Sección III.1: Sobre la notación $\mathcal{O}$ y o

En esta pequeña nota se introducen los conceptos de la Notación de Landau, que dan origen a las llamadas "o minúscula" y "O mayúscula". Estas se usan en la comparación asintótica de funciones y son fundamentalas para la definición de la cota inferior asintótica, la cota superior asintótica y la cota ajustada asintótica.

Las definiciones originales consideran que tanto $f(x)$ como $g(x)$ son dos funciones definidas en un subconjunto de los complejos. Nosotros adoptaremos una notación más simple, pues diremos que estas funciones están bien definidas en un subconjunto de $\mathbb{R}$ al que pertenece un x_o fijo. De hecho, en general adoptaremos la convicción de que x_o es el origen o tiende al infinito. Así es usual escribir la **O-mayúscula** por

(1) $f(x) = \mathcal{O}(g(x))$ cuando $x \to x_o$
$$\iff \quad \exists M,\, \delta \in \mathbb{R}^+ \text{ tal que } |f(x)| \leq M|g(x)| \text{ para } |x - x_o| < \delta.$$

(2) $f(x) = \mathcal{O}(g(x))$ cuando $x \to \infty$
$$\iff \quad \exists M,\, y \in \mathbb{R}^+ \text{ tal que } |f(x)| \leq M|g(x)| \text{ para } \forall x \geq y.$$

En ambos casos se lee como "f es **O-mayúscula** de g" cuando x tiene a x_o. Una versión alternativa de (1) cuando $g(x_o) \neq 0$ es

$$\limsup_{x \to x_o} \left| \frac{f(x)}{g(x)} \right| \; < \; \infty.$$

En el caso de (2) notamos que entrelíneas se dice que $f(x)$ tiene a lo mucho un crescimiento limitado por $g(x)$. Los usos clásicos para una función son $T(n) = \mathcal{O}(n^k)$ o $T(n) \in \mathcal{O}(n^k)$ sin decir realmente el "cuando $x \to x_o$" que se entiende bajo contexto pues típicamente se toma $x_o = 0$ o x_o como $\pm\infty$. En los casos recién expuestos no es claro, pero cuando escribimos

$$e^x \; = \; 1 + x + \frac{x^2}{2} + \mathcal{O}(x^3),$$

queda subentendido que es cuando $x \to 0$.

De la definición salen directamente algunas propiedades sobre el producto y la suma como pueden ser las siguientes:

(a) $\mathcal{O}(x^n) + \mathcal{O}(x^m) \; = \; \mathcal{O}(x^\ell)$, con $\ell = \text{máx}\{n,\, m\}$.
(b) $\mathcal{O}(x^n) \cdot \mathcal{O}(x^m) \; = \; \mathcal{O}(x^{n+m})$
(c) Si $a \neq 0$ entonces $\mathcal{O}(ax^n) \; = \; \mathcal{O}(x^n)$, o en general, $\mathcal{O}(ag(x)) \; = \; \mathcal{O}(g(x))$
(d) Si $f_1 = \mathcal{O}(g_1)$ y $g_1 = \mathcal{O}(g_2)$, entonces $f_1 = \mathcal{O}(g_2)$.
(e) Si $f_1 = \mathcal{O}(g_1)$ y $f_2 = \mathcal{O}(g_2)$, entonces $f_1 f_2 = \mathcal{O}(g_1 g_2)$.
(f) $f_2\mathcal{O}(g_1) = \mathcal{O}(f_2 g_1)$ (aquí es igualdad entre conjuntos).
(g) Si $f_1 = \mathcal{O}(g_1)$ y $f_2 = \mathcal{O}(g_2)$, entonces $f_1 + f_2 = \mathcal{O}(|g_1| + |g_2|)$.

Ahora no debemos confundir lo dicho anteriormente con el concepto de "o minúscula". La relación $f(x) \in o(g(x))$ se lee como "f es **o-minúscula** de g" y significa que $g(x)$ crece mucho más rápido que $f(x)$.

