Algoritmi e Strutture Dati

Algoritmi Ricorsivi e Ricorrenze

Maria Rita Di Berardini, Emanuela Merelli[1]

[1]Dipartimento di Matematica e Informatica
Università di Camerino

A.A. 2006/07

L'isola dei conigli

Leonardo da Pisa (noto anche come Fibonacci) si interessò di molte cose, tra cui il seguente problema di dinamica delle popolazioni:

Quanto velocemente si espanderebbe una popolazione di conigli sotto appropriate condizioni?

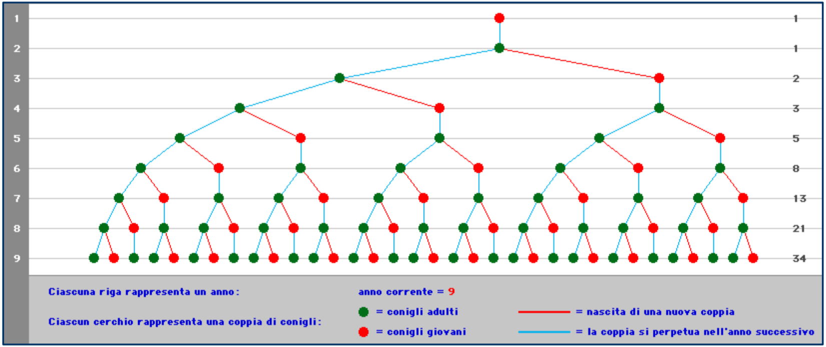
In particolare, partendo da una coppia di conigli in un'isola deserta, quante coppie si avrebbero nell'anno n?

Le regole di riproduzione

- Una coppia di conigli genera due coniglietti ogni anno
- I conigli cominciano a riprodursi soltanto al secondo anno dopo la loro nascita
- I conigli sono immortali

L'albero dei conigli

La riproduzione dei conigli può essere descritta in un albero come segue:



La regola di espansione

Nell'anno n, ci sono tutte le coppie dell'anno precedente, più una nuova coppia di conigli per ogni coppia presente due anni prima

Indicando con F_n il numero di coppie dell'anno n, abbiamo la seguente relazione di ricorrenza:

$$F(n) = \begin{cases} F(n-1) + F(n-2) & \text{se } n \geq 3 \\ 1 & \text{se } n = 1, 2 \end{cases}$$

Il problema

Come calcoliamo $F(n)$?

Un approccio numerico

Possiamo usare una funzione matematica che calcoli direttamente i numeri di Fibonacci

Si può dimostrare che:

$$F(n) = \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right)$$

dove

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$$

e

$$\hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -0.618$$

Algoritmo `fibonacci1`

algoritmo `fibonacci1`(intero n) $\rightarrow$ intero

return $\frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right)$

L'algoritmo `fibonacci1` è corretto?

Problema: qual è l'accuratezza su ϕ e $\hat{\phi}$ per ottenere un risultato corretto?

Ad esempio con tre cifre decimali: $\phi \approx 1.618$ e $\hat{\phi} \approx -0.618$

n	`fibonacci1`	arrotondamento	$F(n)$
3	1.99992	2	2
16	986.698	987	987
18	2583.1	2583	2584

L'algoritmo `fibonacci2`

L'algoritmo `fibonacci1` non è corretto; un possibile approccio alternativo consiste nell'utilizzare un algoritmo ricorsivo:

algoritmo `fibonacci2`(intero n) $\rightarrow$ intero
    if $(n \leq 2)$ **return** 1
    else return fibonacci2$(n-1)+$
        fibonacci2$(n-2)$

Opera solo su numeri interi

L'algoritmo `fibonacc12`

Il costo di esecuzione di `fibonacci2` è espresso dalla seguente funzione ricorsiva (equazione di ricorrenza o semplicemente ricorrenza)

$$T(n) = \begin{cases} T(n-1) + T(n-2) & \text{se } n \geq 3 \\ c & \text{se } n = 1, 2 \end{cases}$$

vi ricorda qualcosa ??

