Crescita funzioni

20 novembre 2006

1 Funzioni di costo

Definizione 1 (Funzione di costo)**.** *Utilizziamo il termine* funzione di costo *per indicare una funzione* $f : \mathbb{N} \to \mathbb{R}$ *dall'insieme dei numeri naturali ai reali.*

Come possiamo confrontare le funzioni di costo che abbiamo ottenuto finora?

- Insertion Sort, caso ottimo: $f(n) = a_1 n + b_1$
- Insertion Sort, caso pessimo-medio: $f(n) = a_2 n^2 + b_2 n + c_2$
- Merge Sort (sempre): $f(n) = a_1 n \log n + b_2 n + c_2$

Ci sono troppi dettagli: termini di vari ordini, costanti moltiplicative. In realtà: siamo interessati solo al "tasso di crescita" di queste funzioni ovvero, al loro comportamento asintotico.

Consideriamo queste coppie di funzioni di costo; quali sono preferibili?

- $\log_2 n$ oppure $\log_{10} n$
- $\log n^{1000}$ oppure $\log n$
- $\log^{1000} n$ oppure $n^{0.001}$
- $1000n$ oppure $0.001 \cdot n^2$
- n^{1000} oppure $0.001 \cdot 2^n$

Vediamo come crescono vari tipi di funzione al crescere di n:

$f(n)$	10	100	1000	10000	tipo
$\log n$	3	6	9	13	logaritmico
$\sqrt{n}$	3	10	31	100	sublineare (polinomiale)
n	10	100	1000	10000	lineare (polinomiale)
$n \log n$	30	664	9965	132877	loglineare (polinomiale)
n^2	10^2	10^4	10^6	10^8	quadratico (polinomiale)
n^3	10^3	10^6	10^9	10^{12}	cubico (polinomiale)
2^n	1024	10^{30}	10^{300}	10^{3000}	esponenziale

Figura 1: Esempi di valori di complessità

E' da notare come ci siano autentici "salti" di complessità da una funzione ad un'altra; inoltre, è possibile in qualche modo definire delle "grandi classi", con il comportamento logaritmico, polinomiale e esponenziale chiaramente separati.

2 Notazioni O, Ω, Θ

Siamo interessati a caratterizzare la crescita delle funzioni in base al loro comportamento *asintotico*, ovvero al crescere del valore di ingresso (dimensione del problema).

Definizione 2 (Notazione O)**.** *Sia $g(n)$ una funzione di costo; indichiamo con $O(g(n))$ l'insieme delle funzioni $f(n)$ tali per cui:*

$$\exists c > 0, n_0 \geq 0 : 0 \leq f(n) \leq cg(n), \forall n \geq n_0.$$

In altre parole:

- *asintoticamente la funzione giace sotto $cg(n)$;*
- *$f(n)$ cresce al più come $g(n)$;*
- *$g(n)$ è un limite* **asintotico** *superiore per $f(n)$.*

Definizione 3 (Notazione Ω)**.** *Sia $g(n)$ una funzione di costo; indichiamo con $\Omega(g(n))$ l'insieme delle funzioni $f(n)$ tali per cui:*

$$\exists c > 0, n_0 \geq 0 : 0 \leq cg(n) \leq f(n), \forall n \geq n_0.$$

In altre parole:

- *asintoticamente la funzione giace sopra $cg(n)$;*
- *$f(n)$ cresce almeno quanto $g(n)$;*
- *$g(n)$ è un limite* **asintotico** *inferiore per $f(n)$.*

Definizione 4 (Notazione Θ)**.** *Sia $g(n)$ una funzione di costo; indichiamo con $\Omega(g(n))$ l'insieme delle funzioni $f(n)$ tali che per cui:*

$$\exists c_1 > 0, c_2 > 0, n_0 \geq 0 : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0.$$

In altre parole $f(n) = \Theta(g(n))$ se e solo se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$.

Abusiamo delle notazioni appena introdotte scrivendo:

$$f(n) = O(g(n)), f(n) = \Omega(g(n)), f(n) = \Theta(g(n))$$

anche se $O(g(n))$ in realtà è un insieme e dovremmo scrivere

$$f(n) \in O(g(n)), f(n) \in \Omega(g(n)), f(n) \in \Theta(g(n))$$

Diciamo "$f(n)$ è un O grande, Ω, Θ di $g(n)$".

La figura 2 illustra graficamente il significato delle notazioni appena introdotte.

La notazione O a volte è confusa con la notazione Θ; un uso che deriva dalla letteratura passata sull'argomento. Infatti può essere utilizzata più semplicemente della notazione Θ per descrivere la complessità di un algoritmo: se utilizzata per descrivere il caso peggiore, limita il tempo massimo necessario per eseguire un algoritmo.

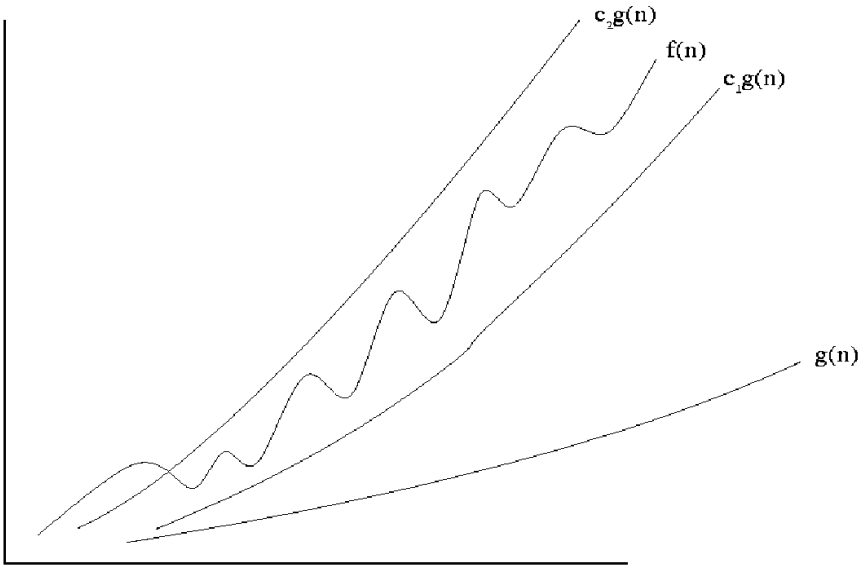

Figura 2: Comportamento asintotico della notazione O

3 Esercizi e proprietà

3.1 Vero o falso: $10n^3 + 2n^2 + 7 = O(n^3)$?

Si consideri la seguente funzione cubica: $f(n) = 10n^3 + 2n^2 + 7$. Vogliamo provare che $f(n) = O(n^3)$. Dobbiamo provare che:

