Algoritmi e Strutture Dati

Analisi di algoritmi
Funzioni di costo, notazione asintotica

Alberto Montresor and Davide Rossi

Università di Bologna

21 ottobre 2024

Notazioni O, Ω, Θ

Definizione – Notazione O

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

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

- Come si legge: $f(n)$ è "O grande" (big-O) di $g(n)$
- Come si scrive: $f(n) = O(g(n))$
- $g(n)$ è un limite asintotico superiore per $f(n)$
- $f(n)$ cresce al più come $g(n)$

Notazioni O, Ω, Θ

Definizione – Notazione Ω

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

$$\exists c > 0, \exists m \geq 0 : f(n) \geq cg(n), \forall n \geq m$$

- Come si legge: $f(n)$ è "Omega grande" di $g(n)$
- Come si scrive: $f(n) = \Omega(g(n))$
- $g(n)$ è un limite asintotico inferiore per $f(n)$
- $f(n)$ cresce almeno quanto $g(n)$

Notazioni O, Ω, Θ

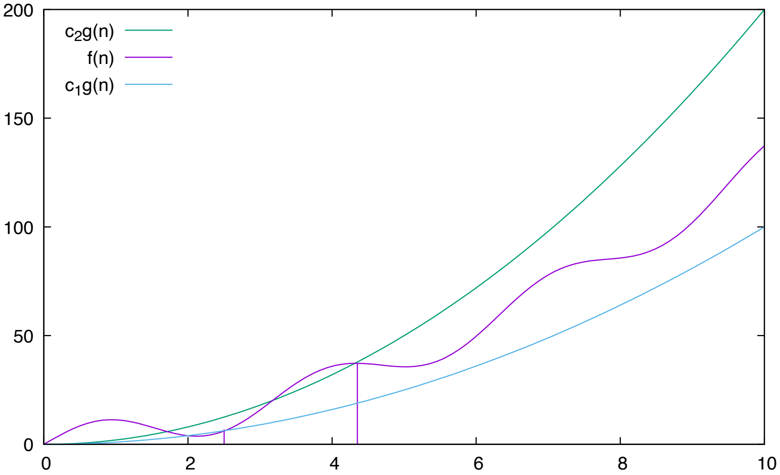
Definizione – Notazione Θ

Sia $g(n)$ una funzione di costo; indichiamo con $\Theta(g(n))$ l'insieme delle funzioni $f(n)$ tali per cui:

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

- Come si legge: $f(n)$ è "Theta" di $g(n)$
- Come si scrive: $f(n) = \Theta(g(n))$
- $f(n)$ cresce esattamente come $g(n)$
- $f(n) = \Theta(g(n))$ se e solo se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$

Graficamente



Algoritmi e Strutture Dati

Analisi di algoritmi
Proprietà della notazione asintotica

Alberto Montresor and Davide Rossi

Università di Bologna

21 ottobre 2024

Sommario

Regola generale

Espressioni polinomiali

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

Limite superiore: $\exists c > 0, \exists m \geq 0 : f(n) \leq cn^k, \forall n \geq m$

$$\begin{aligned}
f(n) &= a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \\
&\leq a_k n^k + |a_{k-1}| n^{k-1} + \ldots + |a_1| n + |a_0| \\
&\leq a_k n^k + |a_{k-1}| n^k + \ldots + |a_1| n^k + |a_0| n^k \qquad \forall n \geq 1 \\
&= (a_k + |a_{k-1}| + \ldots + |a_1| + |a_0|) n^k \\
&\overset{?}{\leq} cn^k
\end{aligned}$$

che è vera per $c \geq (a_k + |a_{k-1}| + \ldots + |a_1| + |a_0|) > 0$ e per $m = 1$.

Regola generale

Espressioni polinomiali

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

Limite inferiore: $\exists d > 0, \exists m \geq 0 : f(n) \geq dn^k, \forall n \geq m$

$$\begin{aligned}
f(n) &= a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0 \\
&\geq a_k n^k - |a_{k-1}| n^{k-1} - \dots - |a_1| n - |a_0| \\
&\geq a_k n^k - |a_{k-1}| n^{k-1} - \dots - |a_1| n^{k-1} - |a_0| n^{k-1} \qquad \forall n \geq 1 \\
&\stackrel{?}{\geq} dn^k
\end{aligned}$$

L'ultima equazione è vera se:

$$d \leq a_k - \frac{|a_{k-1}|}{n} - \frac{|a_{k-2}|}{n} - \dots - \frac{|a_1|}{n} - \frac{|a_0|}{n} > 0 \Leftrightarrow n > \frac{|a_{k-1}| + \dots + |a_0|}{a_k}$$

Proprietà

Dualità

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$

Dimostrazione:

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

Proprietà

Eliminazione delle costanti

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

Dimostrazione:

$$\begin{aligned}
f(n) = O(g(n)) &\Leftrightarrow f(n) \leq cg(n), \forall n \geq m \\
&\Leftrightarrow af(n) \leq acg(n), \forall n \geq m, \forall a \geq 0 \\
&\Leftrightarrow af(n) \leq c'g(n), \forall n \geq m, c' = ac > 0 \\
&\Leftrightarrow af(n) = O(g(n))
\end{aligned}$$

Proprietà

Sommatoria (sequenza di algoritmi)

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

Dimostrazione (Lato O)

$$\begin{aligned}
f_1(n) = O(g_1(n)) \wedge f_2(n) = O(g_2(n)) &\quad \Rightarrow \\
f_1(n) \leq c_1 g_1(n) \wedge f_2(n) \leq c_2 g_2(n) &\quad \Rightarrow \\
f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n) &\quad \Rightarrow \\
f_1(n) + f_2(n) \leq \max\{c_1, c_2\}(2 \cdot \max(g_1(n), g_2(n))) &\quad \Rightarrow \\
f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n))) &
\end{aligned}$$

Proprietà

Prodotto (Cicli annidati)

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

Dimostrazione

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

Proprietà

Simmetria

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

Dimostrazione

Grazie alla proprietà di dualità:

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

Proprietà

Transitività

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

Dimostrazione

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

Notazioni o, ω

Definizione – Notazioni o, ω

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

$$\forall c, \exists m : f(n) < cg(n), \forall n \geq m.$$

Sia $g(n)$ una funzione di costo; indichiamo con $\omega(g(n))$ l'insieme delle funzioni $f(n)$ tali per cui:

$$\forall c, \exists m : f(n) > cg(n), \forall n \geq m.$$

- Come si leggono: $f(n)$ è "o piccolo", "omega piccolo" di $g(n)$
- Come si scrivono: $f(n) = o(g(n))$ oppure $f(n) = \omega(g(n))$

Notazioni o, ω

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

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

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \neq 0 \Rightarrow f(n) = \Theta(g(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty \Rightarrow f(n) = \omega(g(n))$$

Si noti che:

$$f(n) = o(g(n)) \quad \Rightarrow \quad f(n) = O(g(n))$$

$$f(n) = \omega(g(n)) \quad \Rightarrow \quad f(n) = \Omega(g(n))$$

Classificazione delle funzioni

E' possibile trarre un'ordinamento delle principali espressioni, estendendo le relazioni che abbiamo dimostrato fino ad ora

Per ogni $r < s, h < k, a < b$:

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

Algoritmi e Strutture Dati

Analisi di algoritmi
Ricorrenze, metodo dell'albero di ricorsione

Alberto Montresor and Davide Rossi

Università di Bologna

21 ottobre 2024

Sommario

Introduzione

Equazioni di ricorrenza

Quando si calcola la complessità di un algoritmo ricorsivo, questa viene espressa tramite un'equazione di ricorrenza, ovvero una formula matematica definita in maniera... ricorsiva!

MergeSort

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

Introduzione

Forma chiusa

Il nostro obiettivo è ottenere, quando possibile, una formula chiusa che rappresenti la classe di complessità della funzione.

MergeSort

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

Introduzione

Forma chiusa

Il nostro obiettivo è ottenere, quando possibile, una formula chiusa che rappresenti la classe di complessità della funzione.

MergeSort

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

Metodo dell'albero di ricorsione, o per livelli

Metodi per risolvere ricorrenze

- Analisi per livelli
- Analisi per tentativi, o per sostituzione (vedi libro di testo)
- Metodo dell'esperto, o delle ricorrenze comuni

Metodo dell'albero di ricorsione, o per livelli

"Srotoliamo" la ricorrenza in un albero i cui nodi rappresentano i costi ai vari livelli della ricorsione

Primo esempio

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

È possibile risolvere questa ricorrenza nel modo seguente:

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

Assumiamo per semplicità:
$n = 2^k$, ovvero $k = \log n$

Primo esempio

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

È possibile risolvere questa ricorrenza nel modo seguente:

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

Assumiamo per semplicità: $n = 2^k$, ovvero $k = \log n$

$T(n) = b \log n + c = \Theta(\log n)$

Secondo esempio

$$T(n) = \begin{cases} 4T(n/2) + n^3 & n > 1 \\ 1 & n \leq 1 \end{cases}$$

Proviamo a visualizzare l'albero delle chiamate, per i primi tre livelli:

$$\overbrace{\left(\frac{n}{2}\right)^3 \quad \left(\frac{n}{2}\right)^3 \quad \left(\frac{n}{2}\right)^3 \quad \left(\frac{n}{2}\right)^3}^{n^3}$$

$$\overbrace{\left(\frac{n}{4}\right)^3 \left(\frac{n}{4}\right)^3 \left(\frac{n}{4}\right)^3 \left(\frac{n}{4}\right)^3} \overbrace{\left(\frac{n}{4}\right)^3 \left(\frac{n}{4}\right)^3 \left(\frac{n}{4}\right)^3 \left(\frac{n}{4}\right)^3} \overbrace{\left(\frac{n}{4}\right)^3 \left(\frac{n}{4}\right)^3 \left(\frac{n}{4}\right)^3 \left(\frac{n}{4}\right)^3} \overbrace{\left(\frac{n}{4}\right)^3 \left(\frac{n}{4}\right)^3 \left(\frac{n}{4}\right)^3 \left(\frac{n}{4}\right)^3}$$