L'algoritmo `fibonacci2`

Il costo di esecuzione di `fibonacci2` è espresso dalla seguente funzione ricorsiva (equazione di ricorrenza o semplicemente ricorrenza)

$$T(n) = \begin{cases} T(n-1) + T(n-2) & \text{se } n \geq 3 \\ c & \text{se } n = 1, 2 \end{cases}$$

vi ricorda qualcosa ??

$$F(n) = \begin{cases} F(n-1) + F(n-2) & \text{se } n \geq 3 \\ 1 & \text{se } n = 1, 2 \end{cases}$$

L'algoritmo `fibonacci2`

Il costo di esecuzione di `fibonacci2` è espresso dalla seguente funzione ricorsiva (equazione di ricorrenza o semplicemente ricorrenza)

$$T(n) = \begin{cases} T(n-1) + T(n-2) & \text{se } n \geq 3 \\ c & \text{se } n = 1, 2 \end{cases}$$

vi ricorda qualcosa ??

$$F(n) = \begin{cases} F(n-1) + F(n-2) & \text{se } n \geq 3 \\ 1 & \text{se } n = 1, 2 \end{cases}$$

Possiamo dimostrare che

$$T(n) = cF(n) = c\frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n)$$

L'algoritmo `fibonacci2`

In realtà `fibonacci2` è un algoritmo molto lento: $T(n) = O(2^n)$

Dimostrazione.

Dalla definizione di O, dobbiamo dimostrare che esistono delle costanti positive c_0 ed n_0 tali che $T(n) \leq c_0 2^n$ per ogni $n \geq n_0$. La prova è per induzione su n □

Induzione

Problema: dimostrare che una affermazione $Aff(n)$ è vera per ogni $n \geq 0$

Theorem (induzione 1)

Se

1. $Aff(0)$ *è vera;*
2. $Aff(n-1)$ *vera implica* $Aff(n)$ *vera;*

allora $Aff(n)$ *è vera per ogni* $n \geq 0$

Theorem (induzione 2)

Se

1. $Aff(0)$ *è vera;*
2. $Aff(n')$ *vera per ogni* $n' < n$ *implica* $Aff(n)$ *vera;*

allora $Aff(n)$ *è vera per ogni* $n \geq 0$

Dimostrazione per induzione: un esempio

Proviamo a dimostrare per induzione su n che

$$Aff(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Caso Base$(i = 1)$:

$$\sum_{i=1}^{1} i = 1 = \frac{1(2)}{2} = \frac{1(1+1)}{2} \quad \text{OK}$$

Passo induttivo: Assumiamo $Aff(n-1)$ vera, cioè

$$\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$$

e dimostriamo che, allora, anche $Aff(n)$ è vera

Dimostrazione per induzione: un esempio

$$\sum_{i=1}^{n} i = \sum_{i=1}^{n-1} i + n = \frac{(n-1)n}{2} + n = \frac{(n-1)n + 2n}{2}$$

$$= \frac{n(n-1+2)}{2} = \frac{n(n+1)}{2} \quad \text{OK}$$

L'algoritmo `fibonacc12`

$T(n) = O(2^n)$.

Dalla definizione di O, dobbiamo dimostrare che esistono delle costanti positive c_0 ed n_0 tali che $T(n) \leq c_0 2^n$ per ogni $n \geq n_0$.

Casi base

- $n = 1$: $T(1) = c \leq c_0 2$ basta scegliere $c_0 \geq c/2$
- $n = 2$: $T(2) = c \leq c_0 4$ vero se $c_0 \geq c/2$ (infatti $c_0 \geq c/2$ implica $4c_0 \geq 4\, c/2 = 2c \geq c$)

Passo induttivo: assumiamo $T(n') \leq c2^{n'}$ per ogni $n' \leq n$. Allora:

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) \\ &\leq c_0 2^{n-1} + c_0 2^{n-2} \\ &= c_0 2^{n-2}(2+1) \\ &\leq c_0 2^{n-2} 4 \\ &= c_0 2^n \end{aligned}$$

Ricerca in un'insieme non ordinato

ricercaSequenziale(array A, elem x)

> **for** i $\leftarrow$ 1 to lenght[A]
>   **do if** $A[i] = x$ **return** trovato
> **return** non trovato

$T_{\text{best}}(n) = 1$	x è in prima posizione
$T_{\text{worst}}(n) = n$	x è in ultima posizione oppure non è in L
$T_{\text{avarange}}(n) = (n+1)/2$	sotto un assunzione di equi-distribuzione delle istanze

Ricerca in un'insieme ordinato

RicercaBinaria(array A, elem x)

n ← Lunghezza di A
if n=0 **return** non trovato
i ← $\lceil n/2 \rceil$
if A[i] = x **return** trovato
else if $A[i] > x$ RicercaBinaria($A[1; i-1]$, x)
    else RicercaBinaria($A[i+1; n]$, x)

Ricerca in un'insieme ordinato

RicercaBinaria(array A, elem x)

n $\leftarrow$ Lunghezza di A
if n=0 **return** non trovato
i $\leftarrow \lceil n/2 \rceil$
if A[i] $=$ x **return** trovato
else if $A[i] > x$ RicercaBinaria($A[1; i-1]$, x)
    else RicercaBinaria($A[i+1; n]$, x)

Come analizziamo questo algoritmo?