$$\exists c > 0, n_0 \geq 0 : 10n^3 + 2n^2 + 7 \leq cn^3, \forall n \geq n_0$$

Si può procedere così:

$$\begin{aligned} f(n) &= 10n^3 + 2n^2 + 7 \\ &\leq 10n^3 + 2n^3 + 7 \\ &\leq 10n^3 + 2n^3 + n^3, \forall n \geq 2 \\ &= 13n^3, \forall n \geq 2 \\ &\leq cn^3 \end{aligned}$$

che è vera per $n_0 = 2$ e $c \geq 13$.

3.2 Vero o falso: $3n^2 + 7n = \Theta(n^2)$?

Si consideri la seguente funzione cubica: $f(n) = f(n) = 3n^2 + 7n$. Vogliamo provare che $f(n) = \Theta(n^2)$. Dobbiamo provare che:

$$\exists c_1 > 0, c_2 > 0, n_0 \geq 0 : c_1 n^2 \leq 3n^2 + 7n \leq c_2 n^2, \forall n \geq n_0$$

Si può procedere così:

$$\begin{aligned} f(n) &= 3n^2 + 7n \\ &\leq 3n^2 + 7n^2 \\ &= 10n^2 \\ &\leq c_2 n^2 \end{aligned}$$

che è vera per $n_0 = 0$ e $c_2 \geq 10$. Inoltre,

$$\begin{aligned} f(n) &= 3n^2 + 7n \\ &\geq 3n^2 \\ &\geq c_1 n^2 \end{aligned}$$

che è vera per $n_0 = 0$ e $c_1 \leq 3$.

3.3 Espressioni polinomiali

Abbiamo visto due espressioni polinomiali. Cerchiamo di trarne una regola generale. Vogliamo provare che un polinomio asintoticamente positivo di grado k cresce come n^k:

$$f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0, a_k > 0 \Leftrightarrow f(n) = \Theta(n^k)$$

Dobbiamo innanzitutto provare che $f(n) = O(n^k)$:

$$\begin{aligned} f(n) &= a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \\ &\leq a_k n^k + |a_{k-1}| n^{k-1} + \ldots + |a_1| n + |a_0| \\ &\leq a_k n^k + |a_{k-1}| n^k + \ldots + |a_1| n^k + |a_0| n^k \\ &= (a_k + |a_{k-1}| + \ldots + |a_1| + |a_0|) n^k \\ &\leq c_2 n^k \end{aligned}$$

L'ultima relazione è vera per $c_2 \geq (|a_k| + |a_{k-1}| + \ldots + |a_1| + |a_0|)$ e per $n_0 = 0$.

Dobbiamo poi provare che $f(n) = \Omega(n^k)$. E' facile notare che

$$a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \leq c n^k \Leftrightarrow c \leq a_k + \frac{a_{k-1}}{n} + \frac{a_{k-2}}{n^2} + \ldots + \frac{a_0}{n^k} > 0$$

In altre parole, se troviamo un valore di n per cui la parte centrale della disequazione è maggiore di 0, lo possiamo scegliere come valore di c.

$$\begin{aligned} a_k + \frac{a_{k-1}}{n} + \frac{a_{k-2}}{n^2} + \ldots + \frac{a_0}{n^k} &\geq \\ a_k - \frac{|a_{k-1}|}{n} - \frac{|a_{k-2}|}{n^2} - \ldots - \frac{|a_0|}{n^k} &\geq && \text{Mettiamo il segno negativo a tutti i coefficienti} \\ a_k - \frac{|a_{k-1}|}{n} - \frac{|a_{k-2}|}{n} - \ldots - \frac{|a_0|}{n} &> 0 && \text{Rimuoviamo l'esponente} \end{aligned}$$

L'ultima espressione è vera se

$$n > \frac{|a_{k-1}| + \ldots + |a_0|}{a_k}$$

3.4 Vero o falso: $6n^2 = O(n)$?

Dobbiamo dimostrare che:

$$\exists c > 0, n_0 \geq 0 : 6n^2 \leq cn, \forall n \geq n_0$$

Purtroppo abbiamo che:

$$6n^2 \leq cn \Leftrightarrow c \geq 6n$$

Questo significa che c dipende da n, ovvero che non esiste una "costantè' c.

3.5 Vero o falso: $6n^2 = \Omega(n^3)$?

Dobbiamo dimostrare che

$$\exists c > 0, \exists n_0 \geq 0 : cn^3 \leq 6n^2$$

che implica che $c \leq 6/n$; ovvero, qualsiasi sia il valore di c, esisterà un valore di n per cui la disequazione non è soddisfatta.

3.6 Qual è la complessità di $f(n) = 5$?

$$5 = \Theta(n^0) = \Theta(1)$$

3.7 Qual è la complessità di $f(n) = 5 + \sin(n)$?

Il seno è compreso fra -1 e 1; quindi comunque il valore può essere limitato superiormente e inferiormente da una una costante, e la complessità è $\Theta(1)$.