Secondo esempio

$$T(n) = \begin{cases} 4T(n/2) + n^3 & n > 1 \\ 1 & n \leq 1 \end{cases}$$

Livello	Dim.	Costo chiam.	N. chiamate	Costo livello
0	n	n^3	1	n^3
1	$n/2$	$(n/2)^3$	4	$4(n/2)^3$
2	$n/4$	$(n/4)^3$	16	$16(n/4)^3$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
i	$n/2^i$	$(n/2^i)^3$	4^i	$4^i(n/2^i)^3$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$\ell - 1$	$n/2^{\ell-1}$	$(n/2^{\ell-1})^3$	$4^{\ell-1}$	$4^{\ell-1}(n/2^{\ell-1})^3$
$\ell = \log n$	1	$T(1)$	$4^{\log n}$	$4^{\log n}$

Secondo esempio

$$T(n) = \begin{cases} 4T(n/2) + n^3 & n > 1 \\ 1 & n \leq 1 \end{cases}$$

La sommatoria dà origine a:

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log n - 1} 4^i \cdot n^3/2^{3i} \; + 4^{\log n} \\ &= n^3 \sum_{i=0}^{\log n - 1} \frac{2^{2i}}{2^{3i}} \; + 4^{\log n} \\ &= n^3 \sum_{i=0}^{\log n - 1} \left(\frac{1}{2}\right)^i \; + 4^{\log n} \end{aligned}$$

Passaggi algebrici

Passaggi algebrici

Secondo esempio

$$T(n) = \begin{cases} 4T(n/2) + n^3 & n > 1 \\ 1 & n \leq 1 \end{cases}$$

La sommatoria dà origine a:

$$\begin{aligned} T(n) &= n^3 \sum_{i=0}^{\log n - 1} \left(\frac{1}{2}\right)^i + 4^{\log n} \\ &= n^3 \sum_{i=0}^{\log n - 1} \left(\frac{1}{2}\right)^i + n^2 \\ &\leq n^3 \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i + n^2 \end{aligned}$$

Cambiamento di base

Estensione della sommatoria

Secondo esempio

$$T(n) = \begin{cases} 4T(n/2) + n^3 & n > 1 \\ 1 & n \leq 1 \end{cases}$$

La sommatoria dà origine a:

$$\begin{aligned} T(n) &\leq n^3 \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i + n^2 \\ &= n^3 \cdot \frac{1}{1 - \frac{1}{2}} + n^2 \\ &= 2n^3 + n^2 \end{aligned}$$

Serie geometrica infinita decrescente:

$$\forall x, |x| < 1 : \sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$

Secondo esempio

$$T(n) = \begin{cases} 4T(n/2) + n^3 & n > 1 \\ 1 & n \leq 1 \end{cases}$$

Abbiamo dimostrato che:

$$T(n) \leq 2n^3 + n^2$$

- Possiamo affermare che $T(n) = O(n^3)$
- La dimostrazione precedente non afferma che $T(n) = \Theta(n^3)$, perché ad un certo punto siamo passati a $\leq$
- Però è possibile notare che $T(n) \geq n^3$, quindi è possibile affermare che $T(n) = \Omega(n^3)$ e quindi $T(n) = \Theta(n^3)$

Terzo esempio

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

Livello	**Dimensione**	**Costo chiamata**	**N. chiamate**	**Costo livello**
0	n	n^2	1	n^2
1	$n/2$	$(n/2)^2$	4	$4(n/2)^2$
2	$n/4$	$(n/4)^2$	16	$16(n/4)^2$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
i	$n/2^i$	$(n/2^i)^2$	4^i	$4^i(n/2^i)^2$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$\ell - 1$	$n/2^{\ell-1}$	$(n/2^{\ell-1})^2$	$4^{\ell-1}$	$4^{\ell-1}(n/2^{\ell-1})^2$
$\ell = \log n$	1	$T(1)$	$4^{\log n}$	$4^{\log n}$

Terzo esempio

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

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

Algoritmi e Strutture Dati

Analisi di algoritmi
Ricorrenze comuni

Alberto Montresor and Davide Rossi

Università di Bologna

21 ottobre 2024

Metodo dell'esperto

Metodi per risolvere ricorrenze

- Metodo dell'albero di ricorsione, o per livelli
- Metodo di sostituzione, o per tentativi (vedi libro di testo)
- Metodo dell'esperto, o delle ricorrenze comuni

Ricorrenze comuni

Esiste un'ampia classe di ricorrenze che possono essere risolte facilmente facendo ricorso ad alcuni teoremi, ognuno dei quali si occupa di una classe particolare di equazioni di ricorrenza.