Rigurosamente: $f(x) = o(g(x))$ cuando $x \to \infty$
$$\iff \quad \forall \epsilon > 0,\, \exists y \in \mathbb{R}^+ \text{ tal que } |f(x)| \leq \epsilon |g(x)| \text{ para } \forall x \geq y.$$

Es decir que en la O mayúscula tenemos la existencia de por lo menos un $M \in \mathbb{R}^+$, aquí es para todo $\epsilon > 0$. Es más severa la o minúscula de que la O mayúscula, pero el nombre proviene de que necesitamos que la relación se cumpla inclusive para cualquier ϵ pequeño.

Análogamente a la condición cuando $g(x) \nrightarrow 0$ cuando $x \to \infty$, tenemos que $f(x) \in o(g(x))$ si y sólo si

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} \;=\; 0.$$

(No basta que sea limitado.)

En este caso, tenemos ejemplos semejantes sobre la suma y el producto, pero debemos notar que a diferencia de lo que se espera para O mayúscula, hay inclusiones que no existen para o minúscula:

(a) $2x \in o(x^2)$, $\quad 2x \in \mathcal{O}(x^2)$.
(b) $2x^2 \notin o(x^2)$, $\quad 2x^2 \in \mathcal{O}(x^2)$.
(c) $1/x \in o(1)$, $\quad 1/x \in \mathcal{O}(1)$.

Pues en general tenemos que $o(g(x)) \in \mathcal{O}(g(x))$, pero no al contrario. Es claro que $x^n \in \mathcal{O}(x^n)$ pero $x^n \notin o(x^n)$.

Hemos dicho que los conceptos de o y $\mathcal{O}$ dan origen a las cotas asintóticas. Veamos que podemos definir un conjunto para $\mathcal{O}(g(x))$ por ejemplo. Así $g(x)$ es una **cota superior asintótica** (es decir, cuando $x \to \infty$) para todas las funciones en el conjunto:

$$\mathcal{O}(g(x)) \;=\; \Big\{ f(x) \;\Big|\; \exists M,\, y \in \mathbb{R}^+ \text{ tal que } |f(x)| \le M|g(x)|,\, \forall x \ge y \Big\}.$$

Por otro lado, $g(x)$ es una **cota inferior asintótica** para todas las funciones en el conjunto:

$$\Omega(g(x)) \;=\; \Big\{ f(x) \;\Big|\; \exists M,\, y \in \mathbb{R}^+ \text{ tal que } M|g(x)| \le |f(x),\, \forall x \ge y \Big\}.$$

Finalmente, la **cota asintótica ajustada** es cuando se representan las anteriores juntas, es decir, $\Theta(g(x)) = \mathcal{O}(g(x)) \cap \Omega(g(x))$. Se pueden crear los mismo conceptos con $o(g(x))$, $\omega(g(x))$ y $\theta(g(x))$ respectivamente.

Sección VII.2: Complejidad de un algoritmo

La idea de hacer un algoritmo más eficiente o rápido es una cuestión importante, en general no tenemos el deseo de aguardar a la máquina realizar un cálculo muy demorado. A veces parece que si un algoritmo tarda 7 nanosegundos y el otro solamente 3 nanosegundos, no hay una diferencia. Esto no es bien así, en general una rutina puede ser realizada millares de veces y así los ejemplos anteriores deben ser considerados.

Empero, unos pocos nanosegundos no son la causa real del análisis del desempeño y sí cómo el tiempo de los algoritmos se incrementa cuando el problema crece en tamaño. No es lo mismo que el tiempo se multiplique por una expresión de $\mathcal{O}(n)$ que por una $\mathcal{O}(n^2)$; el orden cuadrático aunque sea más rápido para pocos elementos n, cuando comienza a crecer será mucho mayor que el lineal.

Así necesitamos contar el número de operaciones que hay en un algoritmo dado. Tomemos como ejemplo el pseudo-código (realmente en código MATLAB) para la *Substitución para adelante (por líneas) de* $Ly = b$, la cual es la manera de resolver el problema $Ly = b$ para una matriz triangular inferior

$$L = \begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix}.$$

El algoritmo lleva el criterio de que una vez calculado un y_i, el equivalente b_i no es más usado, pues

$$y_1 = \frac{b_1}{l_{11}}, \quad y_2 = \frac{b_2 - l_{21}y_1}{l_{22}}, \quad \cdots \quad y_n = \frac{b_n - \sum_{j=1}^{n-1} l_{nj}y_j}{l_{nn}},$$

que como se puede apreciar en y_n sólo se necesita b_n.

```
+--------------------------------------+
|      Substitución para adelante      |
|            ( por lineas )            |
+--------------------------------------+
| Entradas:   L - matriz triangular    |
|             b - vector llegada       |
| Salidas:    b - vector solución      |
|                 ( como y en Ly = b ) |
|       ---------- // ----------       |
| for i = 1:n                          |
|     for j = 1:i-1                    |
|         b(i) = b(i) - L(i, j)*b(j);  |
|     end                              |
|     b(i) = b(i)/L(i, i);             |
| end                                  |
+--------------------------------------+
```

