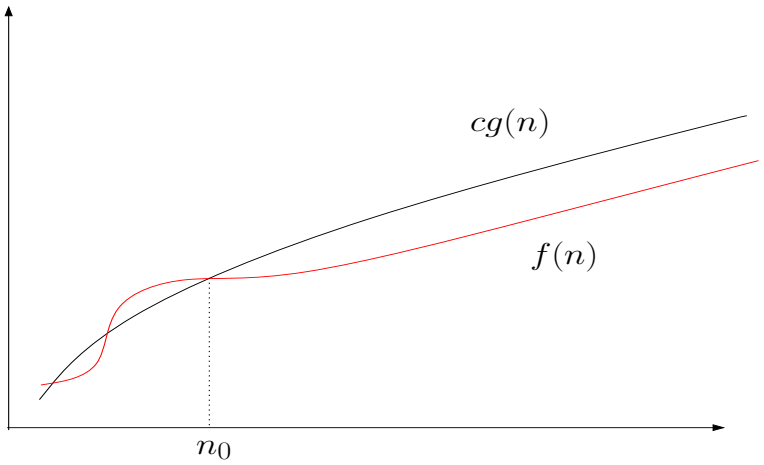
Progettazione di Algoritmi **Anno Accademico 2023–2024**

Lezione 3

Ugo Vaccaro

Date funzioni $f : n \in \mathbb{N} \to f(n) \in \mathbb{R}_+$, $g : n \in \mathbb{N} \to g(n) \in \mathbb{R}_+$, diremo che $f(n) = O(g(n))$ se e solo se esistono costanti $c > 0$ e $n_0 \in \mathbb{N}$ tale che $f(n) \leq cg(n)$, per ogni $n \geq n_0$.

Informalmente, $f(n) = O(g(n))$ se la funzione $f(n)$ non cresce più velocemente della funzione $g(n)$. Graficamente, abbiamo una situazione siffatta, ovvero il grafico della funzione $f(n)$, dal punto n_0 in poi, si trova sempre sotto il grafico della funzione $cg(n)$, per qualche costante $c > 0$.



Vediamo un esempio. Sia $f(n) = 2n^2 + 3n + 6$, $g(n) = n^2$, e tentiamo di mostrare che $2n^2 + 3n + 6 = O(n^2)$. Occorrerà quindi provare che esistono costanti $c > 0$ e $n_0 \in \mathbb{N}$ per cui $2n^2 + 3n + 6 \leq cn^2$, per ogni $n \geq n_0$. A tal fine osserviamo che $2n^2 + 3n + 6 \leq 2n^2 + 3n^2 + 6 \leq 2n^2 + 3n^2 + n^2 = 6n^2$, per ogni valore di n per cui $n^2 \geq 6$, ovvero per ogni valore di $n \geq 3$. Quindi, le costanti c e n_0 che cercavamo per provare che $2n^2 + 3n + 6 \leq cn^2$, per ogni $n \geq n_0$, possono essere scelte come $c = 6$ e $n_0 = 3$. Ovviamente, possono esistere anche altri valori di c e n_0 per cui la diseguaglianza $2n^2 + 3n + 6 \leq cn^2$, per ogni $n \geq n_0$, sia soddisfatta. Tuttavia, al fine di provare che $2n^2 + 3n + 6 = O(n^2)$ ci basta aver provato che $c = 6$ e $n_0 = 3$ vanno bene.

L'esempio appena visto ammette una semplice generalizzazione. Ovvero, possiamo provare che per ogni k costante vale che

$$a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 = O(n^k). \tag{1}$$

Quindi, ad esempio, la (1) ci dice che $7n^{10} + 8n^5 + 5 = O(n^{10})$. Cosa vuol dire ciò *in pratica*? Vuol dire che la velocità di crescita della funzione $7n^{10} + 8n^5 + 5$ non è maggiore di quella della più semplice funzione n^{10}, infatti se consideriamo il rapporto tra le due funzioni $7n^{10} + 8n^5 + 5$ ed n^{10}, otteniamo

$$\frac{7n^{10} + 8n^5 + 5}{n^{10}} = 7 + \frac{8}{n^5} + \frac{5}{n^{10}},$$

ed entrambi i termini $8/n^5$ e $5/n^{10}$ vanno a zero al crescere di n, ovvero i termini $8n^5$ e 5 sono entrambi trascurabili, rispetto a n^{10}, quando n è molto grande.