3.8 Logaritmo vs lineare

Vogliamo provare che $\log n = O(n)$. Dobbiamo dimostrare che

$$\exists c > 0, n_0 \geq 0 : \log n \leq cn, \forall n \geq n_0;$$

La dimostrazione è per induzione su n:

- per $n = 1$, $\log 1 = 0 \leq c$ per qualunque valore positivo di c;
- in generale, per $n \geq 1$:

$$\begin{aligned}
\log(n+1) &\leq \log(n+n) && \text{maggiorazione, } \forall n \geq 1 \\
&= \log 2n \\
&= \log 2 + \log n \\
&= 1 + \log n \\
&\leq 1 + cn && \text{per ipotesi induttiva} \\
&\leq c + cn, c \geq 1 \\
&= c(n+1), c \geq 1
\end{aligned}$$

Abbiamo quindi provato che $\log n \leq cn$ per qualsiasi $c \geq 1, n \geq 1$.

3.9 E se il logaritmo non è in base 2?

Non cambia niente. Sappiamo infatti che $\log_a n = (\log_a 2)(\log_2 n)$, per $a \neq 1 \neq b$, inoltre abbiamo già provato che $\log n \leq cn$, per opportuna costante c, per tanto:

$$\log_a n = (\log_a 2)(\log_2 n) \leq (\log_a 2)cn = dn \Leftrightarrow \log_a n = O(n)$$

con $d = c \log_a 2$.

3.10 E se invece di $\log n$ abbiamo $\log n^a, a > 1$?

Non cambia niente. Sappiamo che $\log n^a = a \log n$ e $\log n \leq cn$, per opportuna costante $c > 0$, quindi:

$$\log n^a = a \log n \leq acn \Leftrightarrow \log n^a = O(n)$$

3.11 Polilogaritmi contro polinomi

Generalizziamo; vogliamo dimostrare che per qualsiasi costante $a, b, k > 0$ vale $\log^a n^b = O(n^k)$.
Occorre provare che per ogni $a, b, k > 0$,

$$\exists c > 0, n_0 \geq 0 : (\log^a n^b) \leq cn^k, \forall n \geq n_0$$

Proviamolo innanzitutto nel caso $a = 1$. Ricordiamo che:

$$\log x^y = y \log x$$
$$\log x \leq dx, \forall n \geq 1, d \geq 1$$

Pertanto:

$$\log n^b = b \log n = b\frac{1}{k} k \log n = b\frac{1}{k} \log n^k \leq bd\frac{1}{k} n^k$$

dove $c = bd\frac{1}{k}$.
Per il caso generale:

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

(notare che abbiamo scelto di limitare il contenuto della parentesi con uno specifico valore di k).

3.12 Polinomi vs esponenziali

Vogliamo provare che $n^k = O(a^n), \forall k \geq 0, a > 1$. Dobbiamo dimostrare che:

$$\exists c > 0, n_0 \geq 0 : n^k \leq ca^n, \forall n \geq n_0$$

Si può procedere in questo modo:

$$\begin{aligned} \exists c > 0 : \log_a n^k \leq cn, \forall n \geq 1 \quad &\Leftrightarrow \quad \text{Logaritmi vs lineari} \\ a^{\log_a n^k} \leq a^{cn} \quad &\Leftrightarrow \quad \text{Esponenziazione} \\ n^k \leq a^c a^n \end{aligned}$$

per tutti i valori di $n \geq 1$.

3.13 Esponenziali con base diversa

Qual è il rapporto fra 2^n e 3^n? Ovviamente $2^n = O(3^n)$, ma è vero anche il contrario $3^n = O(2^n)$?

$$3^n = \left(\frac{3}{2} \cdot 2\right)^n = \left(\frac{3}{2}\right)^n 2^n \leq c2^n$$

Quindi $c \geq \left(\frac{3}{2}\right)^n$, ovviamente impossibile.

3.14 Esponenziale: 2^{n+1}

2^{n+1} è $\Theta(2^n)$ se e solo se $\exists c_1 \geq 0, \exists c_2 \geq 0, n_0 > 0$ tale che

$$c_1 2^n \leq 2^{n+1} \leq c_2 2^n, \forall n \geq n_0$$

E' facile vedere che $2^{n+1} = 2 \cdot 2^n$, quindi $c_1 \leq 2, c_2 \geq 2, n_0 = 1$.

3.15 Esponenziale: 2^{2n}

2^{2n} è $O(2^n)$ implica $2^{2n} = 2^n \cdot 2^n \leq c2^n$, quindi $c \geq 2^n$, il che è ovviamente assurdo perché non esiste una c costante.

3.16 Notazione o e ω

Abbiamo dimostrato che $\log^a n = O(n^k)$, ma è vero anche il contrario, ovvero $n^k = O(\log^a n)$? La risposta è no; introduciamo una nuova notazione per descrivere questo fatto.

Definizione 5 (Notazione o)**.** *Sia $g(n)$ una funzione di costo; indichiamo con $o(g(n))$ l'insieme delle funzioni $f(n)$ tali per cui:*

$$\forall c_2, \exists n_0 : f(n) < c_2 g(n), \forall n \geq n_0.$$

Definizione 6 (Notazione ω)**.** *Sia $g(n)$ una funzione di costo; indichiamo con $\omega(g(n))$ l'insieme delle funzioni $f(n)$ tali per cui esistono delle costanti positive c_1, n_0 tali che*

$$\forall c_1, \exists n_0 : c_1 g(n) < f(n), \forall n \geq n_0.$$

Queste due notazioni vengono utilizzate per definire limiti "non stretti". si noti la differenza fra o e O, oppure fra ω e Ω: la relazione deve valere per tutte le possibili costanti c. Intuitivamente, nella notazione o la funzione $f(n)$ diventa insignificante rispetto a $g(n)$ quando n tende all'infinito; ovvero

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

Si noti che:

$$\begin{aligned} f(n) = o(g(n)) &\Rightarrow f(n) = O(g(n)) \\ f(n) = \omega(g(n)) &\Rightarrow f(n) = \Omega(g(n)) \end{aligned}$$