Tempo di esecuzione di RicercaBinaria

È descritto mediante la seguente **equazione di ricorrenza** :

$$T(n) = \begin{cases} c + T(\lceil (n-1)/2 \rceil) & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

Cosa è una relazione di ricorrenza

Una **relazione di ricorrenza** - o più semplicemente **ricorrenza** - è una equazione che descrive una funzione in termini del suo valore con input più piccoli

Esistono tre grandi metodi per risolvere le ricorrenze - ovvero per ottenere dei limiti asintotici "Θ" o "O"

- Il metodo di sostituzione
- Il metodo iterativo – metodo della albero di ricorsione
- Il metodo dell'esperto che consente di calcolare i limiti per ricorrenze della forma $T(n) = aT(n/b) + f(n)$ dove $a \geq 1$, $b > 0$ ed $f(n)$ è una funzione data

Ricorrenze: un caso semplice

Metodo iterativo: consiste nello srotolare la ricorsione fino ad ottenere una sommatoria dipendente da n

Esempio:

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

Proviamo ad applicare il metodo iterativo a $T(n)$

$$\begin{aligned} T(n) = & \; 1 + T(n/2) \\ = & \; 1 + 1 + T(n/4) = 2 + T(n/4) \\ = & \; 2 + 1 + T(n/8) = 3 + T(n/8) \\ = & \; 3 + 1 + T(n/16) = 4 + T(n/16) \\ = & \; \cdots \end{aligned}$$

Ricorrenze: un caso semplice

$$
\begin{aligned}
T(n) = & \; 1 + T(n/2) \\
= & \; 1 + 1 + T(n/4) = 2 + T(n/4) = 2 + T(n/2^2) \\
= & \; 2 + 1 + T(n/8) = 3 + T(n/8) = 3 + T(n/2^3) \\
= & \; 3 + 1 + T(n/16) = 4 + T(n/16) = 4 + T(n/2^4) \\
= & \; \dots \\
= & \; k + T(n/2^k)
\end{aligned}
$$

Continuiamo a srotolare la ricorsione fin quando $n/2^k = 1$; ora $n/2^k = 1$ implica $2^k = n$ e quindi $k = \log_2 n$ (k è il logaritmo in base 2 di n). Allora:

$$T(n) = \log_2 n + 1 = O(\log_2 n)$$

Metodo iterativo: un altro esempio (meno semplice)

Il metodo iterativo può essere applicato a qualsiasi ricorrenza, ma a volte può essere di difficile soluzione

Esempio:

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

Proviamo ad applicare il metodo iterativo a $T(n)$

$$\begin{aligned} T(n) &= n + T(n/2) \\ &= n + n/2 + T(n/4) \\ &= n + n/2 + n/4 + T(n/8) \\ &= n + n/2 + n/4 + n/8 + T(n/16) \\ &= \dots \end{aligned}$$

Metodo iterativo: un altro esempio (meno semplice)

$$\begin{aligned}
T(n) &= n + T(n/2) \\
&= n + n/2 + T(n/4) \\
&= n + n/2 + n/4 + T(n/8) \\
&= n + n/2 + n/2^2 + n/2^3 + T(n/16) \\
&= \dots \\
&= n + n/2 + n/2^2 + n/2^3 + \dots + n/2^{k-1} + T(n/2^k) \\
&= \textstyle\sum_{i=0}^{k-1} n/2^i + T(n/2^k)
\end{aligned}$$

Di nuovo ci fermiamo $k = \log_2 n$. Allora:

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

Metodo iterativo: un altro esempio (meno semplice)

