Esercizi sulle equazioni ricorrenti

Roberto Cordone

Esercizio 1 Scrivere l'equazione ricorrente che descrive la complessità computazionale del seguente algoritmo per calcolare l'n-esimo numero di Fibonacci.

Fibonacci(n)

if $n = 0$ **or** $n = 1$

then Return 1;

else Return Fibonacci$(n-1)$ + Fibonacci$(n-2)$

Dimostrare per sostituzione che l'algoritmo ha complessità in $\Omega\left(\left(\frac{3}{2}\right)^n\right)$.
Dire se è possibile risolvere l'equazione con il metodo principale.

Soluzione L'equazione è banalmente

$$T(n) = \begin{cases} 1 & \text{per } n \leq 1 \\ T(n-1) + T(n-2) & \text{per } n \geq 2 \end{cases}$$

Il metodo principale non si può applicare, perché l'equazione non esprime $T(n)$ in funzione di $T(n/b)$, cioè di T valutata in una frazione fissa di n, bensì nei due punti che precedono immediatamente n.

Vogliamo dimostrare

$$\exists n_0 \in \mathbb{N}, c > 0 : T(n) \geq c\left(\frac{3}{2}\right)^n \quad \text{per ogni } n \geq n_0$$

Scomponiamo questa tesi in due:

1. caso base
$$\exists n_0 \in \mathbb{N}, c > 0 : T(n_0) \geq c\left(\frac{3}{2}\right)^{n_0}$$
2. passo induttivo
$$\exists n_0 \in \mathbb{N}, c > 0 : \text{se } T(i) \geq c\left(\frac{3}{2}\right)^i \quad \text{per } i = n_0, \ldots, n-1 \text{ allora } T(n) \geq c\left(\frac{3}{2}\right)^n$$

Possiamo scegliere arbitrariamente n_0. Per semplicità, proviamo a fissare $n_0 = 1$.

$$T(n_0) = 1 \geq c\left(\frac{3}{2}\right)^1$$

Per raggiungere questa conclusione, basta porre $c \leq 2/3$.

Ora possiamo assumere, oltre all'espressione di $T(n)$ data dall'equazione ricorrente, che $T(i) \geq c\left(\frac{3}{2}\right)^i$ per ogni i compresa fra $n_0 = 1$ e $n-1$. Dobbiamo dimostrare che questo è vero anche per $i = n$.

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

I punti $i = n-1$ e $i = n-2$ ricadono nell'ipotesi, per cui

$$\begin{cases} T(n-1) \geq c\left(\frac{3}{2}\right)^{n-1} \\ T(n-2) \geq c\left(\frac{3}{2}\right)^{n-2} \end{cases} \Rightarrow T(n) \geq c\left(\frac{3}{2}\right)^{n-1} + c\left(\frac{3}{2}\right)^{n-2} =$$

$$= c\left(\frac{3}{2}+1\right)\left(\frac{3}{2}\right)^{n-2} = \frac{5}{2}c\left(\frac{2}{3}\right)^2\left(\frac{3}{2}\right)^n = \frac{10}{9}c\left(\frac{3}{2}\right)^n$$

Quindi è vero che $T(n) \geq c\,(3/2)^n$, come si voleva dimostrare.

In conclusione, i valori $n_0 = 1$ e $c = 2/3$ permettono di dimostrare la tesi.

Esercizio 2 Dimostrare per sostituzione che l'equazione ricorrente

$$T(n) = \begin{cases} 1 & \text{per } n = 1 \\ T(n-1) + T(n-2) + \ldots + T(1) & \text{per } n > 1 \end{cases}$$

ha soluzione in $O(2^n)$.

Soluzione La tesi da dimostrare è

$$\exists n_0 \in \mathbb{N}, c > 0 : T(n) \leq c2^n$$

Al solito, la scomponiamo in due tesi ausiliarie

1. caso base
$$\exists n_0 \in \mathbb{N}, c > 0 : T(n_0) \leq c2^1$$

2. passo induttivo

$$\exists n_0 \in \mathbb{N}, c > 0 : \text{se } T(i) \leq c2^i \quad \text{per } i = n_0, \ldots, n-1 \text{ allora } T(n) \leq c2^n$$

Per semplicità, proviamo con $n_0 = 1$. Dobbiamo dimostrare che

$$T(1) = 1 \leq c2^1$$

per cui basta imporre $c \geq 1/2$.