Si inizia qui a comprendere la utilità della notazione asintotica, che permette in maniera compatta di esprimere come una funzione cresce, al crescere del suo argomento.

Proviamo ora la (1).

$$\begin{aligned} a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 &\leq |a_k| n^k + |a_{k-1}| n^{k-1} + \ldots + |a_1| n + |a_0| \\ &\leq |a_k| n^k + |a_{k-1}| n^k + \ldots + |a_1| n^k + |a_0| n^k \\ &\leq (|a_k| + |a_{k-1}| + \ldots + |a_1| + |a_0|) n^k = c n^k. \end{aligned}$$

L'essenza della formula (1) consiste nel fatto che, se vogliamo descrivere la velocità di crescita di un polinomio, basterà concentrarci solo sul suo termine di grado massimo.

Per esercizio, analizziamo la complessità del seguente semplice algoritmo, che prende in input una sequenza $a = a[0]a[1]\ldots a[n-1]$ di n numeri.

```
Algoritmo(a)
x = 0; y = 0
FOR(i = 0; i < n; i = i + 1){
    FOR(j = i; j < n; j = j + 1){
        x = x + a[j]
    }
   y = y + i
}
RETURN x + y
```

Quando $i = 0$, l'algoritmo esegue l'istruzione $x = x + a[j]$ all'interno del secondo `FOR` un numero di volte pari ad n, quando $i = 1$, l'algoritmo esegue l'istruzione $x = x + a[j]$ un numero di volte pari ad $n - 1$, quando $i = 2$ l'algoritmo esegue l'istruzione $x = x + a[j]$ un numero di volte pari ad $n - 2$ e così via. In totale, il numero di operazioni effettuate sarà quindi

$$n + (n-1) + (n-2) + \ldots + 2 + 1 = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}. \tag{2}$$

Proviamo l'ultima uguaglianza nella (2). É più facile calcolare $2\sum_{i=1}^{n} i$. Infatti, essa sarà

$$\begin{aligned} 2\sum_{i=1}^{n} i =& 1 + \quad 2 + \quad 3 + \ldots \quad + \quad n + \\ & n + (n-1) + (n-2) + \ldots + 2 + 1 \\ =& (n+1)n \end{aligned}$$

da cui la (2).

Il numero di operazioni $y = y + i$ al di fuori del secondo `FOR` dell'algoritmo `Algoritmo`(a) è pari ad n, per cui il numero totale $T(n)$ di operazioni elementari che l'algoritmo esegue è pari a $n(n+1)/2 + n$, per cui, applicando la definizione di O, possiamo dire che $T(n) = O(n^2)$.

Ricordiamo che $\forall n$ il valore $x = \log_2 n$ è tale che $2^x = n$. Proviamo che $\log n = O(n)$, ovverosia proviamo che esiste $c > 0$, $n_0 \geq 0$ tale che $\log n \leq cn$, per ogni $n \geq n_0$. Lo proveremo per induzione su n, con costanti $c = 1 = n_0$. Per $n = 1$ abbiamo che $\log 1 = 0 \leq 1$. Supposto vero $\log n \leq n$, proviamo che $\log(n+1) \leq n+1$. Avremo

$$\begin{aligned}\log(n+1) &\leq \log(n+n)\\ &= \log(2n)\\ &= \log 2 + \log n\\ &\leq 1 + n \quad \text{(dall'ipotesi induttiva)}\end{aligned}$$

Osserviamo inoltre che, poichè vale $\log n^k = k \log n$, per ogni costante k abbiamo anche che $\log n^k = O(n)$.