Utilizzando il concetto di limite, date due funzioni $f(n)$ e $g(n)$ si possono fare le seguenti affermazioni:

- se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, allora $f(n) = o(g(n))$;
- se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \neq 0$, allora $f(n) = \Theta(g(n))$;
- se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, allora $f(n) = \omega(g(n))$;

3.17 Classificazione delle funzioni

E' possibile trarre un'ordinamento di tutte le funzioni, estendendo le relazioni che abbiamo dimostrato fino ad ora: Per qualsiasi $r < s, h < k, a < b$:

$$O(1) \subset O(\log^r n) \subset O(\log^s n) \subset O(n^h) \subset O(n^h \log^r n) \subset O(n^h \log^s n) \subset O(n^k) \subset O(a^n) \subset O(b^n)$$

3.18 Alcune utili regole

3.18.1 Dualità

$$\begin{aligned} f(n) = O(g(n)) &\Leftrightarrow g(n) = \Omega(f(n)) \\ f(n) = o(g(n)) &\Leftrightarrow g(n) = \omega(f(n)) \end{aligned}$$

Dimostriamo la prima delle due:

$$\begin{aligned} f(n) = O(g(n)) &\Leftrightarrow f(n) \leq cg(n), \forall n \geq n_0 \\ &\Leftrightarrow g(n) \geq \frac{1}{c} f(n), \forall n \geq n_0 \\ &\Leftrightarrow g(n) = \Omega(f(n)) \end{aligned}$$

Questa regola ci permette di semplificare le dimostrazioni di molte proprietà.

3.18.2 Eliminazione delle costanti

$$f(n) = O(g(n)) \Leftrightarrow af(n) = O(g(n)), \forall a > 0$$

Dimostriamo la prima relazione:

$$f(n) = O(g(n)) \Leftrightarrow \exists c > 0, n_0 \geq 0 : f(n) \leq cg(n), \forall n > n_0 \Leftrightarrow c > 0, n_0 \geq 0 : af(n) \leq acg(n), \forall n > n_0$$

Naturalmente la proprietà vale anche per le notazioni $\Omega, \Theta, o, \omega$

3.18.3 Sommatoria

$$f_1(n) = O(g_1(n)), f_2(n) = O(g_2(n)) \Rightarrow f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$$

Infatti,

$$\begin{aligned} f_1(n) = O(g_1(n)) \wedge f_2(n) = O(g_2(n)) &\Rightarrow \\ f_1(n) \leq c_1 g_1(n) \wedge f_2(n) \leq c_2 g_2(n) &\Rightarrow \\ f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n) &\Rightarrow \\ f_1(n) + f_2(n) \leq \max\{c_1, c_2\}(g_1(n) + g_2(n)) & \end{aligned}$$

Nota: per ragioni di chiarezza, sono stati omessi tutti i quantificatori esistenziali e universali. Naturalmente la proprietà vale anche per le notazioni $\Omega, \Theta, o, \omega$.

3.18.4 Prodotto

$$f_1(n) = O(g_1(n)), f_2(n) = O(g_2(n)) \Rightarrow f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$$

Infatti,

$$\begin{aligned} f_1(n) = O(g_1(n)) \wedge f_2(n) = O(g_2(n)) &\Rightarrow \\ f_1(n) \leq c_1 g_1(n) \wedge f_2(n) \leq c_2 g_2(n) &\Rightarrow \\ f_1(n) \cdot f_2(n) \leq c_1 c_2 g_1(n) g_2(n) & \end{aligned}$$

Nota: per ragioni di chiarezza, sono stati omessi tutti i quantificatori esistenziali e universali. Naturalmente la proprietà vale anche per le notazioni $\Omega, \Theta, o, \omega$.

3.18.5 Simmetria

$$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$$

Grazie alla proprietà di dualità:

$$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n)) \quad (1)$$

$$f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n)) \Rightarrow g(n) = O(f(n)) \quad (2)$$

$$(3)$$

3.18.6 Transitività

$$f(n) = O(g(n) \wedge g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$

Infatti:

$$\begin{aligned} f(n) = O(g(n) \wedge g(n) = O(h(n)) &\Rightarrow \\ f(n) \leq c_1 g(n) \wedge g(n) \leq c_2 h(n) &\Rightarrow \\ f(n) \leq c_1 c_2 h(n) & \end{aligned}$$

Nota: per ragioni di chiarezza, sono stati omessi tutti i quantificatori esistenziali e universali. Naturalmente la proprietà vale anche per le notazioni $\Omega, \Theta, o, \omega$.

3.19 Problemi vs Algoritmi

Complessità in tempo di un algoritmo: *la più grande quantità di tempo richiesta per un input di dimensione* n

- $O(f(n))$: Dimostrare che per ogni input di dimensione n, l'algoritmo richiede non più di $c \cdot f(n)$ per qualche c; ovvero, caso pessimo.
- $\Omega(f(n))$: Trovare un input di dimensione n, tale per cui l'algoritmo richiede almeno $c \cdot f(n)$
- $\Theta(f(n))$: Entrambe

Complessità in tempo di un problema computazionale: *La complessità in tempo dell'algoritmo più veloce che risolve tale problema*

- $O(f(n))$: Trovare un algoritmo che risolva il problema in tempo $O(f(n))$
- $\Omega(f(n))$: Dimostrare che nessun algoritmo può risolvere il problema più velocemente di $\Omega(f(n))$
- $\Theta(f(n))$: Entrambe

Alcune affermazioni in tal senso:

- Il tempo di esecuzione di Insertion Sort è compreso fra $\Omega(n)$ e $O(n^2)$.
- La complessità di Insertion Sort non è $\Omega(n^2)$.
- La complessità di Insertion Sort nel caso peggiore è $\Omega(n^2)$.
- La complessità di Merge Sort è $\Theta(n \log n)$.
- La complessità del problema dell'ordinamento è $O(n \log n)$.
- Che dire sui limiti inferiori del problema dell'ordinamento?

