciones algorítmicas alternativas
  - Predecir los recursos que usará un algoritmo o ED
  - Mejorar los algoritmos o EDs existentes y guiar el diseño de nuevos algoritmos

En general, dado un algoritmo A cuyo conjunto de entradas es $\mathcal{A}$ su eficiencia o *coste* (en tiempo, en espacio, en número de operaciones de E/S, etc.) es una función T de $\mathcal{A}$ en $\mathbb{N}$ (o $\mathbb{Q}$ o $\mathbb{R}$, según el caso):

$$\begin{aligned} T : \mathcal{A} &\to \mathbb{N} \\ \alpha &\to T(\alpha) \end{aligned}$$

Ahora bien, caracterizar la función T puede ser muy complicado y además proporciona información inmanejable, difícilmente utilizable en la práctica.

Sea $\mathcal{A}_n$ el conjunto de entradas de tamaño n y $T_n : \mathcal{A}_n \to \mathbb{N}$ la función T restringida a $\mathcal{A}_n$.

- *Coste en caso mejor*:

$$T_{\text{mejor}}(n) = \text{mín}\{T_n(\alpha) \mid \alpha \in \mathcal{A}_n\}.$$

- *Coste en caso peor*:

$$T_{\text{peor}}(n) = \text{máx}\{T_n(\alpha) \mid \alpha \in \mathcal{A}_n\}.$$

- *Coste promedio*:

$$\begin{aligned} T_{\text{avg}}(n) &= \sum_{\alpha \in \mathcal{A}_n} \Pr(\alpha)\, T_n(\alpha) \\ &= \sum_{k \geq 0} k \, \Pr(T_n = k). \end{aligned}$$

1. Para todo $n \geq 0$ y para cualquier $\alpha \in \mathcal{A}_n$

$$T_{\text{mejor}}(n) \leq T_n(\alpha) \leq T_{\text{peor}}(n).$$

2. Para todo $n \geq 0$

$$T_{\text{mejor}}(n) \leq T_{\text{avg}}(n) \leq T_{\text{peor}}(n).$$

Estudiaremos generalmente sólo el coste en caso peor:

1. Proporciona garantías sobre la eficiencia del algoritmo, el coste nunca excederá el coste en caso peor
2. Es más fácil de calcular que el coste promedio

Una característica esencial del coste (en caso peor, en caso mejor, promedio) es su tasa de crecimiento

Ejemplo

1. Funciones lineales: $f(n) = a \cdot n + b \Rightarrow f(2n) \approx 2 \cdot f(n)$
2. Funciones cuadráticas:
   $q(n) = a \cdot n^2 + b \cdot n + c \Rightarrow q(2n) \approx 4 \cdot q(n)$

Se dice que las funciones lineales y las cuadráticas tienen tasas de crecimiento distintas. También se dice que son de órdenes de magnitud distintos.

$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	262144
5	32	160	1024	32768	$6{,}87 \cdot 10^{10}$
6	64	384	4096	262144	$4{,}72 \cdot 10^{21}$
			$\dots$		
ℓ	N	L	C	Q	E
$\ell + 1$	$2N$	$2(L + N)$	$4C$	$8Q$	E^2

Los factores constantes y los términos de orden inferior son irrelevantes desde el punto de vista de la tasa de crecimiento: p.e. $30n^2 + \sqrt{n}$ tiene la misma tasa de crecimiento que $2n^2 + 10n \Rightarrow$ notación asintótica

Definición

Dada una función $f : \mathbb{R}^+ \to \mathbb{R}^+$ la clase $\mathcal{O}(f)$ (O-grande de f) es

$$\mathcal{O}(f) = \{g : \mathbb{R}^+ \to \mathbb{R}^+ \mid \exists n_0 \; \exists c \; \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$$

En palabras, una función g está en $\mathcal{O}(f)$ si existe una constante c tal que $g < c \cdot f$ para toda n a partir de un cierto punto (n_0).

Aunque $\mathcal{O}(f)$ es un conjunto de funciones por tradición se escribe a veces $g = \mathcal{O}(f)$ en vez de $g \in \mathcal{O}(f)$. Sin embargo, $\mathcal{O}(f) = g$ no tiene sentido.
Propiedades básicas de la notación $\mathcal{O}$:

1. Si $\lim_{n\to\infty} g(n)/f(n) < +\infty$ entonces $g = \mathcal{O}(f)$
2. Es reflexiva: para toda función $f : \mathbb{R}^+ \to \mathbb{R}^+$, $f = \mathcal{O}(f)$
3. Es transitiva: si $f = \mathcal{O}(g)$ y $g = \mathcal{O}(h)$ entonces $f = \mathcal{O}(h)$
4. Para toda constante $c > 0$, $\mathcal{O}(f) = \mathcal{O}(c \cdot f)$

Los factores constantes no son relevantes en la notación asintótica y los omitiremos sistemáticamente: p.e. hablaremos de $\mathcal{O}(n)$ y no de $\mathcal{O}(4 \cdot n)$; no expresaremos la base de logaritmos ($\mathcal{O}(\log n)$, ya que podemos pasar de una base a otra multiplicando por el factor apropiado:

$$\log_c x = \frac{\log_b x}{\log_b c}$$

Otras notaciones asintóticas son Ω (omega) y Θ (zita). La primera define un conjunto de funciones acotada inferiormente por una dada:

$$\Omega(f) = \{g : \mathbb{R}^+ \to \mathbb{R}^+ \mid \exists n_0 \; \exists c > 0 \; \forall n \geq n_0 : g(n) \geq c \cdot f(n)\}$$

La notación Ω es reflexiva y transitiva; si $\lim_{n\to\infty} g(n)/f(n) > 0$ entonces $g = \Omega(f)$. Por otra parte, si $f = \mathcal{O}(g)$ entonces $g = \Omega(f)$ y viceversa.

