

Resumen de programación 3

Tema 3. Notación asintótica.

Índice:

3.1. Introducción	3
3.2. Una notación para “el orden de”	3
3.3. Otra notación asintótica.....	7

Bibliografía:

Se ha tomado apuntes del libro:

- *Fundamentos de algoritmia*. G. Brassard y P. Bratley

Este tema al igual que ha pasado con los anteriores resumirlo cuesta, ya que en muchas ocasiones se añaden extras para completar el tema que cuesta integrarlos. Por ello, habrá algunas cosas que no queden del todo claro, para ello se debería mirar en el libro, aunque de nuevo se ha tratado de poner los conceptos más importantes.

3.1. Introducción.

Recuérdese que deseamos determinar matemáticamente la **cantidad de recursos** que necesita el algoritmo en función del tamaño (o a veces del valor) de los casos considerados. Dado que no existe una computadora estándar con la cual se puedan comparar las medidas de tiempo de ejecución, vimos también en el apartado 2.3 que nos contentaremos con expresar el tiempo requerido por el algoritmo salvo una constante multiplicativa (denominada constante oculta).

Esta notación se denomina **asintótica** porque trata acerca del comportamiento de funciones en el límite, esto es, para valores suficientemente grandes de su parámetro.

3.2. Una notación para “el orden de”

Tendremos una función arbitraria de los números naturales en los reales no negativos, tal como $t(n) = 27 * n^2 + 355/133 * n + 12$. Se puede pensar que n representa el tamaño del ejemplar sobre el cual es preciso que se aplique un algoritmo dado y en $t(n)$ como representante de la cantidad de un recurso dado que se invierte en ese ejemplar por una representación particular de este algoritmo.

Podría ser que la implementación invirtiera $t(n)$ ms. o quizá $t(n)$ represente la cantidad de espacio. Tal como se ha visto, la función $t(n)$ puede muy bien depender de la implementación más que únicamente del algoritmo. Recuerde el **principio de invarianza** que dice que la razón de los tiempos de ejecución de dos implementaciones diferentes del mismo algoritmo siempre está acotada por encima y por debajo mediante constantes predeterminadas.

Consideremos una función $f: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$ como $f(n) = n^2$. Diremos que $t(n)$ está en **el orden de** $f(n)$ si $t(n)$ está acotada superiormente por un múltiplo real positivo de $f(n)$ para todo n suficientemente grande. Matemáticamente, esto significa que existe una constante real y positiva c y un umbral n_0 , tal que $t(n) \leq c * f(n)$, siempre que $n \geq n_0$.

Por ejemplo, considérese las funciones $f(n)$ y $t(n)$ definidas anteriormente. Está claro que tanto $n \leq n^2$ como $1 \leq n^2$, siempre que $n \geq 1$. Por tanto, siempre y cuando $n \geq 1$ tendremos:

$$t(n) = 27 * n^2 + \frac{355}{133} * n + 12 \geq 27 * n^2 + \frac{355}{133} * n^2 + 12 * n^2 = 42 * \frac{16}{113} * n^2 = 42 * \frac{16}{113} * f(n).$$

Tomando $c = 42 * 16/113$ y $n_0 = 1$. Concluiremos que $t(n)$ es del orden de $f(n)$ por cuanto $t(n) \leq c * f(n)$, siempre que $n \geq n_0$.

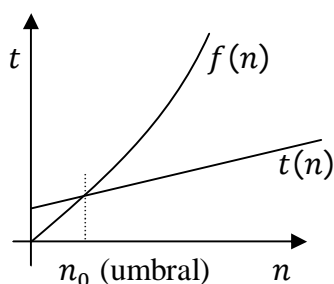
De esta manera, si la implementación de un algoritmo requiere en el **caso peor** un tiempo de $27 * n^2 + 355/133 * n + 12$ ms. Para resolver un caso de tamaño n , podríamos simplificar diciendo que el tiempo está *en el orden de* n^2 . Hay algo más importante: *el orden de* no solamente indica el tiempo requerido por una implementación particular del algoritmo, sino también (por el principio de invarianza) el requerido por **cualquier implementación**.