A mò di esempio, possiamo quindi osservare che $7n + 8\log n \leq 7n + 8n = O(n)$, $7n + 8\log n^3 \leq 7n + 24n = O(n)$.

Analizziamo la complessità del seguente algoritmo.

```
Algoritmo(n)
x = 0; y = 1
WHILE(y < n + 1){
    x = x + 1
    y = 2 × y
}
RETURN x
```

Osserviamo che dopo la prima iterazione del ciclo `WHILE` vale che $y = 2^1$, dopo la seconda iterazione vale che $y = 2^2$, dopo la terza iterazione vale che $y = 2^3, \ldots$, e così via. Pertanto, la iterazione i-esima in cui terminiamo sarà tale che $2^i \leq n < 2^{i+1}$, ovvero $i \leq \log n$. Poichè in ogni iterazione del ciclo `WHILE` eseguiamo un numero costante di operazioni, ne segue che la complessità $T(n)$ dell'algoritmo sarà $T(n) = O(\log n)$.

Abbiamo visto che $\log n \leq n$. Cosa possiamo dire su $\log n$ e $\sqrt{n}$? Ovviamente, vale che $\log \sqrt{n} \leq \log n$, da cui otteniamo che $(1/2)\log n = \log n^{1/2} = \log \sqrt{n} \leq \sqrt{n}$ (in quanto sappiamo che $\log x \leq x$), ovvero $\log n \leq 2\sqrt{n}$, per cui $\log n = O(\sqrt{n})$.

É utile ricordarsi le seguenti relazioni generali tra le più comuni funzioni. Vale che

$$\log n = O(\sqrt{n}),\ \sqrt{n} = O(n),\ n^k = O(n^{k+1}),\ n^k = O(2^n),\ 2^n = O(n!),\ n! = O(n^n).$$

Proviamo quelle non ancora giustificate. Iniziamo con il provare che $n^k = O(2^n)$, per ogni costante k. Data quindi la costante k, scegliamo n maggiore o uguale al più piccolo valore (sia esso n_0) per cui vale che $n/(\log n) \geq k$. Un tale valore esiste sicuramente in quanto il rapporto $n/(\log n)$ è crescente in n. Ne segue che, per ogni $n \geq n_0$ vale che

$$n^k \leq n^{n/(\log n)} = (2^{\log n})^{n/(\log n)} = 2^n,$$

il che prova che $n^k = O(2^n)$.

Per provare che $2^n = O(n!)$, osserviamo che

$$2^n = \underbrace{2 \times 2 \times \ldots \times 2}_{n \text{ volte}} \leq \underbrace{1 \times 2 \times 3 \times \ldots \times n}_{n \text{ volte}} = n!, \qquad \forall n > 3.$$

Infine, per provare che $n! = O(n^n)$ osserviamo che

$$n! = \underbrace{1 \times 2 \times 3 \times \ldots \times n}_{n \text{ volte}} \leq \underbrace{n \times n \times n \times \ldots \times n}_{n \text{ volte}} = n^n, \qquad \forall n \geq 1.$$

Ovviamente, le considerazioni di sopra valgono qualunque sia la base dell'esponenziale (ovvero per funzioni del tipo a^n, per $a > 1$, e qualunque sia la base del logaritmo, purchè anch'essa maggiore di 1.

Chiudiamo questa parte con un ultimo esempio. Proviamo che $n^3 + \sqrt{n}\log n + n^2 = O(n^3)$. Abbiamo che

$$n^3 + \sqrt{n}\log n + n^2 \leq n^3 + n\log n + n^2 \leq n^3 + n \cdot n + n^2 = n^3 + 2n^2 \leq 2n^3, \qquad \forall n \geq 2.$$

Proviamo ora che

$$\forall \text{ costanti } a > 0, b > 0, k > 0 \text{ vale } \log^a n^b = O(n^k). \tag{3}$$

Ad esempio, $\log^5 n^6 = O(\sqrt[3]{n})$, infatti basta infatti porre $a = 5, b = 6, k = 1/3$ nella (3).

