

Algoritmi e Strutture Dati

Analisi di algoritmi ricorsivi:
Il Teorema master (*)

Punto della situazione

- Abbiamo imparato i concetti fondamentali di **complessità (temporale o spaziale) di un algoritmo**, e quelli di delimitazione superiore (*upper bound*) e inferiore (*lower bound*) alla **complessità (temporale o spaziale) di un problema**
- Ad esempio, l'algoritmo di **ricerca sequenziale** di un elemento in un **insieme non ordinato** di n elementi ha complessità temporale $T(n) = O(n)$, in quanto su **alcune** istanze costa $\Theta(n)$, mentre su altre costa $o(n)$. Ne consegue che l'*upper bound* al problema della ricerca di un elemento in un insieme non ordinato di n elementi è pari a $O(n)$
- Invece, il *lower bound* temporale del problema della ricerca di un elemento in un **insieme non ordinato** di n elementi è pari a $\Omega(n)$: infatti, **ogni** algoritmo di risoluzione deve per forza di cose guardare tutti gli elementi dell'insieme per decidere se l'elemento cercato appartiene o meno ad esso! Quindi, l'algoritmo di **ricerca sequenziale** è **ottimo!**
- Invece, l'algoritmo di **ricerca binaria** di un elemento in un **insieme ordinato** di n elementi ha complessità temporale $T(n) = O(\log n)$. Vedremo più avanti che anche questo algoritmo è ottimo.

Ricerca binaria in forma ricorsiva

L'algoritmo di ricerca binaria può essere riscritto ricorsivamente come:

algoritmo ricercaBinariaRic(*array* L , *elemento* x) \rightarrow *booleano*

1. $n \leftarrow$ lunghezza di L
2. **if** ($n = 0$) **then return** non trovato
3. $i \leftarrow \lceil n/2 \rceil$
4. **if** ($L[i] = x$) **then return** trovato
5. **else if** ($L[i] > x$) **then return** ricercaBinariaRic($L[1; i - 1], x$)
6. **else return** ricercaBinariaRic($L[i + 1; n], x$)

Come analizzarlo?

Equazioni di ricorrenza

Il tempo di esecuzione dell'algoritmo può essere descritto tramite l'equazione di ricorrenza:

$$T(n) \leq \begin{cases} \Theta(1) + T(\lceil (n-1)/2 \rceil) & \text{se } n \geq 1 \\ \Theta(1) & \text{se } n = 0 \end{cases}$$

dove $\Theta(1)$ è il costo (costante) che viene speso all'interno di ogni chiamata ricorsiva (si noti che utilizzo il simbolo \leq perché l'algoritmo può terminare in una qualsiasi chiamata ricorsiva).

Mostriamo tre metodi per risolvere equazioni di ricorrenza: **iterazione**, **sostituzione**, e **teorema Master**

Metodo dell'iterazione

Idea: “srotolare” la ricorsione, ottenendo una sommatoria dipendente solo dalla dimensione n del problema iniziale (già visto per Fibonacci)

Nel caso della ricerca binaria, semplificando leggermente la relazione di ricorrenza a $T(n) \leq \Theta(1) + T(n/2)$:

$$\begin{aligned} T(n) &\leq \Theta(1) + T(n/2) \leq \Theta(1) + (\Theta(1) + T(n/4)) \leq \\ \rightarrow &\leq \Theta(1) + (\Theta(1) + (\Theta(1) + T(n/8))) \leq \dots \\ &\leq \left(\sum_{j=1}^i \Theta(1) \right) + T(n/2^i) = i \cdot \Theta(1) + T(n/2^i) \end{aligned}$$

Per $i = \lfloor \log n \rfloor + 1$: $T(n) \leq \Theta(1) \cdot \Theta(\log n) + T(0) = \Theta(\log n)$

Esercizi di approfondimento

Risolvere usando il metodo dell'iterazione le seguenti equazioni di ricorrenza:

- $T(n) = n + T(n-1), \quad T(1)=1;$
- $T(n) = 9 T(n/3) + n, \quad T(1)=1;$
(soluzione sul libro di testo: Esempio 2.4)

Metodo della sostituzione

Idea: “**indovinare**” una soluzione, ed usare l’induzione matematica per provare che la soluzione dell’equazione di ricorrenza è effettivamente quella intuita

