Algoritmi e Strutture Dati

Lezione 4 | 02 Marzo

Università degli Studi di Modena e Reggio Emilia — Unimore

1. Notazione Asintotica — a cosa serve?

Per confrontare algoritmi e misurare il loro costo abbiamo bisogno di un formalismo. Entra in scena la **notazione asintotica**.

Ci serve per tre cose:

1. **Stimare** il tempo e lo spazio necessari nel caso peggiore.
2. **Confrontare** le prestazioni di algoritmi diversi per lo stesso problema.
3. **Stabilire la difficoltà** di un problema (dalla difficoltà degli algoritmi che lo risolvono).

Man mano che l'istanza cresce (es. un array sempre più lungo), *come cresce* il tempo di esecuzione? Questa è la domanda chiave.

2. O-Grande (Big-O)

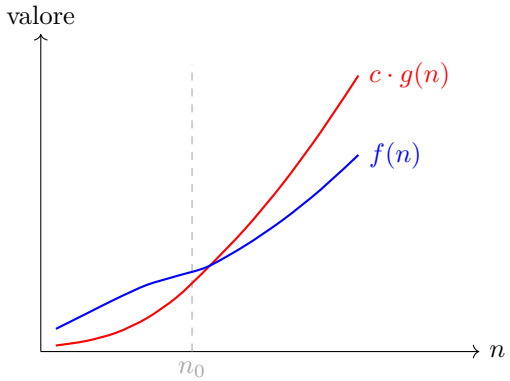
2.1. Definizione

Definizione

Date due funzioni $f(n)$ e $g(n)$, diremo che $f(n) \in O(g(n))$ **se e solo se** esistono due costanti $c > 0$ e $n_0 \in \mathbb{N}$ tali che:

$$f(n) \leq c \cdot g(n) \quad \forall n \geq n_0$$

Intuizione: a partire da $n = n_0$, la funzione $f(n)$ non sta *mai* sopra $c \cdot g(n)$. Quindi $f(n)$ è **limitata superiormente** da $g(n)$, a meno di una costante moltiplicativa.



Nota

Le costanti c e n_0 **non sono uniche** e non sono importanti in sé. L'importante è che *esistano*, non che siano le più piccole possibili. Con $c = 100$ e $n_0 = 1$ va ancora benissimo.

2.2. Esempio: $2n^2 + 3n + 6 \in O(n^2)$

Devo trovare $c > 0$ e $n_0 \in \mathbb{N}$ tali che $2n^2 + 3n + 6 \leq c \cdot n^2$ per ogni $n \geq n_0$.

La catena di disuguaglianze (per $n \geq 1$):

$$2n^2 + 3n + 6 \;\leq\; 2n^2 + 3n^2 + 6 \;\leq\; 2n^2 + 3n^2 + 6n^2 \;=\; 11n^2$$

Quindi con $c = 11$ e $n_0 = 1$ la disuguaglianza vale. ✓

2.3. Proposizione: i polinomi sono $O(n^k)$

Definizione

Per ogni polinomio di grado k a coefficienti reali:

$$a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0 \;\in\; O(n^k)$$

Dimostrazione (per $n > 0$):

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

Quindi $c = \sum_{i=0}^{k} |a_i|$ e $n_0 = 1$.

Esempio: $f(n) = 3n^4 + 2n^3 + 5n^2 + 7n + 1 \in O(n^4)$, con $c = 18$ e $n_0 = 1$.

2.4. Proposizione: $\log n \in O(n)$

Attenzione

Convenzione di questo corso: quando scriviamo $\log n$ intendiamo **sempre** $\log_2 n$, salvo indicazione contraria.