Altro esempio: $\log^2 n^{10} = O(\sqrt[6]{n})$, basta porre $a = 2, b = 10, k = 1/6$ nella (3).

Occorre provare che per ogni $a, b, k > 0$

$$\exists c, n_0 \ : \ (\log n^b)^a \leq cn^k, \ \forall n \geq n_0 \qquad (3)$$

Proviamolo innanzitutto nel caso $a = 1$. Ricordiamo che $\log x^y = y \log x$, e che $\log x \leq dx$. Pertanto

$$(\log n^b) = (b \log n) = (b\frac{1}{k} \cdot k \log n) = (b\frac{1}{k} \cdot \log n^k) \leq b\frac{1}{k} d n^k,$$

provando la (3) per $a = 1$, (ma $\forall k$), con $c = (bd)/k$. Nel caso generale, usando la (3) con parametri $a = 1, b$ arbitrario, e k/a avremo

$$\log^a n^b = (\log n^b)^a \leq (cn^{(k/a)})^a = c^a n^k.$$

La seguente tabella è molto utile per risolvere esercizi. Essa và interpretata nel modo seguente: leggendo dall'alto in basso, una funzione $f(n)$ in "alto" è O di una qualsiasi funzione $g(n)$ più in "basso".

Espressione O	nome
$O(1)$	costante
$O(\log\log n)$	log log
$O(\log n)$	logaritmico
$O(\sqrt[c]{n}),\ c > 1$	sublineare
$O(n)$	lineare
$O(n \log n)$	$n \log n$
$O(n^2)$	quadratico
$O(n^3)$	cubico
$O(n^k)\ (k \geq 1)$	polinomiale
$O(a^n)\ (a > 1)$	esponenziale
$O(n!)$	fattoriale

Per cui, ad esempio

- $3\log^3 n^2 = O(n)$
- $7\log^2 n^3 = O(\sqrt[3]{n})$
- $6n\log^5 n^{10} = O(n^{5/4})$
- $\sqrt[2]{n} = O(n)$
- $\sqrt[2]{n}\log n = O(n) \implies \sqrt[2]{n} = O(n/\log n)$
- $n^{10} = O(2^n)$
- $n^{10} = O(2^n/n^7)$ (in quanto equivale a dire che $n^{10} \cdot n^7 = n^{17} = O(2^n)$)
- $7^n = O(n!)$

Le seguenti regole sono utili per confrontare le funzioni tra di loro.

$$d(n) = O(f(n)) \Rightarrow ad(n) = O(f(n)),\ \forall \text{ costante } a > 0$$

Ad esempio: $\log n = O(n) \Rightarrow 7\log n = O(n)$

$$d(n) = O(f(n)), e(n) = O(g(n)) \Rightarrow d(n) + e(n) = O(f(n) + g(n))$$

Ad esempio: $\log n = O(n), \sqrt{n} = O(n) \Rightarrow \log n + \sqrt{n} = O(n)$

$$d(n) = O(f(n)), e(n) = O(g(n)) \Rightarrow d(n)e(n) = O(f(n)g(n))$$

Ad esempio: $\log n = O(\sqrt{n}), \sqrt{n} = O(\sqrt{n}) \Rightarrow \sqrt{n}\log n = O(n)$

$$d(n) = O(f(n)), f(n) = O(g(n)) \Rightarrow d(n) = O(g(n))$$

Ad esempio: $\log n = O(\sqrt{n}), \sqrt{n} = O(n) \Rightarrow \log n = O(n)$

$$f(n) = a_d n^d + \cdots a_1 n + a_0 \Rightarrow f(n) = O(n^d)$$

Ad esempio: $5n^7 + 6n^4 + 3n^3 + 100 = O(n^7)$

$$n^x = O(a^n),\ \forall \text{ costanti } x > 0, a > 1$$

Ad esempio: $n^{100} = O(2^n)$