3.20 Abuso di notazione

Uso della notazione asintotica nelle equazione e disequazioni:

$$2n^2 + 5n = 2n^2 + \Theta(n)$$
$$2n^2 + \Theta(n) = \Theta(n^2)$$

3.21 Algoritmi ricorsivi

Nel caso degli algoritmi ricorsivi, il tempo di esecuzione può essere descritto da una **funzione di ricorsione**, ovvero un'equazione che descrive una funzione in termini del suo valore con input più piccoli. Esempio di merge sort:

$$T(n) = \begin{cases} 2T(n/2) + \Theta(n) & n > 1 \\ \Theta(1) & n = 1 \end{cases}$$

3.22 Come risolvere le ricorrenze?

- **Metodo di sostituzione**: Ipotizziamo una soluzione e utilizziamo l'induzione matematica per dimostrare che la nostra ipotesi è corretta.
- **Metodo dell'albero di ricorsione**: Converte la ricorrenza in un albero i cui nodi rappresentano i costi ai vari livelli della ricorsione
- **Metodo dell'esperto (Master Theorem)**: Fornisce una soluzione per ricorrenze della forma: $T(n) = aT(N/b) + f(n)$ con $a > 0, b > 1$

Alcuni dettagli tecnici:

- Il valore di n è sempre un intero. Spesso però scriviamo le equazioni ignorando questo fatto. Esempio: la vera funzione di Merge Sort è:

$$T(n) = \begin{cases} \lfloor T(n/2) \rfloor + \lceil T(n/2) \rceil + \Theta(n) & n > 1 \\ \Theta(1) & n = 1 \end{cases}$$

- Condizioni al contorno: cosa succede per $n = 1$? Normalmente "dimentichiamo" questo caso. Ma bisogna essere sicuri di ciò che si fa . . . Nel seguito, verificheremo i vari casi.

3.23 Metodo della sostituzione

L'idea è quella di ipotizzare una soluzione, basata sulla propria esperienza, e dimostrare che questa soluzione è vera tramite induzione.

3.23.1 Esempio semplice, facile da dimostrare

$$T(n) = \begin{cases} \lfloor T(n/2) \rfloor + n & n > 1 \\ 1 & n = 1 \end{cases}$$

Guess: $T(n) = O(n)$; dobbiamo dimostrare che $\exists c \geq 0, n_0 \geq 0 : T(n) \leq cn, \forall n \geq n_0$.

- **Caso base**: $T(1) = 1 \leq 1 \cdot c, \forall c \geq 1$.
- **Ipotesi induttiva**: $\forall n' < n : T(n') \leq cn'$.
- **Caso induttivo**:

$$\begin{aligned} T(n) &= \lfloor T(n/2) \rfloor + n \\ &\leq c\lfloor n/2 \rfloor + n && \text{Sostituzione dell'ipotesi induttiva} \\ &\leq cn/2 + n && \text{Intero inferiore} \\ &= (c/2 + 1)n \\ &\leq cn \end{aligned}$$

L'ultima disequazione è vera se e solo se

$$(c/2 + 1) \leq c \Leftrightarrow c \geq 2$$

Quindi abbiamo dimostrato, scegliendo $c = 2$, che $T(n) \leq 2n = O(n)$. Nota: abbiamo anche ottenuto una stima precisa delle costanti coinvolte. E' ragionevole?

$$n + n/2 + n/4 + n/8 + \ldots + 1 = 2n - 1$$

3.23.2 Cosa succede se si sbaglia l'intuizione

$$T(n) = \begin{cases} T(n-1) + n & n > 1 \\ 1 & n = 1 \end{cases}$$

E' facile vedere che $T(n) = \Theta(n^2)$. Supponiamo invece di voler dimostrare che $T(n) = O(n)$. E' possibile? No.

- **Ipotesi induttiva**: Possiamo lavorare su un'ipotesi induttiva come questa: $T(n-1) = O(n-1)$? E quindi scrivere:

$$T(n) = n + O(n-1) = O(n)?$$

No, perché dobbiamo usare le costanti.

$$\begin{aligned} T(N) &= n + T(n-1) \\ &\leq c(n-1) + n \\ &\leq (c+1)n - c \\ &\leq (c+1)n \\ &\not\leq cn \end{aligned}$$

L'ultima disequazione non è soddisfacibile.

3.23.3 Problema: difficoltà matematica

Ci sono casi in cui è possibile ipotizzare la soluzione giusta, ma poi è difficile risolvere la parte induttiva.

$$T(n) = \begin{cases} T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 & n > 1 \\ 1 & n = 1 \end{cases}$$

Supponiamo la soluzione sia $O(n)$. Dobbiamo dimostrare che $T(n) \leq cn, c \geq 0$.

- **Ipotesi induttiva**:

  $$\begin{aligned} T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \\ &\leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\ &= cn + 1 \\ &\not\leq cn \end{aligned}$$

  Ovviamente l'ultima disequazione è ovviamente falsa. Cosa fare? Il problema qui è che la soluzione è giusta, ma non riusciamo a dimostrarlo per un piccolo fattore di ordine inferiore (1).

- **Ipotesi induttiva, più forte** Proviamo con un'ipotesi induttiva più forte: sottraiamo un termine di ordine inferiore:

  $$T(n) \leq cn - b$$

  Proviamo ad utilizzarla:

  $$\begin{aligned} T(n) &= (c\lfloor n/2 \rfloor - b) + (c\lceil n/2 \rceil - b) + 1 \\ &= cn - 2b + 1 \\ &\leq cn - b \end{aligned}$$

  L'ultima disequazione è vera se

  $$-2b + 1 \leq -b \Leftrightarrow b \geq 1.$$

- **Caso base**:

  $$T(1) = 1 \leq c - b$$

  che è vera per $\forall c \geq b + 1$.