Passando alla seconda tesi, tutti i punti che compaiono al secondo membro dell'equazione ricorrente soddisfano l'ipotesi di essere compresi fra n_0 e $n-1$. Quindi

$$T(n) = T(n-1) + T(n-2) + \ldots + T(1) \Rightarrow$$

$$\Rightarrow T(n) \leq c2^{n-1} + c2^{n-2} + \ldots + c2^0 = c\sum_{h=0}^{n-1} 2^h = c\frac{2^n - 1}{2 - 1}$$

che è $< c2^n$, come volevasi dimostrare.

Esercizio 3 Risolvere approssimativamente l'equazione ricorrente che segue.

$$T(n) = \begin{cases} 0 & \text{per } n = 2 \\ 2T(n/2) + 2 & \text{per } n > 2 \end{cases}$$

Quindi, determinare la soluzione esatta, ispirandosi alla soluzione asintotica per ipotizzare un'espressione parametrica, i cui parametri andranno determinati con il metodo di sostituzione.

Soluzione Per determinare un'espressione approssimata, applichiamo il metodo iterativo.

$$\begin{aligned}
T(n) = 2T(n/2) + 2 &\Rightarrow T(n) = 2T(n/2) + 2 \\
T(n/2) = 2T(n/4) + 2 &\Rightarrow T(n) = 2(2T(n/4) + 2) + 2 = 4T(n/4) + 4 + 2 \\
T(n/4) = 2T(n/8) + 2 &\Rightarrow T(n) = 4(2T(n/8) + 2) + 4 + 2 = 8T(n/8) + 8 + 4 + 2 \\
\ldots = \ldots &\Rightarrow T(n) = 2^h T\left(n/2^h\right) + \sum_{i=1}^{h} 2^i
\end{aligned}$$

Vogliamo interrompere lo sviluppo al livello h tale che $n/2^h = 2$ (si badi bene, non 1!) Quindi $h = \log_2(n/2) = \log_2 n - 1$. Da cui

$$T(n) = 2^{\log_2 n - 1} T\left(n/2^h\right) + \sum_{i=1}^{\log_2 n - 1} 2^i = \frac{n}{2} T(2) + \sum_{i=0}^{\log_2 n - 1} 2^i - 2^0 =$$

$$= \frac{n}{2} \cdot 0 + \frac{2^{\log_2 n} - 1}{2 - 1} - 1 = n - 2$$

Poiché siamo stati molto attenti a sviluppare sommatorie e termini vari nel modo più esatto, possiamo attenderci di aver trovato l'espressione corretta. In effetti, se ipotizziamo che l'espressione abbia una forma lineare

$$G(n) = an + b$$

per determinare i coefficienti a e b occorre soddisfare la seguente uguaglianza

$$T(n) = G(n) \text{ per ogni } n$$

Possiamo scomporre questa uguaglianza in due:

1. caso base
$$T(2) = G(2)$$
2. passo induttivo
$$\text{Se } T(i) = G(i) \text{ per ogni } i = 2, \ldots, n-1 \text{ allora } T(n) = G(n)$$

Per dimostrare la prima tesi occorre

$$\begin{cases} T(2) = 0 \\ G(2) = 2a + b \end{cases} \Rightarrow 0 = 2a + b$$

Osservando che $n/2 \leq n - 1$, è $T(n/2) = G(n/2))$

$$T(n) = 2T(n/2) + 2 = 2G(n/2) + 2 = 2(an/2 + b) + 2 = an + 2b + 2$$

Per dimostrare la seconda tesi occorre quindi

$$an + 2b + 2 = an + b \text{ per ogni } n \Rightarrow \begin{cases} a = a \\ 2b + 2 = b \end{cases} \Rightarrow b = -2$$

Poiché $2a + b = 0$, è $a = 1$. Concludendo $T(n) = n - 2$.

Esercizio 4 Dimostrare con tutti i metodi esistenti che l'equazione ricorrente

$$T(n) = \begin{cases} 1 & \text{per } n = 1 \\ 3T\left(\frac{n}{2}\right) + n^3 & \text{per } n > 1 \end{cases}$$

ha soluzione in $\Theta(n^3)$.