Por tanto, tenemos derecho a afirmar que el algoritmo en sí requiere un tiempo que está en el orden de n^2 o que requiere un tiempo cuadrado (independiente de la implementación).

Es conveniente disponer de un símbolo matemático para representar *el orden de*. Sea $f: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$ una función arbitraria de los números naturales en los reales no negativos. Le indicará mediante $O(f(n))$ el conjunto de todas las funciones $t: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$ tales que $t(n) \leq c * f(n)$, para todo $n \geq n_0$ para una constante positiva c y un umbral entero n_0 . En otras palabras:

$$O(f(n)) \equiv \{t: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0} \mid \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N}, \forall n \geq n_0 \mid t(n) \leq c * f(n)\}$$

Gráficamente sería:



siendo:

n_0 : Cierta umbral del tamaño del problema.

$f(n)$: Acota superiormente a la función $t(n)$.

Deduciendo del grafico anterior, podremos decir que habrá un cierto umbral que permitirá acotar superiormente el problema.

Ejemplo: Para representar que n^2 es del orden de n^3 lo haremos así:

$$n^2 \in O(n^3)$$

Como en teoría de conjuntos, se usa el símbolo \in . El umbral de n_0 suele resultar útil para simplificar argumentos, pero nunca es necesario para funciones estrictamente positivas. Sean $f, t: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$ dos funciones de los números naturales en los reales estrictamente positivos. La **regla del umbral** afirma que $t(n) \in O(f(n))$ si y sólo si existe una constante real positiva, tal que $t(n) \in O(f(n))$ para *cada* número natural n .

Una regla útil para demostrar que una función es del orden de otra es la regla del máximo. Sea $f, g: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$ dos funciones arbitrarias de los números naturales en los reales no negativos, La **regla del máximo** dice que $O(f(n), g(n)) = O(\max(f(n), g(n)))$.

Más específicamente, sean $p, q: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$ definidas para todo número natural n mediante $p(n) = f(n) + g(n)$ y $q(n) = \max(f(n), g(n))$ y consideremos una función arbitraria $t: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$, la regla del máximo dice que $t(n) \in O(f(n))$ si y sólo si $(\Leftrightarrow) t(n) \in O(g(n))$.

Ejemplo: Consideremos un algoritmo que se realiza en tres pasos: Inicialización, procesamiento y finalización. Supongamos que estos pasos requieren un tiempo de $O(n^2)$, $O(n^3)$ y $O(n * \log(n))$, respectivamente. Queda claro que el algoritmo completo requiere un tiempo $O(n^2 + n^3 + n * \log(n))$. Por la regla del máximo tendríamos:

$$O(n^2 + n^3 + n * \log(n)) = O(\max(n^2, n^3, n * \log(n))) = O(n^3)$$

Aún cuando el tiempo requerido por el algoritmo sea lógicamente la suma de los tiempos requeridos por sus partes separadas, veremos que lo determinará la parte que más consuma, siempre y cuando el número de partes sean constantes, independientemente del tamaño de la entrada.

Demostración: Demostraremos la regla del máximo para el caso de dos funciones. Observamos que:

$$f(n) + g(n) = \min(f(n), g(n)) + \max(f(n), g(n))$$

y

$$0 \leq \min(f(n), g(n)) \leq \max(f(n), g(n))$$

Se sigue entonces que:

$$\max(f(n), g(n)) \leq f(n) + g(n) \leq 2 * \max(f(n), g(n))$$

De lo que podemos deducir que:

- La cota inferior indica que una de las dos funciones será máxima con respecto a la otra. En este caso, sería la función máxima $(f(n) \text{ o } g(n))$.
- La cota superior es cuando las dos funciones son máximas.