Con $c = 1$ e $n_0 = 1$, dimostriamo per **induzione** che $\log n \leq n$ per ogni $n \geq 1$:

Caso base ($n = 1$):

$$\log_2 1 = 0 \leq 1 \quad ✓$$

Passo induttivo: assumiamo $\log n \leq n$ (ipotesi induttiva), vogliamo $\log(n+1) \leq n + 1$.

$$\log(n+1) \leq \log(n+n) = \log(2n) = \log 2 + \log n = 1 + \log n \leq 1 + n = n + 1 \quad ✓$$

2.5. Tabella delle classi di complessità

Notazione	Nome
$O(1)$	Costante (sublineare)
$O(\log \log n)$	Logaritmo iterato
$O(\log n)$	Logaritmica
$O(\sqrt{n})$	Radice
$O(n)$	Lineare
$O(n \log n)$	Quasilineare
$O(n^2)$	Quadratica
$O(n^3)$	Cubica
$O(2^n)$	Esponenziale
$O(n!)$	Fattoriale

Polinomiale ci va ancora bene. Le ultime due — esponenziale e fattoriale — sono **veramente brutte**: crescono così velocemente che anche per istanze modeste i tempi diventano proibitivi.

Nota psicologica: anche solo sapere che esiste una soluzione polinomiale (anche se non lineare) è già un passo avanti, perché significa che non siamo nell'inferno dell'esponenziale.

3. Proprietà utili di O-Grande

Nota

La prof dice che non le chiede all'esame, ma sono un buon esercizio per capire come funziona la definizione.

Proprietà 1 — Costante moltiplicativa:

Se $f(n) \in O(g(n))$, allora $a \cdot f(n) \in O(g(n))$ per ogni costante $a > 0$.

Le costanti moltiplicative *non influenzano* la classe asintotica.

Dimostrazione: da $f(n) \leq c \cdot g(n)$ segue $a \cdot f(n) \leq (ac) \cdot g(n) = c' \cdot g(n)$.

Esempio: $\log n \in O(n) \Rightarrow 7 \log n \in O(n)$.

Proprietà 2 — Somma:

Se $d(n) \in O(f(n))$ e $e(n) \in O(g(n))$, allora:

$$d(n) + e(n) \ \in \ O(\max\{f(n), g(n)\})$$

Proprietà 3 — Prodotto e Transitività: vedere le slides.

4. Omega-Grande (Ω) — il limite inferiore

La notazione O dà il tetto ("al più cresce così"). Ma se non sono preciso, potrei dichiarare $O(n^2)$ per una funzione che è in realtà $O(n)$. Per dare il **limite inferiore** usiamo Ω:

Definizione

Date $f, g : \mathbb{N} \to \mathbb{R}^+$, diremo che $f(n) \in \Omega(g(n))$ **se e solo se** esistono due costanti $c > 0$ e $n_0 \in \mathbb{N}$ tali che:

$$f(n) \geq c \cdot g(n) \quad \forall n \geq n_0$$

Intuizione: $f(n)$ non si abbassa *mai* sotto $c \cdot g(n)$ per $n \geq n_0$. È il **pavimento** invece del soffitto.

Se ti dico "$f(n)$ cresce almeno come n^2", significa che può crescere come n^2, n^3, n^{100}... ma non meno. Per avere un'informazione davvero utile, ho bisogno anche di un tetto.

5. Theta-Grande (Θ) — il match preciso

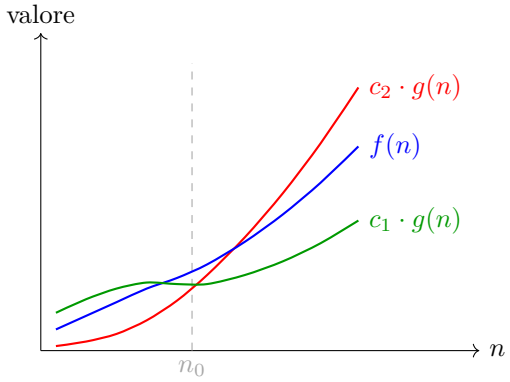
Se il tetto e il pavimento coincidono, abbiamo una stima **precisa**.