Ricorrenze lineari con partizione bilanciata

Teorema

Siano a e b costanti intere tali che $a \geq 1$ e $b \geq 2$, e c, β costanti reali tali che $c > 0$ e $\beta \geq 0$. Sia $T(n)$ data dalla relazione di ricorrenza:

$$T(n) = \begin{cases} aT(n/b) + cn^\beta & n > 1 \\ d & n \leq 1 \end{cases}$$

Posto $\alpha = \log a / \log b = \log_b a$, allora:

$$T(n) = \begin{cases} \Theta(n^\alpha) & \alpha > \beta \\ \Theta(n^\alpha \log n) & \alpha = \beta \\ \Theta(n^\beta) & \alpha < \beta \end{cases}$$

Ricorrenze lineari con partizione bilanciata

Assunzioni

Assumiamo che n sia una potenza intera di b: $n = b^k, k = \log_b n$

Perchè ci serve?

Semplifica tutti i calcoli successivi

Influisce sul risultato?

- Supponiamo che l'input abbia dimensione $b^k + 1$
- Estendiamo l'input fino ad una dimensione b^{k+1} (padding)
- L'input è stato esteso al massimo di un fattore costante b
- Ininfluente al fine della complessità computazionale

Ricorrenze lineari con partizione bilanciata

$$T(n) = aT(n/b) + cn^{\beta} \qquad T(1) = d$$

Liv.	**Dim.**	**Costo chiam.**	**N. chiamate**	**Costo livello**
0	b^k	$cb^{k\beta}$	1	$cb^{k\beta}$
1	b^{k-1}	$cb^{(k-1)\beta}$	a	$acb^{(k-1)\beta}$
2	b^{k-2}	$cb^{(k-2)\beta}$	a^2	$a^2cb^{(k-2)\beta}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
i	b^{k-i}	$cb^{(k-i)\beta}$	a^i	$a^icb^{(k-i)\beta}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$k-1$	b	cb^{β}	a^{k-1}	$a^{k-1}cb^{\beta}$
k	1	d	a^k	da^k

Ricorrenze lineari con partizione bilanciata

Liv.	**Dim.**	**Costo chiam.**	**N. chiamate**	**Costo livello**
i	b^{k-i}	$cb^{(k-i)\beta}$	a^i	$a^i cb^{(k-i)\beta}$
k	1	d	a^k	da^k

Sommando i costi totali di tutti i livelli, si ottiene:

$$T(n) = \quad da^k + cb^{k\beta} \sum_{i=0}^{k-1} \frac{a^i}{b^{i\beta}} = \quad da^k + cb^{k\beta} \sum_{i=0}^{k-1} \left(\frac{a}{b^\beta}\right)^i$$

Ricorrenze lineari con partizione bilanciata

$$T(n) = \quad da^k + cb^{k\beta} \sum_{i=0}^{k-1} \frac{a^i}{b^{i\beta}} = \quad da^k + cb^{k\beta} \sum_{i=0}^{k-1} \left(\frac{a}{b^\beta}\right)^i$$

Osservazioni

- $a^k = a^{\log_b n} = a^{\log n / \log b} = 2^{\log a \log n / \log b} = n^{\log a / \log b} = n^\alpha$
- $\alpha = \log a / \log b \Rightarrow \alpha \log b = \log a \Rightarrow \log b^\alpha = \log a \Rightarrow a = b^\alpha$
- Poniamo $q = \frac{a}{b^\beta} = \frac{b^\alpha}{b^\beta} = b^{\alpha-\beta}$

$$T(n) = da^k + cb^{k\beta} \sum_{i=0}^{k-1} \left(\frac{a}{b^\beta}\right)^i = dn^\alpha + cb^{k\beta} \sum_{i=0}^{k-1} q^i$$

Ricorrenze lineari con partizione bilanciata

Caso 1: $\alpha > \beta$

Ne segue che: $q = b^{\alpha-\beta} > 1$:

$$
\begin{aligned}
T(n) &= dn^\alpha + cb^{k\beta} \sum_{i=0}^{k-1} q^i \\
&= n^\alpha d + cb^{k\beta}[(q^k - 1)/(q-1)] && \text{Serie geometrica finita} \\
&\leq n^\alpha d + cb^{k\beta} q^k/(q-1) && \text{Disequazione} \\
&= n^\alpha d + \frac{cb^{k\beta} a^k}{b^{k\beta}}/(q-1) && \text{Sostituzione } q \\
&= n^\alpha d + ca^k/(q-1) && \text{Passi algebrici} \\
&= n^\alpha [d + c/(q-1)] && a^k = n^\alpha, \text{raccolta termini}
\end{aligned}
$$

- Quindi $T(n)$ è $O(n^\alpha)$.
- Per via della componente dn^α, $T(n)$ è anche $\Omega(n^\alpha)$, e quindi $T(n) = \Theta(n^\alpha)$.