3.23.4 Problema con i casi base

$$T(n) = \begin{cases} 2T(\lfloor n/2 \rfloor) + n & n > 1 \\ 1 & n = 1 \end{cases}$$

Guess: $T(n) = O(n \log n) \Leftrightarrow T(n) \leq cn \log n$.

- **Passo induttivo**:

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2cn/2 \log n/2 + n \\ &= cn \log n/2 + n \\ &= cn(\log n - 1) + n \\ &= cn \log n - cn + n \\ &\leq cn \log n \end{aligned}$$

L'ultima disequazione è vera per $c \geq 1$.

- Proviamo con il passo base:

$$T(n) = 1 \not\leq c \cdot 1 \log 1 = 0$$

E' falso, ma non è un problema. Non a caso si chiama notazione asintotica; il valore iniziale di n_0 lo possiamo scegliere noi.

Quindi:

$$\begin{aligned} T(2) &= 2T(\lfloor 1 \rfloor) + 2 = 4 \leq c \cdot 2 \log 2 \\ T(3) &= 2T(\lfloor 1 \rfloor) + 3 = 5 \leq c \cdot 3 \log 3 \\ T(4) &= 2T(\lfloor 2 \rfloor) + 4 \end{aligned}$$

E' facile trovare un valore di c che soddisfi le prime due disequazioni; per la terza, non viene espressa come $T(1)$, quindi non è necessario andare avanti.

3.23.5 All togheter now!

$$T(n) = \begin{cases} 9T(\lfloor n/3 \rfloor) + n & n > 1 \\ 1 & n = 1 \\ 2 & n = 2 \end{cases}$$

- **Ipotesi induttiva**: $T(n) = O(n^2), T(n) \leq cn^2$
- **Passo induttivo**:

$$\begin{aligned} T(n) &= 9T(\lfloor n/3 \rfloor) + n \\ &\leq 9c(\lfloor n/3 \rfloor)^2 + n \\ &\leq 9c(n^2/9) + n \\ &= cn^2 + n \\ &\not\leq cn^2 \end{aligned}$$

L'ipotesi è corretta tuttavia sottraiamo un valore di ordine inferiore.

- **Ipotesi induttiva piú forte**: $T(n) \leq c(n^2 - n)$
- **Passo induttivo**:

$$\begin{aligned} T(n) &= 9T(\lfloor n/3 \rfloor) + n \\ &\leq 9c(\lfloor n/3 \rfloor^2 - \lfloor n/3 \rfloor) + n \\ &\leq cn^2 - 3cn + n \\ &\leq cn^2 - cn \end{aligned}$$

L'ultima disequazione è vera per $c \geq 1/2$.

- **Passo base**:
  - $T(1) = 1 \geq c(1^2 - 1) = 0$, falso
  - $T(2) = 9T(1) + 2 = 11 \geq c(4 - 1) \Rightarrow c \leq \frac{11}{3}$
  - $T(3) = 9T(1) + 3 = 12 \geq c(9 - 1) \Rightarrow c \leq \frac{3}{2}$
  - $T(4) = 9T(1) + 4 = 13 \geq c(16 - 1) \Rightarrow c \leq \frac{13}{15}$
  - $T(5) = 9T(1) + 5 = 14 \geq c(25 - 1) \Rightarrow c \leq \frac{14}{24}$
  - $T(6) = 9T(2) \ldots$

3.24 Metodo dell'iterazione o dell'albero di ricorsione

3.24.1 $T(n) = T(n/2) + c$

Questa è la funzione di ricorrenza della ricerca binaria:

$$T(n) = \begin{cases} T(n/2) + c & n > 1 \\ 1 & n = 1 \end{cases}$$

E' possibile risolverla in questa maniera:

$$\begin{aligned} T(n) &= c + T(n/2) \\ &= c + c + T(n/4) \\ &= c + c + c + T(n/8) \\ &= c + c + c + \ldots + T(1) \end{aligned}$$

$n/2^i = 1$ quando $i = \log n$; quindi $T(n) = c \log n + T(1) = O(\log n)$.

3.24.2 $T(n) = 4T(n/2) + n$

$$T(n) = \begin{cases} 4T(n/2) + n & n > 1 \\ 1 & n = 1 \end{cases}$$

Normalmente, applicare questo metodo non è così semplice. Ci vogliono una serie di manipolazioni algebriche, come quelle mostrate qui sotto.

$$\begin{aligned} T(n) &= n + 4T(n/2) \\ &= n + 4n/2 + 16T(n/4) \\ &= n + 2n + 16n/4 + 64T(n/8) \\ &= \ldots \\ &= n + 2n + 4n + 8n + \ldots + 2^{\log n - 1} n + 4^{\log n} T(1) \\ &= n \sum_{j=0}^{\log n - 1} 2^j + 4^{\log n} \end{aligned}$$

Infatti, $n/2^i = 1$ quando $i = \log n$.
Abbiamo due termini, che trattiamo separatamente. Innanzittuto sfruttiamo questa formula:

$$a^{\log_b n} = n^{\log_b a} \qquad (3.15)$$

Quindi scriviamo:

$$4^{\log_2 n} = n^{\log_2 4} = n^2$$

Si consideri poi la serie geometrica:

$$\sum_{j=0}^{k} x^j = \frac{x^{k+1}-1}{x-1}, x \neq 0 \qquad (A.5)$$

Applicando questa formula alla prima parte della nostra espressione:

$$n \sum_{j=0}^{\log n - 1} 2^j = n\frac{2^{\log n}-1}{2-1} = n(n-1)$$

Quindi il costo totale dell'algoritmo è $\Theta(n^2)$.