Esempio: $T(n) = n + T(n/2)$, $T(1)=1$

Ipotizziamo che la soluzione sia $T(n) \leq c \cdot n$ per una costante c opportuna, e verifichiamolo:

- **Passo base:** $T(1)=1 \leq c \cdot 1$ per ogni $c \geq 1$ OK
- **Passo induttivo:** $T(n) = n + T(n/2) \leq n + c \cdot (n/2) = (c/2 + 1)n$
Ma $(c/2 + 1)n \leq c n$ per $c \geq 2$, quindi $T(n) \leq c \cdot n$ per $c \geq 2$

Teorema Master

Permette di analizzare algoritmi basati sulla tecnica del *divide et impera*:

- **dividi** il problema (di dimensione iniziale n) in $a \geq 1$ sottoproblemi di dimensione n/b , $b > 1$
- risolvi i sottoproblemi **ricorsivamente**
- **ricombina** le soluzioni

Sia $f(n)$ il tempo speso nella chiamata ricorsiva (escluse le sottochiamate ricorsive), incluso quindi il tempo per dividere e ricombinare istanze di dimensione n . La relazione di ricorrenza è data da:

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

Esempio: algoritmo fibonacci6

algoritmo fibonacci6(*intero* n) \rightarrow *intero*

1. $A \leftarrow \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
2. $M \leftarrow \text{potenzaDiMatrice}(A, n - 1)$
3. **return** $M[1][1]$

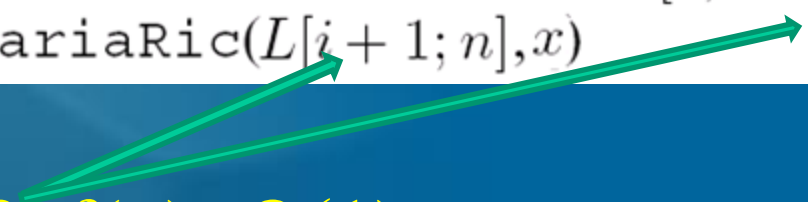
funzione potenzaDiMatrice(*matrice* A , *intero* k) \rightarrow *matrice*

4. **if** ($k \leq 1$) **then** $M \leftarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
5. **else** $M \leftarrow \text{potenzaDiMatrice}(A, \lfloor k/2 \rfloor)$
6. $M \leftarrow M \cdot M$
7. **if** (k è dispari) **then** $M \leftarrow M \cdot A$
8. **return** M


$$a=1, b=2, f(n)=\Theta(1)$$

Esempio: algoritmo di ricerca binaria

algoritmo ricercaBinariaRic(*array* L , *elemento* x) \rightarrow *booleano*

1. $n \leftarrow$ lunghezza di L
 2. **if** ($n = 0$) **then return** non trovato
 3. $i \leftarrow \lceil n/2 \rceil$
 4. **if** ($L[i] = x$) **then return** trovato
 5. **else if** ($L[i] > x$) **then return** ricercaBinariaRic($L[1; i - 1], x$)
 6. **else return** ricercaBinariaRic($L[i + 1; n], x$)
- 

$$a=1, b=2, f(n)=\Theta(1)$$

Algoritmo fibonacci2

```
algoritmo fibonacci2(intero n) → intero  
if (n ≤ 2) then return 1  
else return fibonacci2(n-1) +  
                fibonacci2(n-2)
```

Ricorsivo? Sì

Divide et impera?

No, perché la dimensione dei sottoproblemi è $\Theta(n)$, e non $\Theta(n/b)$ con $b > 1$

Teorema Master (o teorema principale)

La relazione di ricorrenza:

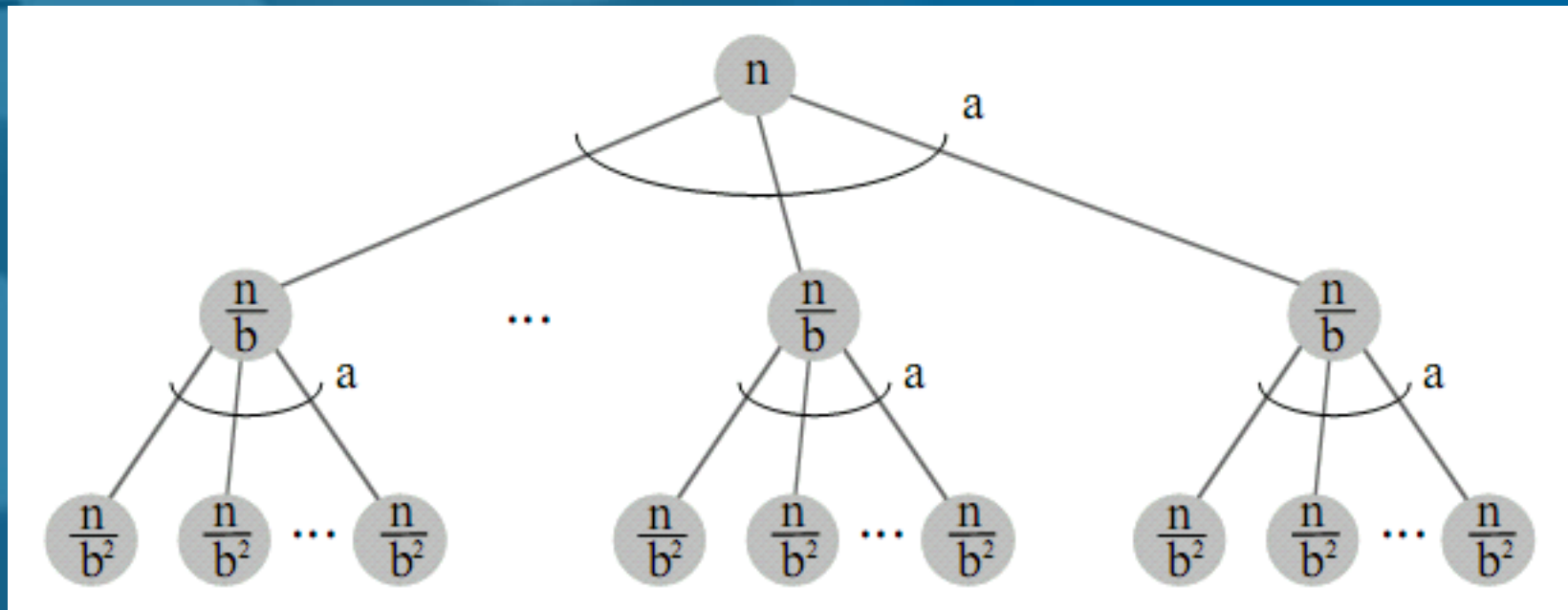
$$T(n) = \begin{cases} aT(n/b) + f(n) & \text{se } n > 1 \\ \Theta(1) & \text{se } n = 1 \end{cases}$$

ha soluzione:

1. $T(n) = \Theta(n^{\log_b a})$ se $f(n) = O(n^{\log_b a - \varepsilon})$ per qualche $\varepsilon > 0$
2. $T(n) = \Theta(n^{\log_b a} \log n)$ se $f(n) = \Theta(n^{\log_b a})$
3. $T(n) = \Theta(f(n))$ se $f(n) = \Omega(n^{\log_b a + \varepsilon})$ per qualche $\varepsilon > 0$
(ma sotto l'ulteriore ipotesi che $f(n)$ soddisfi la *condizione di regolarità*:
a $f(n/b) \leq c f(n)$ per qualche $c < 1$ ed n sufficientemente grande)

Dimostrazione del caso 1

Albero della ricorsione



Proprietà dell'albero della ricorsione

- **Proprietà 1:** il numero di nodi a livello i dell'albero della ricorsione è a^i (ricorda che la radice è a livello 0)
- **Proprietà 2:** i sottoproblemi a livello i dell'albero della ricorsione hanno dimensione n/b^i
- **Proprietà 3:** il contributo al tempo di esecuzione di un nodo a livello i (escluso tempo sottochiamate ricorsive, che vengono conteggiate esplicitamente) è $f(n/b^i)$
- **Proprietà 4:** il numero di livelli dell'albero è (circa) $\log_b n$