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

Fact

$$\sum_{i=0}^{m} \alpha^i = \frac{1-\alpha^{m+1}}{1-\alpha}$$

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

Metodo iterativo: un altro esempio (meno semplice)

$$T(n) = \quad n\,\frac{1-(1/2)^{\log_2 n}}{1-(1/2)} + 1$$

Metodo iterativo: un altro esempio (meno semplice)

$$T(n) = n \frac{1-(1/2)^{\log_2 n}}{1-(1/2)} + 1$$

$$= n \frac{1-(1/2)^{\log_2 n}}{1/2} + 1$$

Metodo iterativo: un altro esempio (meno semplice)

$$
\begin{aligned}
T(n) &= n\,\frac{1-(1/2)^{\log_2 n}}{1-(1/2)}+1 \\
&= n\,\frac{1-(1/2)^{\log_2 n}}{1/2}+1 \\
&= 2n\left(1-(1/2)^{\log_2 n}\right)+1
\end{aligned}
$$

Metodo iterativo: un altro esempio (meno semplice)

$$
\begin{aligned}
T(n) &= n\,\frac{1-(1/2)^{\log_2 n}}{1-(1/2)} + 1 \\
&= n\,\frac{1-(1/2)^{\log_2 n}}{1/2} + 1 \\
&= 2n\left(1 - (1/2)^{\log_2 n}\right) + 1 \\
&= 2n\left(1 - (1/2^{\log_2 n})\right) + 1
\end{aligned}
$$

Metodo iterativo: un altro esempio (meno semplice)

$$
\begin{aligned}
T(n) &= n \frac{1-(1/2)^{\log_2 n}}{1-(1/2)} + 1 \\
&= n \frac{1-(1/2)^{\log_2 n}}{1/2} + 1 \\
&= 2n\left(1 - (1/2)^{\log_2 n}\right) + 1 \\
&= 2n\left(1 - (1/2^{\log_2 n})\right) + 1 \\
&= 2n\left(1 - 1/n\right) + 1
\end{aligned}
$$

Metodo iterativo: un altro esempio (meno semplice)

$$
\begin{aligned}
T(n) &= n\,\frac{1-(1/2)^{\log_2 n}}{1-(1/2)} + 1 \\
&= n\,\frac{1-(1/2)^{\log_2 n}}{1/2} + 1 \\
&= 2n\left(1 - (1/2)^{\log_2 n}\right) + 1 \\
&= 2n\left(1 - (1/2^{\log_2 n})\right) + 1 \\
&= 2n\left(1 - 1/n\right) + 1 \\
&= 2n\left((n-1)/n\right) + 1
\end{aligned}
$$

Metodo iterativo: un altro esempio (meno semplice)

$$
\begin{aligned}
T(n) &= n\,\frac{1-(1/2)^{\log_2 n}}{1-(1/2)} + 1 \\
&= n\,\frac{1-(1/2)^{\log_2 n}}{1/2} + 1 \\
&= 2n\left(1-(1/2)^{\log_2 n}\right) + 1 \\
&= 2n\left(1-(1/2^{\log_2 n})\right) + 1 \\
&= 2n\left(1-1/n\right) + 1 \\
&= 2n\left((n-1)/n\right) + 1 \\
&= 2\left(n-1\right) + 1 = 2n-1 = O(n)
\end{aligned}
$$

Alberi di ricorsione

Vengono usati in alternativa al metodo iterativo

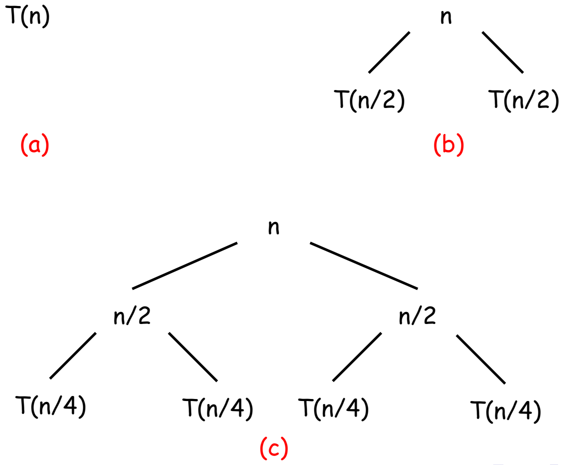
Un albero di ricorsione è un albero in cui ogni nodo rappresenta il costo di un sottoproblema da qualche parte nell'insieme delle chiamate ricorsive