Soluzione Cominciamo con il metodo principale. Si può applicare perché il valore di T in n dipende dal suo valore in $n/2$. Ora si tratta di confrontare la complessità della parte divide e combina ($f(n) = n^3$) con $n^{\log_b a}$. Nel caso presente, $n^{\log_b a} = n^{\log_2 3}$. Quindi, $f(n) = n^3 \in \Omega\left(n^{\epsilon+\log_2 3}\right)$ con $\epsilon = 3 - \log_2 3 > 0$. È verificata la prima condizione per trovarsi nel terzo caso. Quindi la parte di divide e combina prevale su quella di soluzione dei casi base. Verifichiamo la seconda condizione:

$$\exists c \in (0;1) : af\left(\frac{n}{b}\right) \leq cf(n)$$

È $af(n/b) = 3(n/2)^3 n^3 (3/8)$. Basta porre $c = 3/8$ per garantire che $af(n/b) \leq cf(n)$. Ricadiamo quindi nel terzo caso, e concludiamo che il divide e combina della radice prevale su tutto il resto dell'algoritmo: $T(n) = f(n) = n^3$.

Passiamo al metodo iterativo

$$\begin{aligned}
T(n) = 3T\left(\frac{n}{2}\right) + n^3 \quad &\Rightarrow T(n) = 3T\left(\frac{n}{2}\right) + n^3 \\
T(n/2) = 3T\left(\frac{n}{4}\right) + \frac{n^3}{2^3} \quad &\Rightarrow T(n) = 3\left(3T\left(\frac{n}{4}\right) + \frac{n^3}{2^3}\right) + n^3 = \\
&\qquad = 9T\left(\frac{n}{4}\right) + 3\frac{n^3}{2^3} + n^3 \\
T(n/4) = 3T\left(\frac{n}{8}\right) + \frac{n^3}{4^3} \quad &\Rightarrow T(n) = 9\left(3T\left(\frac{n}{8}\right) + \frac{n^3}{4^3}\right) + 3\frac{n^3}{2^3} + n^3 = \\
&\qquad = 27T\left(\frac{n}{4}\right) + 9\frac{n^3}{4^3} + 3\frac{n^3}{2^3} + n^3 \\
\ldots = \ldots \quad &\Rightarrow T(n) = 3^h T\left(n/2^h\right) + \sum_{i=1}^{h-1} \frac{3^i}{8^i} n^3
\end{aligned}$$

Il numero di livelli h deve essere tale da portarci al caso base, vale a dire da

rendere $n/2^h = 1$, ovvero $h = \log_2 n$. Quindi

$$T(n) = 3^{\log_2 n} T\left(n/2^{\log_2 n}\right) + \sum_{i=1}^{\log_2 n - 1} \frac{3^i}{8^i} n^3 = n^{\log_2 3} T(1) + n^3 \sum_{i=1}^{\log_2 n - 1} \left(\frac{3}{8}\right)^i =$$

$$= n^{\log_2 3} + n^3 \left(\frac{1 - \left(\frac{3}{8}\right)^{\log_2 n}}{1 - \frac{3}{8}} - 1 \right) = n^{\log_2 3} + \frac{3}{5} n^3 - \frac{8}{5} n^{3 + \log_2 (3/8)} = \frac{3}{5} \left(n^3 - n^{\log_2 3}\right)$$

Infine, impieghiamo il metodo di sostituzione per confermare che $T(n) \in \Theta(n^3)$.

$$\exists n_0 \in \mathbb{N}, c_1 > 0, c_2 > 0 : c_1 n^3 \leq T(n) \leq c_2 n^3 \text{ per ogni } n \geq n_0$$

Come al solito, scomponiamo la tesi in due parti:

1. caso base
$$c_1 1^3 \leq T(1) \leq c_2 1^3$$

2. passo induttivo

   Se $c_1 i^3 \leq T(i) \leq c_2 i^3$ per ogni $i = 2, \ldots, n-1$ allora $c_1 n^3 \leq T(n) \leq c_2 n^3$

La prima tesi è valida perché

$$c_1 1^3 \leq 1 \leq c_2 1^3$$

Basta porre $c_1 \leq 1$ e $c_2 \geq 1$.