Considérese ahora cualquier $t(n) \in O(f(n) + g(n))$. Sea c una constante adecuada tal que $t(n) \leq c * (f(n) + g(n))$ para todo n suficientemente grande. Se sigue que $t(n) \leq 2c * \max(f(n), g(n))$. Por tanto, $t(n)$ está **acotado superiormente** por un múltiplo real y positivo ($2c$) de $\max(f(n), g(n))$, para todo n suficientemente grande, lo cual demuestra que $t(n) \in \max(f(n), g(n))$.

Mostraremos varios ejemplos de **uso incorrecto** de la regla del máximo:

1. El siguiente razonamiento es erróneo:

$$O(n) = O(n + n^2 - n^2) = O(\max(n, n^2, n^2)) = O(n^2)$$

Hemos usado la regla del máximo con alguna de las funciones cuando son negativas con infinita frecuencia.

2. Tenemos esta función $t(n) = 12n^3 * \log(n) - 5n^2 + \log^2(n) + 36$

Razonando incorrectamente tendríamos:

$$O(t(n)) = O(\max(12n^3 * \log(n), -5n^2, \log^2(n) + 36)) = O(n^3 * \log(n)).$$

No usamos esta regla correctamente por ser $-5n^2$ negativo. El razonamiento correcto sería:

$$\begin{aligned} O(t(n)) &= O(12n^3 * \log(n) + n^3 * \log(n) - 5n^2 + \log^2(n) + 36) \\ &= O(\max(12n^3 * \log(n), n^3 * \log(n), -5n^2 + \log^2(n) + 36)) \\ &= O(n^3 * \log(n)). \end{aligned}$$

Por último, falta por decir que no especificamos la base del algoritmo dentro de la notación asintótica, porque podremos emplear las operaciones de los logaritmos para cambiar la base sin que afecte al tiempo.

Propiedades de el orden de:

- Reflexiva: $f(n) \in O(f(n))$ para toda función $f: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$.

Por ejemplo: $f(n) \leq 2 * f(n)$

- Transitiva: $f(n) \in O(g(n))$ y $g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$.

La demostración será:

$$\exists c \in \mathbb{R}^+, n_0 \in \mathbb{N} \forall n > n_0, f(n) = c * g(n) \quad \text{Por definición.}$$

$$\exists d \in \mathbb{R}^+, n_0' \in \mathbb{N} \forall n > n_0', g(n) = d * h(n) \quad \text{Por definición.}$$

$$f(n) \leq c * g(n) \leq c * d * h(n) \quad (f(n) \in O(h(n)))$$

El primer \leq se cumple por la expresión $n \geq n_0$ y el segundo por la siguiente expresión $n \geq \max(n, n_0')$.

Deducimos que existirá, por tanto, un umbral tal que quede acotado el valor.

Para demostrar que una función dada no pertenece al orden de otra función $f(n)$ tendremos estas formas:

- Demostración por contradicción: Es la forma más sencilla. Consiste en demostrar la veracidad de una sentencia demostrando que su negación da lugar a una contradicción.
- La regla del umbral generalizado: Implica la existencia de una constante real y positiva c tal que $t(n) \leq c * f(n)$ para todos los $n \geq 1$ (tomaremos n_0 como 1, nos interesa más la definición dada por la regla del umbral sin generalizar).
- La regla del límite: Lo definiremos completamente tras analizar la cota superior y el coste exacto.

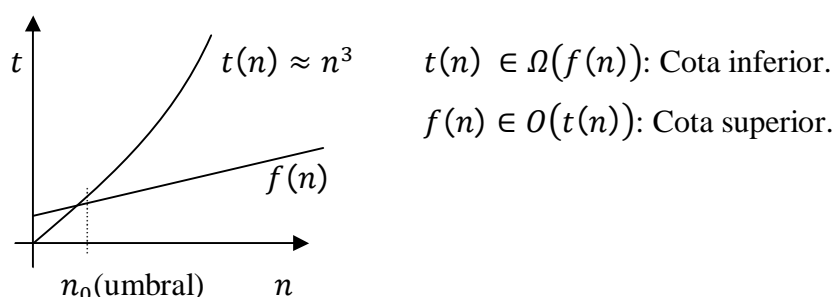
3.3. Otra notación asintótica.