Ricorrenze lineari con partizione bilanciata

Caso 2: $\alpha = \beta$

Ne segue che: $q = b^{\alpha-\beta} = 1$:

$$
\begin{aligned}
T(n) &= dn^\alpha + cb^{k\beta} \textstyle\sum_{i=0}^{k-1} q^i \\
&= n^\alpha d + cn^\beta k && q^i = 1^i = 1 \\
&= n^\alpha d + cn^\alpha k && \alpha = \beta \\
&= n^\alpha (d + ck) && \text{Raccolta termini} \\
&= n^\alpha [d + c \log n / \log b] && k = \log_b n
\end{aligned}
$$

e quindi $T(n)$ è $\Theta(n^\alpha \log n)$;

Ricorrenze lineari con partizione bilanciata

Caso 3: $\alpha < \beta$

Ne segue che: $q = b^{\alpha-\beta} < 1$:

$$
\begin{aligned}
T(n) &= dn^\alpha + cb^{k\beta} \sum_{i=0}^{k-1} q^i \\
&= n^\alpha d + cb^{k\beta}[(q^k - 1)/(q-1)] && \text{Serie geometrica finita} \\
&= n^\alpha d + cb^{k\beta}[(1 - q^k)/(1-q)] && \text{Inversione} \\
&\leq n^\alpha d + cb^{k\beta}[1/(1-q)] && \text{Disequazione} \\
&= n^\alpha d + cn^\beta/(1-q) && b^k = n
\end{aligned}
$$

- Quindi $T(n)$ è $O(n^\beta)$.
- Poichè $T(n) = \Omega(n^\beta)$ per il termine non ricorsivo, si ha che $T(n) = \Theta(n^\beta)$.

Ricorrenze lineari con partizione bilanciata (Estesa)

Teorema

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

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

Sono dati tre casi:

(1)	$\exists \epsilon > 0 : f(n) = O(n^{\log_b a - \epsilon})$	$\Rightarrow$	$T(n) = \Theta(n^{\log_b a})$
(2)	$f(n) = \Theta(n^{\log_b a})$	$\Rightarrow$	$T(n) = \Theta(f(n) \log n)$
(3)	$\exists \epsilon > 0 : f(n) = \Omega(n^{\log_b a + \epsilon}) \wedge$ $\exists c : 0 < c < 1, \exists m \geq 0 :$ $af(n/b) \leq cf(n), \forall n \geq m$	$\Rightarrow$	$T(n) = \Theta(f(n))$

Alcuni esempi

Ricorrenza	a	b	$\log_b a$	Caso	Funzione
$T(n) = 9T(n/3) + n \log n$					

Alcuni esempi

Ricorrenza	**a**	**b**	$\mathbf{log_b a}$	**Caso**	**Funzione**
$T(n) = 9T(n/3) + n \log n$	9	3	2	(1)	$T(n) = \Theta(n^2)$

$f(n) = n \log n = O(n^{\log_b a - \epsilon}) = O(n^{2-\epsilon})$, con $\epsilon < 1$

Alcuni esempi

Ricorrenza	**a**	**b**	$\mathbf{log_b a}$	**Caso**	**Funzione**
$T(n) = T(2n/3) + 1$					

Alcuni esempi

Ricorrenza	**a**	**b**	$\mathbf{log_b a}$	**Caso**	**Funzione**
$T(n) = T(2n/3) + 1$	1	$\frac{3}{2}$	0	(2)	$T(n) = \Theta(\log n)$

$$f(n) = n^0 = \Theta(n^{\log_b a}) = \Theta(n^0)$$

Alcuni esempi

Ricorrenza	**a**	**b**	$\log_b a$	**Caso**	**Funzione**
$T(n) = 3T(n/4) + n \log n$					

Alcuni esempi

Ricorrenza	a	b	$\log_b a$	Caso	Funzione
$T(n) = 3T(n/4) + n \log n$	3	4	≈ 0.79	(3)	$T(n) = \Theta(n \log n)$

$f(n) = n \log n = \Omega(n^{\log_4 3+\epsilon})$, con $\epsilon < 1 - \log_4 3 \approx 0.208$

Dobbiamo dimostrare che:
$\exists c \leq 1, \exists m \geq 0 : af(n/b) \leq cf(n), \forall n \geq m$

$$\begin{aligned} af(n/b) &= 3(n/4 \log n/4) \\ &= 3/4n(\log n - \log 4) \\ &\leq 3/4n \log n \\ &\stackrel{?}{\leq} cn \log n \end{aligned}$$

L'ultima disequazione è soddisfatta da $c = 3/4$ e qualsiasi m.