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

...quindi, nel caso 1

$$f(n) = O(n^{\log_b a - \varepsilon})$$

$$\begin{aligned} \Rightarrow a^i f(n/b^i) &= O(a^i (n/b^i)^{\log_b a - \varepsilon}) = \\ &= O(n^{\log_b a - \varepsilon} (a b^\varepsilon / b^{\log_b a})^i) = \\ &= O(n^{\log_b a - \varepsilon} (b^\varepsilon)^i) \end{aligned}$$

$$\begin{aligned} T(n) &= \sum_{i=0, \dots, \log_b n} O(n^{\log_b a - \varepsilon} (b^\varepsilon)^i) = O(n^{\log_b a - \varepsilon} \sum_{i=0, \dots, \log_b n} (b^\varepsilon)^i) \\ &= O\left(n^{\log_b a - \varepsilon} \left(\frac{b^{\varepsilon(\log_b n + 1)} - 1}{b^\varepsilon - 1}\right)\right) = O\left(n^{\log_b a - \varepsilon} \left(\frac{b^\varepsilon n^\varepsilon - 1}{b^\varepsilon - 1}\right)\right) \\ &= O(n^{\log_b a - \varepsilon} (n^\varepsilon)) = O(n^{\log_b a}) \end{aligned}$$

Inoltre, $T(n) \geq a^{\log_b n}$ (ultimo termine sommatoria slide precedente)
 $= b^{\log_b(a \log_b n)} = b^{\log_b n \cdot \log_b a} = n^{\log_b a}$ cioè $T(n) = \Omega(n^{\log_b a})$ da cui la tesi.

I casi 2 e 3 possono essere mostrati in modo analogo.

QED

Esempi (per tutti assumiamo $T(1)=\Theta(1)$)

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

$a=2, b=2, f(n)=n=\Theta(n^{\log_2 2}) \Rightarrow$ caso 2 TM

$\Rightarrow T(n)=\Theta(n^{\log_2 2} \log n) = \Theta(n \log n)$

2) $T(n) = 3T(n/9) + 7$

$a=3, b=9, f(n)=7=O(n^{\log_9 3 - \varepsilon}) \Rightarrow$ caso 1 TM

$\Rightarrow T(n) = \Theta(n^{\log_9 3}) = \Theta(\sqrt{n})$

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

$a=3, b=9, f(n)=n=\Omega(n^{\log_9 3 + \varepsilon})$

inoltre $3(n/9) \leq c n$ per $c=1/3$

\Rightarrow caso 3 del TM $\Rightarrow T(n)=\Theta(n)$

Esempi

$$4) T(n) = 2T(n/2) + n \log n$$

$a=2$, $b=2$ e quindi ovviamente non ricade nel caso 1 perché

$$f(n) = n \log n \neq O(n^{\log_2 2 - \varepsilon}) \equiv O(n^{1 - \varepsilon})$$

inoltre $f(n) \neq \Theta(n^{\log_2 2})$, e quindi non ricade nel caso 2,

infine non esiste alcun $\varepsilon > 0$ per cui $f(n) = \Omega(n^{\log_2 2 + \varepsilon}) \equiv \Omega(n^{1 + \varepsilon})$

$$\text{(infatti, } \lim_{n \rightarrow \infty} \frac{n \log n}{n^{1 + \varepsilon}} = \frac{\log n}{n^\varepsilon} = 0 \text{ per ogni } \varepsilon > 0 \text{)}$$

non rientra nemmeno nel caso 3 e quindi
non si può applicare il teorema Master!

Esempi

$$5) T(n) = 3T(n/3) + n/\log^4 n$$

$a=3$, $b=3$ e quindi non ricade nel caso 2 perché

$$f(n) = n/\log^4 n \neq \Theta(n^{\log_3 3}) \equiv O(n)$$

inoltre ovviamente non rientra nel caso 3 perché
 $f(n) \neq \Omega(n^{1+\varepsilon})$

infine non esiste alcun $\varepsilon > 0$ per cui $f(n) = O(n^{\log_3 3 - \varepsilon}) \equiv O(n^{1-\varepsilon})$

(infatti, $\lim_{n \rightarrow \infty} \frac{n/\log^4 n}{n^{1-\varepsilon}} = \frac{n^\varepsilon}{\log^4 n} = \infty$ per ogni $\varepsilon > 0$)

non rientra nemmeno nel caso 1 e quindi
 non si può applicare il teorema Master!