Definizione

Date $f, g : \mathbb{N} \to \mathbb{R}^+$, diremo che $f(n) \in \Theta(g(n))$ **se e solo se** esistono costanti $c_1 > 0$, $c_2 > 0$ e $n_0 \in \mathbb{N}$ tali che:

$$c_1 \cdot g(n) \;\leq\; f(n) \;\leq\; c_2 \cdot g(n) \quad \forall n \geq n_0$$

Equivalentemente: $f(n) \in \Theta(g(n)) \iff f(n) \in O(g(n))$ **e** $f(n) \in \Omega(g(n))$.



Se ti dico che cresce *al più* come una parabola e *almeno* come una parabola, allora sappiamo che **è esattamente una parabola** (a meno di costanti).

5.1. Esempio: $3n^4 + 2n^3 + 5n^2 + 7n + 1 \in \Theta(n^4)$

- **Lato O:** già dimostrato, $c_2 = 18$, $n_0 = 1$.
- **Lato Ω:** $3n^4 + \cdots \geq 3n^4$, quindi $c_1 = 3$, $n_0 = 1$. ✓

5.2. Esempio: $4n^2 + \log n \in \Theta(n^2)$

Lato O: poiché $\log n < n^2$ per $n > 1$:

$$4n^2 + \log n < 4n^2 + n^2 = 5n^2 \quad \Rightarrow \quad c_2 = 5$$

Lato Ω: poiché $\log n \geq 0$ per $n \geq 1$:

$$4n^2 + \log n \geq 4n^2 \quad \Rightarrow \quad c_1 = 4$$

Quindi $4n^2 + \log n \in \Theta(n^2)$. ✓

6. Costo Computazionale — contare le operazioni

Teoria bella. Ma come si calcola concretamente il costo di un algoritmo?

Cosa si conta: assegnamenti e operazioni logico-aritmetiche (le cosiddette *operazioni elementari*).

Goal: trovare $T(n)$ = numero massimo di operazioni in funzione di n, e poi trovare la $f(n)$ più piccola possibile tale che $T(n) \in O(f(n))$.

Nota

Trucchetto: se ho un intervallo intero da i a j (estremi inclusi), il numero di elementi è $j - i + 1$. Esempio: da 0 a $n-1$ ci sono $(n-1) - 0 + 1 = n$ elementi $\Rightarrow$ n iterazioni.

6.1. Esempio 1 — ciclo semplice

```
for i := 0 to n-1
    c := c + 1        (* 2 operazioni: assegnamento + somma *)
    d := d + 1        (* 2 operazioni *)
c := c + d            (* 2 operazioni, fuori dal ciclo *)
```

- Corpo del ciclo: 4 operazioni per iterazione.
- Il ciclo gira n volte $\Rightarrow$ $4n$ operazioni.
- Fuori dal ciclo: 2 operazioni.

$$T(n) = 4n + 2 \in O(n)$$

6.2. Esempio 2 — cicli annidati

```
for i := 0 to n-1
    for j := 0 to n-1
        x := x + 1     (* 2 operazioni *)
    y := y + 1          (* 2 operazioni *)
```

- Ciclo interno: $2n$ operazioni per ogni iterazione del ciclo esterno.
- Ciclo esterno: gira n volte, ogni volta esegue $2n + 2$ operazioni.

$$T(n) = n \cdot (2n + 2) = 2n^2 + 2n \in O(n^2)$$

(La prof lo calcola con le sommatorie, ma il risultato è lo stesso.)

6.3. Esempio 3 — ciclo while con raddoppio

```
x := 0
y := 1
while y < n
    x := x + 1
    y := 2*y
return x
```

Prima dell'inizio: $y = 1$. Ad ogni iterazione y si raddoppia: $1,\ 2,\ 4,\ 8,\ \ldots,\ 2^k$.

Il ciclo si ferma quando $y \geq n$, ovvero quando $2^k \geq n$, cioè $k \geq \log_2 n$.

$$T(n) \in O(\log n)$$

Nota

Se non è immediato capire quante volte gira un ciclo `while`, aiuta fissare un valore di n piccolo (es. $n = 5$) e contare manualmente le iterazioni, poi alzare n e cercare il

pattern.