Consideriamo la ricorrenza

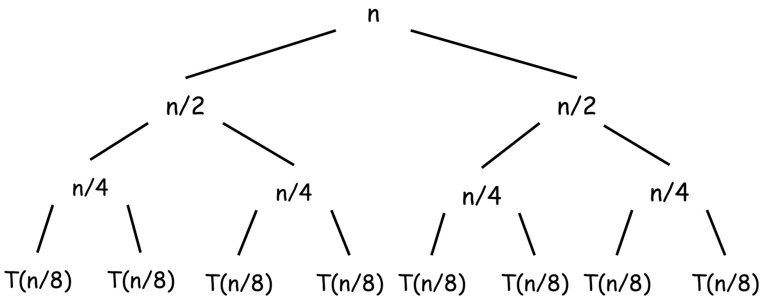
$$T(n) = \begin{cases} n + 2T(n/2) & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

e costruiamo la derivazione dell'albero delle ricorrenze per $T(n)$

Alberi di ricorsione: un esempio

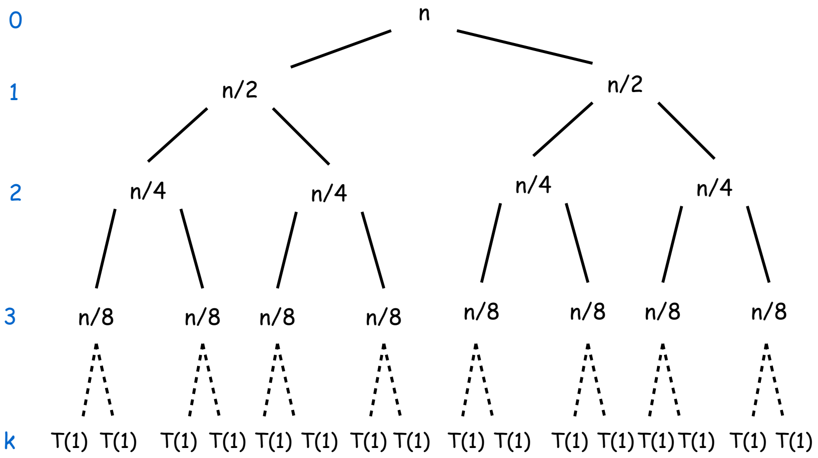


Alberi di ricorsione



(d)

Alberi di ricorsione



Alberi di ricorsione

Sia i un dato livello dell'albero di ricorsione ($i = 0, \ldots, \#livelli$) (al momento, non meglio specificato)

- Quanti nodi ci sono al livello i? esattamente 2^i nodi (al livello 0 abbiamo $1 = 2^0$ nodi, ed ogni livello ha un numero di nodi doppio rispetto al livello precedente)
- Quale è il costo di un nodo al livello i? ogni nodo al livello i costa $n/2^i$ nodi (l'unico nodo al livello 0 ha un costo pari ad $n = n/1 = n/2^0$, ed ogni volta che scendiamo di livello il costo dei nodi si dimezza)
- il costo complessivo dei nodi al livello i è

$$\#nodi(i) \times costo\_nodo(i) = 2^i \times n/2^i = n$$

Alberi di ricorsione

Quale è il costo complessivo della chiamata $T(n)$?

$$T(n) = \sum_{j=0}^{\#livelli} costo\_livello(j) = \sum_{j=0}^{\#livelli} n = n\,(\#livelli + 1)$$

Ci rimane da determinare questo $\#livelli$

L'ultimo livello corrisponde ad una serie di chiamate di $T(1)$ (ci fermiamo quando la dimensione del problema è 1)

$1 = n/2^k$ con k pari all'altezza dell'albero di ricorsione e quindi al $\#livelli$: $n/2^k = 1$ implica $2^k = n$ e quindi $\#livelli = k = \log_2 n$

Alberi di ricorsione

Ricapitolando:

$$T(n) = n\,(\log_2 n + 1) = \Theta(n \log_2 n)$$

Metodo della sostituzione

Metodo della sostituzione: "indovinare" una possibile soluzione ed usare l'induzione matematica per dimostrare che la soluzione è corretta

Consideriamo di nuovo la ricorrenza

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

e dimostriamo, applicando il metodo della sostituzione, che

$$T(n) = O(n)$$

Metodo della sostituzione

Dobbiamo dimostrare che che esistono delle costanti positive c ed n_0 tali che $0 \leq T(n) \leq cn$ per ogni $n \geq n_0$