Se dice que $\mathcal{O}(f)$ es la clase de las funciones que crecen no más rápido que f. Análogamente, $\Omega(f)$ es la clase de las funciones que crecen no más despacio que f.
Finalmente,

$$\Theta(f) = \Omega(f) \cap \mathcal{O}(f)$$

es la clase de la funciones con la misma tasa de crecimiento que f.
La notación Θ es reflexiva y transitiva, como las otras. Es además simétrica: $f = \Theta(g)$ si y sólo si $g = \Theta(f)$. Si $\lim_{n\to\infty} g(n)/f(n) = c$ donde $0 < c < \infty$ entonces $g = \Theta(f)$.

Propiedades adicionales de las notaciones asintóticas (las inclusiones son estrictas):

1. Para cualesquiera constantes $\alpha < \beta$, si f es una función creciente entonces $\mathcal{O}(f^\alpha) \subset \mathcal{O}(f^\beta)$.
2. Para cualesquiera constantes a y $b > 0$, si f es creciente, $\mathcal{O}((\log f)^a) \subset \mathcal{O}(f^b)$.
3. Para cualquier constante $c > 0$, si f es creciente, $\mathcal{O}(f) \subset \mathcal{O}(c^f)$.

Los operadores convencionales (sumas, restas, divisiones, etc.) sobre clases de funciones definidas mediante una notación asintótica se extienden de la siguiente manera:

$$A \otimes B = \{h \mid \exists f \in A \wedge \exists g \in B : h = f \otimes g\},$$

donde A y B son conjuntos de funciones. Expresiones de la forma $f \otimes A$ donde f es una función se entenderá como $\{f\} \otimes A$.
Este convenio nos permite escribir de manera cómoda expresiones como $n + \mathcal{O}(\log n)$, $n^{\mathcal{O}(1)}$, ó $\Theta(1) + \mathcal{O}(1/n)$.

Regla de las sumas:

$$\Theta(f) + \Theta(g) = \Theta(f + g) = \Theta(\max\{f, g\}).$$

Regla de los productos:

$$\Theta(f) \cdot \Theta(g) = \Theta(f \cdot g).$$

Reglas similares se cumplen para las notaciones $\mathcal{O}$ y Ω.

Análisis de algoritmos iterativos

1. El coste de una operación elemental es $\Theta(1)$.
2. Si el coste de un fragmento S_1 es f y el de S_2 es g entonces el coste de $S_1; S_2$ es $f + g$.
3. Si el coste de S_1 es f, el de S_2 es g y el coste de evaluar B es h entonces el coste en caso peor de

   if B **then** S_1
   else S_2

   es $\max\{f + h, g + h\}$.

4. Si el coste de S durante la i-ésima iteración es f_i, el coste de evaluar B es h_i y el número de iteraciones es g entonces el coste T de

   while B **do**
   S

   es

$$T(n) = \sum_{i=1}^{i=g(n)} f_i(n) + h_i(n).$$

   Si $f = \max\{f_i + h_i\}$ entonces $T = \mathcal{O}(f \cdot g)$.

Análisis de algoritmos recursivos

El coste (en caso peor, medio, ...) de un algoritmo recursivo $T(n)$ satisface, dada la naturaleza del algoritmo, una ecuación recurrente: esto es, $T(n)$ dependerá del valor de T para tamaños menores. Frecuentemente, la recurrencia adopta una de las dos siguientes formas:

$$T(n) = a \cdot T(n-c) + g(n),$$
$$T(n) = a \cdot T(n/b) + g(n).$$

La primera corresponde a algoritmos que tiene una parte no recursiva con coste $g(n)$ y hacen a llamadas recursivas con subproblemas de tamaño $n-c$, donde c es una constante.
La segunda corresponde a algoritmos que tienen una parte no recursiva con coste $g(n)$ y hacen a llamadas recursivas con subproblemas de tamaño (aproximadamente) n/b, donde $b > 1$.

Teorema

Sea $T(n)$ el coste (en caso peor, en caso medio, ...) de un algoritmo recursivo que satisface la recurrencia

$$T(n) = \begin{cases} f(n) & \text{si } 0 \leq n < n_0 \\ a \cdot T(n-c) + g(n) & \text{si } n \geq n_0, \end{cases}$$

donde n_0 es una constante, $c \geq 1$, $f(n)$ es una función arbitraria y $g(n) = \Theta(n^k)$ para una cierta constante $k \geq 0$. Entonces

$$T(n) = \begin{cases} \Theta(n^k) & \text{si } a < 1 \\ \Theta(n^{k+1}) & \text{si } a = 1 \\ \Theta(a^{n/c}) & \text{si } a > 1. \end{cases}$$

Teorema

Sea $T(n)$ el coste (en caso peor, en caso medio, . . .) de un algoritmo recursivo que satisface la recurrencia

$$T(n) = \begin{cases} f(n) & \text{si } 0 \leq n < n_0 \\ a \cdot T(n/b) + g(n) & \text{si } n \geq n_0, \end{cases}$$

donde n_0 es una constante, $b > 1$, $f(n)$ es una función arbitraria y $g(n) = \Theta(n^k)$ para una cierta constante $k \geq 0$. Sea $\alpha = \log_b a$. Entonces

$$T(n) = \begin{cases} \Theta(n^k) & \text{si } \alpha < k \\ \Theta(n^k \log n) & \text{si } \alpha = k \\ \Theta(n^\alpha) & \text{si } \alpha > k. \end{cases}$$