Osserviamo poi che $n/2 \leq n - 1$. Quindi

$$c_1 \left(\frac{n}{2}\right)^3 \leq T\left(\frac{n}{2}\right) \leq c_2 \left(\frac{n}{2}\right)^3$$

da cui (essendo $T(n) = 3T\left(\frac{n}{2}\right) + n^3$)

$$3c_1 \left(\frac{n}{2}\right)^3 + n^3 \leq T(n) \leq 3c_2 \left(\frac{n}{2}\right)^3 + n^3$$

$$\left(\frac{3}{8} c_1 + 1\right) n^3 \leq T(n) \leq \left(\frac{3}{8} c_2 + 1\right) n^3$$

Per giungere alla tesi, è sufficiente fissare c_1 e c_2 in modo che il membro di sinistra sia $\geq c_1 n^3$ e quello di destra $\leq c_2 n^3$.

$$\begin{cases} \frac{3}{8}c_1 + 1 \geq c_1 \\ \frac{3}{8}c_2 + 1 \leq c_2 \end{cases} \Rightarrow \begin{cases} c_1 \leq \frac{8}{5} \\ c_2 \geq \frac{8}{5} \end{cases}$$

Ricordandosi dei vincoli già imposti ($c_1 \leq 1$ e $c_2 \geq 1$), concludiamo che basta

$$c_1 \leq 1 \qquad c_2 \geq \frac{8}{5}$$

Esercizio 5 Valutare la complessità asintotica del seguente algoritmo:

Algo(n)

If $n \leq 2$ Return 1;

For $i := 1$ to n do

    $m := n$;
    While $m \geq 1$ do
        $m := m/2$;

Return 2∗ Algo$(n/4)$ + Algo$(n/4)$;

scrivendo e risolvendo la relativa equazione ricorrente.

Si noti bene che non si chiede di determinare il valore restituito dall'algoritmo, ma la sua complessità!

Proporre una piccola modifica che ne migliori la complessità.

Soluzione Nel caso $n \leq 2$, l'algoritmo si compone di una sola istruzione. Altrimenti, oltre all'istruzione di test, si esegue il ciclo For esterno (sempre esattamente n volte). Nel ciclo For, compare una singola operazione ($m := n$), seguita da un ciclo While. Quest'ultimo viene eseguito tante volte quante ne occorrono per far calare m dal valore n a valori < 1 dividendo m ogni volta per 2:

$$\frac{n}{2^h} < 1 \Rightarrow h > \log_2 n \Rightarrow h = \lfloor \log_2 n \rfloor + 1$$

Quindi, si chiama due volte ricorsivamente l'algoritmo e si eseguono una moltiplicazione e una somma.

Riassumendo

$$T(n) = \begin{cases} 1 & \text{per } n \leq 2 \\ 1 + n\left[1 + (\lfloor \log_2 n \rfloor + 1) \cdot 1\right] + 2T\left(\frac{n}{4}\right) + 2 & \text{per } n > 2 \end{cases}$$

Possiamo applicare il metodo principale, e osservare che $n^{\log_b a} = n^{\log_4 2} = n^{1/2} = \sqrt{n}$. La parte divide e combina dell'algoritmo ha complessità

$$f(n) = n \lfloor \log_2 n \rfloor + 2n + 3$$

che è polinomialmente superiore a $n^{\log_b a}$. Quindi, potremmo essere nel terzo caso del teorema. Per garantirlo, occorre anche $af(n/b) \leq cf(n)$.

$$af\left(\frac{n}{b}\right) = 2f\left(\frac{n}{4}\right) = 2\frac{n}{4}\left\lfloor \log_2 \frac{n}{4} \right\rfloor + 2\frac{n}{4} + 3 =$$

$$= \frac{n}{2}\lfloor \log_2 n \rfloor - \frac{n}{2}2 + 2\frac{n}{4} + 3 = \frac{n}{2}\lfloor \log_2 n \rfloor - \frac{n}{2} + 3$$

Vogliamo dimostrare che è $\leq 1/2\,(n \lfloor \log_2 n \rfloor + 2n + 3)$. Per ottenerlo basta imporre

$$-\frac{n}{2} + 3 \leq n + \frac{3}{2}$$

ma con n sufficientemente grande, questo avviene ($n \geq n_0 = 1$).

La modifica più importante da fare è non chiamare due volte l'algoritmo con lo stesso argomento ($n/4$), sostituendo "Return 2∗ Algo($n/4$) + Algo($n/4$);" con "Return 3∗ Algo($n/4$);". Un altro banale miglioramento consiste nell'osservare che i cicli For e While sono del tutto inutili e si possono omettere senza modificare il risultato dell'algoritmo.