La notación Omega: Necesitamos tener una notación dual para cotas inferiores. Esto es la notación Ω . Considérese una vez más dos funciones $t, f: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$ de los números naturales en los números reales no negativos. Diremos que $t(n)$ está en Omega (Ω) de $f(n)$, lo cual se denota como $t(n) \in \Omega(f(n))$, si $t(n)$ está acotada inferiormente por un múltiplo real positivo de $f(n)$ para todo n suficientemente grande.

Matemáticamente, esto significa que existe una constante real positiva d y un umbral entero n_0 tal que $t(n) \geq d * f(n)$ siempre que $n \geq n_0$.

$$\Omega(f(n)) \equiv \{t: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0} | \exists d \in \mathbb{R}^+, n_0 \in \mathbb{N}, \forall n \geq n_0 | t(n) \geq d * f(n)\}$$

Gráficamente sería:



La **regla de la dualidad** se definirá así:

$$t(n) \in \Omega(f(n)) \Leftrightarrow f(n) \in O(t(n))$$

Demostramos la implicación de la derecha (\Rightarrow)

$$\Rightarrow \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N}, \forall n \in n_0. t(n) \geq c * f(n) \Rightarrow \frac{1}{c} * t(n) \geq f(n) \Rightarrow f(n) \leq \frac{1}{c} * t(n).$$

Por la definición, deducimos que $f(n) \in O(t(n))$.

La notación Theta: Diremos que $t(n)$ está en Theta de $f(n)$, o lo que es igual que $t(n)$ está en el orden exacto de $f(n)$ y lo denotamos $t(n) \in \theta(f(n))$, si $t(n)$ pertenece tanto a $O(f(n))$ como a $\Omega(f(n))$.

La definición formal de θ es:

$$\theta(f(n)) = O(f(n)) \cap \Omega(f(n)).$$

Por tanto,

$$\theta(f(n)) \equiv \{t: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0} \mid \exists a, b \in \mathbb{R}^+, n_0 \in \mathbb{N}, \forall n \geq n_0 \mid a * f(n) \leq t(n) \leq b * f(n)\}.$$

Decimos que el conjunto del orden exacto está acotado tanto inferior como superiormente por $f(n)$. Podemos probarlo tanto por la definición como por la regla del límite, aunque preferiremos en los ejercicios hacerlo por este segundo método.

La regla del límite: Nos permite comparar dos funciones en cuanto a la notación asintótica se refiere. Tendremos que calcular el siguiente límite:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}.$$

Al resolver el límite se nos darán 3 posibles resultados:

$$1. \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c \in \mathbb{R} \Rightarrow \Rightarrow \left\{ \begin{array}{lll} f(n) \in O(g(n)) & f(n) \in \Omega(g(n)) & f(n) \in \theta(g(n)) \\ g(n) \in O(f(n)) & g(n) \in \Omega(f(n)) & g(n) \in \theta(f(n)) \end{array} \right\}.$$

Estas funciones se comportan igual, diferenciándose en una constante multiplicativa.

$$2. \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty \Rightarrow \Rightarrow \left\{ \begin{array}{lll} f(n) \notin O(g(n)) & f(n) \in \Omega(g(n)) & f(n) \notin \theta(g(n)) \\ g(n) \in O(f(n)) & g(n) \notin \Omega(f(n)) & g(n) \notin \theta(f(n)) \end{array} \right\}.$$

Por muy alta que sea la constante multiplicativa de $g(n)$ nunca superará a $f(n)$.

$$3. \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \Rightarrow \Rightarrow \left\{ \begin{array}{lll} f(n) \in O(g(n)) & f(n) \notin \Omega(g(n)) & f(n) \notin \theta(g(n)) \\ g(n) \notin O(f(n)) & g(n) \in \Omega(f(n)) & g(n) \notin \theta(f(n)) \end{array} \right\}.$$

$g(n)$ crece más exponencialmente que $f(n)$, por lo que sería su cota superior.