Alcuni esempi

Ricorrenza	**a**	**b**	$\mathbf{log_b a}$	**Caso**	**Funzione**
$T(n) = 2T(n/2) + n \log n$					

Alcuni esempi

Ricorrenza	a	b	$\log_b a$	Caso	Funzione
$T(n) = 2T(n/2) + n \log n$	2	2	1	–	Non applicabile

$f(n) = n \log n \neq O(n^{1-\epsilon})$, con $\epsilon > 0$
$f(n) = n \log n \neq \Theta(n)$
$f(n) = n \log n \neq \Omega(n^{1+\epsilon})$, con $\epsilon > 0$

Nessuno dei tre casi è applicabile e bisogna utilizzare altri metodi.

Interi inferiori/superiori

- I teoremi appena visti non considerano equazioni di ricorrenza con interi inferiori/superiori

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

- In realtà, le equazioni di ricorrenza dovrebbero sempre essere espresse tramite interi inferiori/superiori, perché operano su dimensioni dell'input

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$$

- I risultati dei teoremi valgono anche quando le funzioni sono espresse tramite interi inferiori/superiori

Ricorrenze lineari di ordine costante

Teorema

Siano $a_1, a_2, \ldots, a_h$ costanti intere non negative, con h costante positiva, c e β costanti reali tali che $c > 0$ e $\beta \geq 0$, e sia $T(n)$ definita dalla relazione di ricorrenza:

$$T(n) = \begin{cases} \sum_{1 \leq i \leq h} a_i T(n-i) + cn^\beta & n > m \\ \Theta(1) & n \leq m \leq h \end{cases}$$

Posto $a = \sum_{1 \leq i \leq h} a_i$, allora:

1. $T(n)$ è $\Theta(n^{\beta+1})$, se $a = 1$,
2. $T(n)$ è $\Theta(a^n n^\beta)$, se $a \geq 2$.

Alcuni esempi

	Ricorrenza	a	β	Caso	Funzione
(A)	$T(n) = T(n-10) + n^2$				
(B)	$T(n) = T(n-2) + T(n-1) + 1$				

Alcuni esempi

	Ricorrenza	a	β	Caso	Funzione
(A)	$T(n) = T(n-10) + n^2$	1	2	(1)	$T(n) = \Theta(n^3)$
(B)	$T(n) = T(n-2) + T(n-1) + 1$	2	0	(2)	$T(n) = 2^n$

(A) Poiché $a = 1$, il costo è polinomiale.

(B) Poiché $a = 2$, il costo è esponenziale.

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?

Esercizio – Soluzione

Poichè $\log_2 7$ è ≈ 2.81, il Master Theorem dice che $T(n) = \Theta(n^{\log_2 7})$.

Utilizzando alcune trasformazioni algebriche, si ottiene che:

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

- $a < 16 \Rightarrow \alpha = \log_4 a < 2 \Rightarrow T'(n) = \Theta(n^2) = O(T(n))$
- $a = 16 \Rightarrow \alpha = \log_4 a = 2 \Rightarrow T'(n) = \Theta(n^2 \log n) = O(T(n))$
- $16 < a \leq 49 \Rightarrow \alpha = \log_4 a > 2 \Rightarrow T'(n) = \Theta(n^\alpha) = O(T(n))$
- $a < 49 \Rightarrow \alpha = \log_4 a > 2 \Rightarrow T'(n) = \Theta(n^\alpha) = \Omega(T(n))$

Algoritmi e strutture dati

Analisi di funzioni
Back to algorithms!

Alberto Montresor and Davide Rossi

Università di Bologna

21 ottobre 2024

Sommario

Complessità della Versione 1

```
int maxsum1(int[] A, int n) {
  int maxSoFar = 0;
  for (int i=0; i < n; i++) {
    for (int j=i; j < n; j++) {
      int sum = 0;
      for (int k=i; k <= j; k++) {
        sum = sum + A[k];
    }
    maxSoFar = max(maxSoFar, sum);
    }
  }
  return maxSoFar;
}
```

La complessità dell'algoritmo può essere approssimata come segue (contando il numero di esecuzioni della riga più interna)

$$T(n) = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j - i + 1)$$

Complessità della Versione 1 - $O(n^3)$

Vogliamo provare che $T(n) = O(n^3)$, i.e.

$$\exists c_2 > 0, \exists m \geq 0 : T(n) \leq c_2 n^3, \forall n \geq m$$

$$\begin{aligned} T(n) &= \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j - i + 1) \\ &\leq \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} n \leq \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n \\ &= \sum_{i=0}^{n-1} n^2 = n^3 \leq c_2 n^3 \end{aligned}$$

Questa disequazione è vera per $n \geq m = 0$ and $c_2 \geq 1$.