Caso base $n = 1$: $T(1) = 1 \leq c1 = c$ per ogni costante $c \geq 1$ (positiva)

Passo induttivo: assumiamo $T(n') \leq cn'$ per ogni $n' < n$. Allora

$$\begin{array}{rcll} T(n) & = & n + T(n/2) & \\ & \leq & n + cn/2 & \\ & = & n(1 + c/2) & \text{se scegliamo } c \geq 2^{[1]} \\ & \leq & cn & \end{array}$$

[1] Infatti se $c \geq 2$, allora $1 \leq c/2$ e $1 + c/2 \leq c/2 + c/2 = c$

Metodo della sostituzione: un altro esempio

Consideriamo la seguente ricorrenza

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

Dimostriamo che

$$T(n) = O(n \log_2 n)$$

ossia che che esistono delle costanti positive c ed n_0 tali che $0 \leq T(n) \leq cn \log_2 n$ per ogni $n \geq n_0$

Caso base $n = 1$: $T(1) = 1 \leq c0 = 0$ è chiaramente falsa

$n = 2$: $T(2) = 2 + 2T(1) = 4 \leq 2c$ basta scegliere $c \geq 2$

Metodo della sostituzione: un altro esempio

Passo induttivo: assumiamo $T(n') \leq cn' \log_2 n'$ per ogni $n' < n$. Allora

$$
\begin{aligned}
T(n) &= n + 2T(n/2) \\
&\leq n + 2c(n/2)\log_2(n/2) \\
&= n + cn\log_2(n/2) \\
&= n + cn(\log_2 n - \log_2 2) \\
&= n + cn(\log_2 n - 1) \\
&= n + cn\log_2 n - cn \\
&= cn\log_2 n + n(1-c) \\
&\leq cn\log_2 n \qquad \text{per ogni } c \geq 2
\end{aligned}
$$

Quindi, se scegliamo $c \geq 2$ ed $n_0 = 2$, allora $T(n) \leq cn\log_2 n$ per ogni $n \geq n_0$

Teorema dell'esperto – Teorema Master

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

- dividi il problema (di dimensione n) in a sotto-problemi di dimensione n/b
- risolvi i sotto-problemi ricorsivamente
- ricombina le soluzioni

Sia $f(n)$ il tempo per dividere e ricombinare istanze di dimensione n. La relazione di ricorrenza è data da:

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

Teorema dell'esperto – Teorema Master

La soluzione della ricorrenza

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

dipende da $f(n)$. Più precisamente:

1. Se $f(n) = O(n^{\log_b a - \varepsilon})$ per qualche costante $\varepsilon > 0$, allora $T(n) = \Theta(n^{\log_b a})$
2. Se $f(n) = \Theta(n^{\log_b a})$, allora $T(n) = \Theta(n^{\log_b a} \log_2 n)$
3. Se $f(n) = \Omega(n^{\log_b a + \varepsilon})$ per qualche costante $\varepsilon > 0$ e se, per qualche costante $c < 1$, $af(n/b) \leq cf(n)$, allora $T(n) = \Theta(f(n))$

Applicazioni del teorema dell'esperto

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

$$a = b = 2, \ \log_b a = 1$$
$$f(n) = \Theta(n) = \Theta(n^{\log_b a})$$
(caso 2 del teorema master)

$$\Downarrow$$

$$T(n) = \Theta(n^{\log_b a} \log_2 n) = \Theta(n \log_2 n)$$

Applicazioni del teorema dell'esperto

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

$$a = 9,\ b = 3,\ \log_b a = \log_3 9 = 2$$
$$f(n) = c = O(n) = O(n^{\log_b a - \varepsilon}) = O(n^{2-\varepsilon}) \text{ con } \varepsilon = 1$$
(caso 1 del teorema master)

$$\Downarrow$$

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$$

Applicazioni del teorema dell'esperto

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

$$a = 3,\ b = 9,\ \log_b a = \log_9 3 = 1/2\ (3 = \sqrt{9} = 9^{1/2})$$
$$f(n) = n = \Omega(n) = \Omega(n^{\log_9 3 + \varepsilon}) \text{ con } \varepsilon = 1/2$$
inoltre,
$$af(n/b) = 3f(n/9) = 3(n/9) = 1/3n \leq cf(n), \text{ per } c = 1/3$$
(caso 3 del teorema master)

$$\Downarrow$$

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