3.24.3 $T(n) = 4T(n/2) + n^3$

Proviamo a risolvere tramite il metodo dell'albero della ricorrenza; un modo diverso di vedere quello che abbiamo già visto.

$$\begin{array}{cc}
n^3 & 4^0 n^3 \\
\frac{n^3}{2^3} \quad \frac{n^3}{2^3} \quad \frac{n^3}{2^3} \quad \frac{n^3}{2^3} & 4^1 \frac{n^3}{2^3} \\
\frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} \quad \frac{n^3}{2^6} & 4^2 \frac{n^3}{2^6} \\
\cdots & \\
\frac{n^3}{2^{3(\log n-1)}} \quad \frac{n^3}{2^{3(\log n-1)}} \quad \frac{n^3}{2^{3(\log n-1)}} \quad \frac{n^3}{2^{3(\log n-1)}} \quad \cdots \quad \frac{n^3}{2^{3(\log n-1)}} & 4^{\log n-1}\frac{n^3}{2^{3(\log n-1)}} \\
T(1) \quad T(1) \quad T(1) \ldots T(1) & 4^{\log n}
\end{array}$$

Ancora una volta, la ricorsione termina quando la dimensione del problema raggiunge 1; infatti, $n/2^i = 1$ quando $i = \log n$.
L'ultima riga costa $4^{\log n}T(1) = \Theta(n^2)$. Sommando le righe precedenti si ottiene:

$$\begin{aligned}
T(n) &= \sum_{i=0}^{\log n-1} n^3/2^{3i} \cdot 4^i \\
&= n^3 \sum_{i=0}^{\log n-1} \frac{2^{2i}}{2^{3i}} \\
&= n^3 \sum_{i=0}^{\log n-1} \left(\frac{1}{2}\right)^i \\
&\leq n^3 \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i && \text{Estensione della sommatoria} \\
&= \frac{1}{1-\frac{1}{2}} n^3 && \text{Serie geometrica}
\end{aligned}$$

da cui si ottiene:

$$T(n) = n^2 + 2n^3 = O(n^3)$$

3.24.4 $T(n) = 4T(n/2) + n^2$

Utilizziamo l'albero della ricorrenza per descrivere i vari termini.

$$
\begin{array}{cccccccccccccccc}
 & & & & & & & n^2 & & & & & & & & 4^0 n^3 \\
 & & & & & & \frac{n^2}{2^2} & \frac{n^2}{2^2} & \frac{n^2}{2^2} & \frac{n^2}{2^2} & & & & & & 4^1 \frac{n^2}{2^2} \\
\frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & \frac{n^2}{2^4} & 4^2 \frac{n^2}{2^4} \\
 & & & & & & & \cdots & & & & & & & & \\
 & & \frac{n^3}{2^{2(\log n-1)}} & & \frac{n^3}{2^{2(\log n-1)}} & & \frac{n^3}{2^{2(\log n-1)}} & & \frac{n^3}{2^{2(\log n-1)}} & & \cdots & & \frac{n^3}{2^{2(\log n-1)}} & & & 4^{\log n-1} \frac{n^3}{2^{2(\log n-1)}} \\
 & & & & & & T(1) & T(1) & T(1) \ldots T(1) & & & & & & & 4^{\log n}
\end{array}
$$

Ancora una volta, la ricorsione termina quando la dimensione del problema raggiunge 1; infatti, $n/2^i = 1$ quando $i = \log n$.
L'ultima riga costa $4^{\log n} T(1) = \Theta(n^2)$. Sommando le righe precedenti si ottiene:

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log n-1} n^2/2^{2i} \cdot 4^i \\
&= n^2 \sum_{i=0}^{\log n-1} \frac{2^{2i}}{2^{2i}} \\
&= n^2 \sum_{i=0}^{\log n-1} 1 \\
&= n^2 \log n
\end{aligned}
$$

da cui si ottiene:

$$T(n) = n^2 + n^2 \log n = O(n^2 \log n)$$

3.25 Metodo dell'esperto

Si considerino ricorrenze della forma:

$$T(n) = \begin{cases} aT(n/b) + f(n) & n > 1 \\ \Theta(1) & n = 1 \end{cases}$$

con $a \geq 1$, $b \geq 1$ e $f(n)$ asintoticamente positiva. Sono generate da algoritmi divide-et-impera, dove il costo dell'operazione divide et combina è rappresentano da $f(n)$. Esempio:

MergeSort: $a = 2$, $b = 2$, $f(n) = \Theta(n)$

Nota: la ricorrenza spesso non è così ben definita per via degli arrotondamenti, ma non importa. È possibile dimostrare che non c'è effetto (ma noi non lo faremo).

Teorema 1. *Sia $a \geq 1$, $b \geq 1$, $f(n)$ asintoticamente positiva, e sia*

$$T(n) = \begin{cases} aT(n/b) + f(n) & n > 1 \\ \Theta(1) & n = 1 \end{cases}$$

Allora $T(n)$ può essere limitata asintoticamente in uno dei modi seguenti:

- *se $f(n) = O(n^{\log_b a - \epsilon})$ per qualche $\epsilon > 0$, allora*

$$T(n) = \Theta(n^{\log_b a})$$

- *se $f(n) = \Theta(n^{\log_b a})$, allora*

$$T(n)\Theta(n^{\log_b a} \log n) = \Theta(f(n) \log n)$$

- *se* $f(n) = \Omega(n^{\log_b a+\epsilon})$ *per qualche* $\epsilon > 0$*, e* $af(n/b) \leq cf(n)$ *per qualche costante* $c < 1$ *e per ogni* n *sufficientemente grande, allora*

$$T(n) = \Theta(f(n))$$

A parole, cosa ci di questo teorema? Se si confronta $n^{\log_b a}$, si ottiene:

- se $n^{\log_b a}$ è più grande di $f(n)$ di un fattore polinomiale n^ϵ, allora $T(n)$ è dominato da $n^{\log_b a}$;
- se $n^{\log_b a}$ è $\Theta(n^{\log_b a}$, allora nessuno dei due domina, e viene aggiunto un fattore logaritmico;
- se $n^{\log_b a}$ è più grande di $f(n)$ di un fattore polinomiale n^ϵ, allora $T(n)$ è dominato da $f(n)$.