Complessità della Versione 1 - $\Omega(n^3)$

Vogliamo provare che $T(n) = \Omega(n^3)$, i.e.

$$\exists c_1 > 0, \exists m \geq 0 : T(n) \geq c_1 n^3, \forall n \geq m$$

$$\begin{aligned}
T(n) &= \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j - i + 1) \\
&\geq \sum_{i=0}^{n/2} \sum_{j=i}^{i+n/2-1} (j - i + 1) \\
&= \sum_{i=0}^{n/2} \sum_{j=i}^{i+n/2-1} n/2 \\
&= \sum_{i=0}^{n/2} n^2/4 \geq n^3/8 \geq c_1 n^3
\end{aligned}$$

L'ultima disequazione è vera per $n \geq m = 0$ and $c_1 \leq 8$.

Complessità della versione 2

```
int maxsum2(int[] A, int n) {
  int maxSoFar = 0;
  for (int i=0; i < n; i++) {
    int sum = 0;
    for (int j=i; j < n; j++) {
      sum = sum + A[j];
      maxSoFar = max(maxSoFar, sum);
    }
  }
  return maxSoFar;
}
```

La complessità di questo algoritmo può essere approssimata come segue (stiamo contando il numero di passi nel ciclo più interno)

$$T(n) = \sum_{i=0}^{n-1} n - i$$

Complessità della versione 2 - $\theta(n^2)$

Vogliamo provare che $T(n) = \theta(n^2)$.

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{n-1} n - i \\
&= \sum_{i=1}^{n} i \\
&= \frac{n(n+1)}{2} = \Theta(n^2)
\end{aligned}
$$

Questo non richiede ulteriori dimostrazioni

Complessità della versione 3

```
int maxsum_rec(int[] A, int i, int j) {
  if (i==j)
    return max(0, A[i]);
  int m = (i+j) / 2;
  int maxs = maxsum_rec(A, i, m);
  int maxd = maxsum_rec(A, m+1, j);
  int maxss = 0;
  int sum = 0;
  for (int k=m; k>=i; k--) {
    sum = sum+A[k];
    maxss = max(maxss, sum);
  }
  int maxdd = 0;
  sum = 0;
  for (int k=m+1; k<=j; k++) {
    sum = sum+A[k];
    maxdd = max(maxdd, sum);
  }
  return max(max(maxs,maxd),maxss+maxdd);
}
```

Per questo, definiamo la equazione di ricorrenza:

Complessità della versione 3

```
int maxsum_rec(int[] A, int i, int j) {
  if (i==j)
    return max(0, A[i]);
  int m = (i+j) / 2;
  int maxs = maxsum_rec(A, i, m);
  int maxd = maxsum_rec(A, m+1, j);
  int maxss = 0;
  int sum = 0;
  for (int k=m; k>=i; k--) {
    sum = sum+A[k];
    maxss = max(maxss, sum);
  }
  int maxdd = 0;
  sum = 0;
  for (int k=m+1; k<=j; k++) {
    sum = sum+A[k];
    maxdd = max(maxdd, sum);
  }
  return max(max(maxs,maxd),maxss+maxdd);
}
```

Per questo, definiamo la equazione di ricorrenza:

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

Utilizzando il teorema, possiamo vedere che $\alpha = \log_2 2 = 1$ e $\beta = 1$, quindi $T(n) = \Theta(n \log n)$.

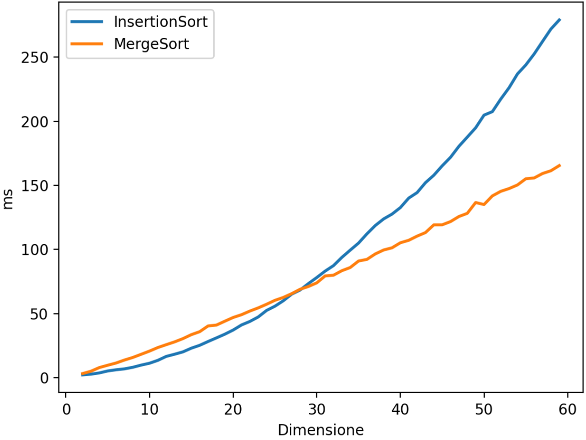
Complessità della versione 4

```
int maxsum4(int A[], int n) {
  int maxSoFar = 0;
  int maxHere = 0;
  for (int i=0; i < n; i++) {
    maxHere = max(maxHere+A[i],0);
    maxSoFar = max(maxSoFar,maxHere);
  }
  return maxSoFar;
}
```

E' facile vedere che la complessità di questa versione è $\theta(n)$.