Vemos que el número de ciclos se están realizando dentro de los dos ciclos `for`. Es claro que podemos cambiar éstos por sumas, si `op.`, `ops.` significa respectivamente una operación o varias, entonces, tenemos

$$\begin{aligned}\text{Total}_{\texttt{ops.}} &= \sum_{i=1}^{n}\left[\left(\sum_{j=1}^{i-1} 2\,\texttt{ops.}\right)+\texttt{op.}\right] = \sum_{i=1}^{n}\left[2\cdot(i-1)+1\right]\texttt{ops.}\\ &= \left[2\left(\frac{n(n-1)}{2}\right)+n\right]\texttt{ops.} = \left[n^2-n+n\right] = n^2\,\texttt{ops.}\end{aligned}$$

Es claro que este es un número exacto. Pero realmente sólo nos interesa saber que el orden del número de operaciones es $\mathcal{O}(n^2)$, por lo que podríamos haber realizado la siguiente cuenta:

$$\sum_{i=1}^{n}\sum_{j=1}^{i} 1\,\texttt{op.} = \sum_{i=1}^{n} i\,\texttt{ops.} = \frac{n(n+1)}{2}\texttt{ops.} \approx \frac{n^2}{2}\texttt{ops.} \in \mathcal{O}(n^2)\texttt{ops.}, \tag{1}$$

con la suma de las operaciones en el interior de los dos ciclos; la operación fuera del ciclo más interior no cuenta gran cosa.

Aunque la cuenta exacta en general no es importante, observa que necesitamos realizar $(2n^2-n)\,\texttt{ops.}$ para el cálculo $x = Cb$ con $C \in \mathbb{R}^{n\times n}$ cualquiera. Si el damos estructura a C, digamos es triangular, entonces resolver $Ly = b$ (como en el algoritmo arriba) requiere simplemente $n^2\,\texttt{ops.}$.[1]

Nota que en (1) el límite de la suma interior lo cambiamos de $i-1$ para i; esto, respecto al orden del número de operaciones no cambia en nada. Del mismo modo, en lugar de realizar sumas que resulta en fórmulas a veces complicadas, es mejor calcular integrales, pues

$$\text{Total}_{\texttt{ops.}} \approx \int_0^n\int_0^i 1\,\texttt{op.}\,dj\,di = \int_0^n i\,\texttt{ops.}\,di = \frac{n^2}{2}\texttt{ops.},$$

nos devuelve siempre el mismo orden de operaciones. (En analogía con integración numérica, tenemos el converso, ahora aproximamos una suma de Riemann por una curva continua.)

Un poco antes hemos comparado números exactos de operaciones que realiza una rutina. Este número es un indicador del tiempo que se utiliza en ejecutar un código, pero no hay una forma directa de obtener el tiempo exacto. Aunque las operaciones aritméticas son las que conllevan al mayor consumo de CPU, esto no lo es todo. Un ejemplo puede ser dado por dos algoritmos análogos que sean construidos a modo que uno accese las matrices por columnas y el otro las accese por renglones, dependiendo del lenguaje utilizado un algoritmo será más eficiente que otro. Para ser sinceros, este tipo de efectos sólo se nota para n realmente grande.

En resumen, podemos pensar que si un algoritmo realiza $2n^2\,\texttt{ops.}$ y otro sólo $n^2\,\texttt{ops.}$, no podemos realmente decir que el primero sea el doble de lento que el segundo. Ya si la constante en lugar de ser 2 es 10^5, pues entonces sí puede ser reflejado el comportamiento no detectado en el primer ejemplo.

[1] Esta es una de las razones de porqué la factorización LU es tan importante.