Esempi teorema dell'esperto:

3.25.1 $T(n) = 9T(n/3) + n$

$a = 9, b = 3, f(n) = n$;
$n^{\log_b a} = n^2 = \Theta(n^2)$; da cui è possibile notare che:
$f(n) = O(n^{2-\epsilon})$, con $\epsilon = 1$.
CASO 1: $T(n) = \Theta(n^2)$.
Si veda la soluzione di $T(n) = T(n/2) + n$ tramite albero di ricorsione.

3.25.2 $T(n) = T(2n/3) + 1$

$a = 1, b = 3/2, f(n) = 1$;
$n^{\log_b a} = n^{\log_b 1} = n^0 = 1$;
$f(n) = \Theta(n^{\log_b a})$;
CASO 2: $T(n) = \Theta(f(n) \log n) = \Theta(\log n)$

3.25.3 $T(n) = 3T(n/4) + n \log n$

$a = 3, b = 4, f(n) = n \log n$;
$n^{\log_b a} = n^{\log_4 3} = n^{0.793}$;
$f(n) = n \log n = \Omega(n^{\log_4 3+\epsilon}))$, dove $\epsilon \approx 0.2$.
CASO 3: $T(n) = \Theta(f(n)) = \Theta(n \log n)$
Ma dobbiamo anche dimostrare che: $\exists c, n_0 \geq 0 : af(n/b) = cf(n), \forall n \geq n_0$. Me è facile vedere che $af(n/b) = 3/4n$, quindi possiamo prendere $c = 3/4$ e $n_0 = 1$.

3.25.4 $T(n) = 2T(n/2) + n \log n$

$a = 2, b = 2, f(n) = n \log n$;
$n^{\log_2 2} = n; f(n) = n \log n = \Omega(n)$, ma non è polinomialmente più grande. Bisogna utilizzare altri metodi.

3.26 Dimostrazione del Master Theorem

Non vediamo una dimostrazione completa, che potete trovare tranquillamente sul libro, con tutti i piũ sottili dettagli tecnici; dimostriamo invece uno solo dei casi, quello in cui $n^{\log_b a} = \Theta(f(n))$, facendo una serie di assunzioni semplificative:

- $n = b^k$ (elimina i problemi di intero superiore/inferiore)
- la ricorsione si ferma quando $n = 1$

Utilizziamo il metodo dell'albero di ricorrenza per risolvere il teorema; dalle due assunzioni precedenti è possibile ricavere le seguenti proprietà:

- I sottoproblemi al livello i dell'albero della ricorsione hanno dimensione n/b^i;
- Il contributo di un nodo al livello i al tempo di esecuzione è $f(n/b^i)$;
- Il numero di livelli dell'albero è $\log_b n$.
- Il numero di nodi al livello i è a^i.

Da queste proprietà ricaviamo una soluzione per la ricorrenza:

$$T(n) = \sum_{i=0}^{\log_b n - 1} a^i f(n/b^i) + a^{\log_b n}$$

Per semplificare l'espressione e ottenere una limitazione superiore e inferiore Θ, ci basiamo sulla proprietà che $f(n) = Theta(n^{\log_b a})$ (dall'ipotesi del teorema) e che $n^{\log_b a} = \Theta(f(n))$ (per la proprietà di simmetria di Θ).

$$\begin{aligned}
T(n) &= \sum_{i=0}^{\log_b n - 1} a^i f(n/b^i) + a^{\log_b n} \\
&= \sum_{i=0}^{\log_b n - 1} a^i f(n/b^i) + n^{\log_b a} && \text{Riscrittura del secondo termine} \\
&= \sum_{i=0}^{\log_b n - 1} \Theta(a^i (n/b^i)^{\log_b a}) + n^{\log_b a} && \text{Sostituzione del termine } \Theta \\
&= \sum_{i=0}^{\log_b n - 1} \Theta(n^{\log_b a} \left(\frac{a}{b^{\log_b a}}\right)^i) + n^{\log_b a} \\
&= \sum_{i=0}^{\log_b n - 1} \Theta(n^{\log_b a} \left(\frac{a}{a}\right)^i) + n^{\log_b a} \\
&= \sum_{i=0}^{\log_b n - 1} \Theta(n^{\log_b a}) + n^{\log_b a} \\
&= \Theta(n^{\log_b a}) \log_b n + n^{\log_b a} \\
&= \Theta(n^{\log_b a} \log n)
\end{aligned}$$

CVD!

3.27 Esercizio

Siano

- $T(n) = 7T(n/2) + n^2$ una funzione di costo di un algoritmo A, e
- $T'(n) = aT'(n/4) + n^2$ una funzione di costo di un algoritmo A'.

Qual è il massimo valore intero di a che rende A' asintoticamente più veloce di A?

Soluzione : Poichè $\log_2 7$ è $\Omega(n^2 + \epsilon)$, utilizzando il master theorem si ottiene che $T(n) = O(n^{\log_2 7})$. Utilizzando alcune trasformazioni algebriche, si ottiene che:

$$\begin{aligned}
\log_2 7 &= \frac{\log_4 7}{\log_4 2} \\
&= \frac{\log_4 7}{\frac{1}{7} 2} \\
&= 2 \log_4 7 && = \log_4 49
\end{aligned}$$

Nel caso di T', dobbiamo confrontare $n^{\log_4 a}$ e n^2. Con valori di a inferiori a 16 ($\log_4 16 = 2$) domina il fattore n^2, e quindi $T'(n) = \Theta(n^2)$. Con $a = 16$, abbiamo che $T'(n) = n^2 \log n$. In entrambi i casi, T' è $O(T(n))$. Con valori $a > 16$, il Master theorem dice che $T'(n) = O(n^{\log_4 a})$. In questo caso, i valori < 49 fanno sì che A' sia più veloce di A.