Fattori moltiplicativi

A volte, i fattori moltiplicativi di una funzione di complessità sono talmente alti che se ne sconsiglia l'uso per piccoli valori di n



GNU Multiple Precision Arithmetic Library

- Utilizzata da Mathematica, Maple, etc.
- Le moltiplicazioni vengono realizzate utilizzando algoritmi diversi, mano a mano che n cresce.
- https://gmplib.org/manual/Multiplication-Algorithms.html

15.1 Multiplication

NxN limb multiplications and squares are done using one of seven algorithms, as the size N increases.

Algorithm	Threshold
Basecase	(none)
Karatsuba	`MUL_TOOM22_THRESHOLD`
Toom-3	`MUL_TOOM33_THRESHOLD`
Toom-4	`MUL_TOOM44_THRESHOLD`
Toom-6.5	`MUL_TOOM6H_THRESHOLD`
Toom-8.5	`MUL_TOOM8H_THRESHOLD`
FFT	`MUL_FFT_THRESHOLD`

GNU Multiple Precision Arithmetic Library

- Utilizzata da Mathematica, Maple, etc.
- I limiti (threshold) dipendono dall'architettura

host type	abi	host name	meas thres	conf thres	cfg file
z10-ibm-linux-gnu	64	lgentoo4.s390.gentoo.wh0rd.org-stat	1728	1728	s390_64/z10/gmp-mparam.h
atom-unknown-linux-gnu	64	gege.gmplib.org-stat	2240	2240	x86_64/atom/gmp-mparam.h
z10esa-ibm-linux-gnu	32	lgentoo3.s390.gentoo.wh0rd.org-stat	2240	2240	s390_32/esame/gmp-mparam.h
power7-unknown-linux-gnu	mode32	gcc1-power7.osuosl.org-stat	2688	2688	powerpc64/mode32/p4/gmp-mparam.h
bulldozer-unknown-freebsd8.3	64	oshell.gmplib.org-stat	3520	3712	x86_64/bd1/gmp-mparam.h
piledriver-unknown-netbsd6.1.3	64	pilenbsd64v61.gmplib.org-stat	3712	3712	x86_64/bd2/gmp-mparam.h
powerpc7447-unknown-linux-gnu	32	spigg.gmplib.org-stat	3712	3712	powerpc32/gmp-mparam.h
coreihwl-unknown-netbsd6.1.2	64	hannahnbsd64v61.gmplib.org-stat	4224	4224	x86_64/coreihwl/gmp-mparam.h
coreinhm-unknown-netbsd6.1.3	64	bikonbsd64v61.gmplib.org-stat	4224	4032	x86_64/coreinhm/gmp-mparam.h
power7-ibm-aix7.1.0.0	mode64	power-aix.fsffrance.org-stat	4288	4288	powerpc64/mode64/p7/gmp-mparam.h
atom-unknown-linux-gnu	32	gege.gmplib.org-stat	4544	4544	x86/atom/gmp-mparam.h
core2-unknown-netbsd6.1.4	64	repentiumnbsd64v61.gmplib.org-stat	4736	4736	x86_64/core2/gmp-mparam.h
coreisbr-apple-darwin12.5.0	64	poire.loria.fr-stat	4736	4736	x86_64/coreisbr/gmp-mparam.h
coreiwsm-unknown-linux-gnu	64	gcc20.fsffrance.org-stat	4736	4032	x86_64/coreinhm/gmp-mparam.h
power7-unknown-linux-gnu	mode64	gcc1-power7.osuosl.org-stat	4736	4288	powerpc64/mode64/p7/gmp-mparam.h
powerpc970-apple-darwin8.11.0	mode32	g5.gmplib.org-stat	4736	2688	powerpc64/mode32/p4/gmp-mparam.h
power7-ibm-aix7.1.0.0	32	power-aix.fsffrance.org-stat	5312	5312	powerpc32/p7/gmp-mparam.h
bobcat-unknown-netbsd6.1.3	64	bobcat.gmplib.org-stat	5504	5504	x86_64/bobcat/gmp-mparam.h
alphaev6-unknown-linux-gnu	standard	agnesi.math.su.se-stat	5760	5760	alpha/ev6/gmp-mparam.h
armcortexa15neon-unknown-linux-	standard	parma.gmplib.org-stat	5760	5760	arm/v7a/cora15/gmp-mparam.h
power7-unknown-linux-gnu	32	gcc1-power7.osuosl.org-stat	5760	5312	powerpc32/p7/gmp-mparam.h
core2-unknown-netbsdelf6.1.4	32	repentiumnbsd32v61.gmplib.org-stat	6784	6784	x86/core2/gmp-mparam.h
coreinhm-unknown-netbsdelf6.1.3	32	bikonbsd32v61.gmplib.org-stat	6784	6784	x86/coreinhm/gmp-mparam.h