Università di Pisa

Dipartimento di Informatica

Technical Report: TR-09-02

Soluzione di ricorrenze nell'analisi di algoritmi.

Fabrizio Luccio

February 1, 2009

ADDRESS: Largo B. Pontecorvo 3, 56127 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

Soluzione di ricorrenze nell'analisi di algoritmi.

Fabrizio Luccio

February 1, 2009

Abstract

Le relazioni di ricorrenza che descrivono il tempo di funzionamento di algoritmi ricorsivi si risolvono in diversi modi. Presentiamo qui alcuni teoremi molto semplici che permettono di affrontare le due famiglie di ricorrenze che si presentano più comunemente. Questi teoremi sono molto più semplici di quelli che si trovano in genere sui libri di testo e si applicano in pratica negli stessi casi. In particolare dimostriamo:

1. Le *relazioni bilanciate* della forma $T(n) = aT(n/b)+\Theta(n^e)$ hanno soluzione: $\Theta(n^e)$ per $a < b^e$; $\Theta(n^e \log n)$ per $a = b^e$; $\Theta(n^{\log_b a})$ per $a > b^e$ (Teorema 2). Questo risultato è identico a quello di Erickson [3], ma è espresso e dimostrato in forma molto più semplice. Un'immediata estensione permmette di sostituire il termine $\Theta(n^e)$ con $\Theta(f(n))$ arbitrario e ottenere con la stessa semplicità una soluzione, a volte in ordine O anziché Θ (Teorema 3).

2. Per le *relazioni di ordine k* della forma $T(n) = a_1T(n-1) + ... + a_kT(n-k)+f(n)$, $a_k \geq 1$, si risolve solo il caso $T(n) = T(n-k)+\Theta(n^e)$, $k \geq 1$, che ha soluzione $\Theta(n^{e+1})$ (Teorema 4), e si dimostra che la soluzione è esponenziale in n se almeno una delle a_i non è nulla, $0 \leq i \leq k-1$, o se $a_k > 1$, cioè se l'algoritmo originale contiene almeno due chiamate ricorsive (Teorema 5).

Se non diversamente specificato, ammettiamo di lavorare in *word model*, cio che un problema sia definito per un numero arbitrario n di dati, ma ognuno di essi sia descritto con un numero di bit limitato superiormante da una costante e sia reperibile in tempo costante (tipicamente sia contenuto in una *parola* di una memoria ad accesso diretto). L'analisi *asintotica*, cioè in ordine di grandezza, del tempo $T(n)$ richiesto da un algoritmo ricorsivo richiede allora in genere la soluzione di una *relazione di ricorrenza lineare* del tipo:

$$T(n) = a_1T(n_1) + ... + a_kT(n_k) + f(n), \tag{1}$$

soggetta alle condizioni iniziali:

$$T(1) = \Theta(1), ..., T(h) = \Theta(1), \tag{2}$$

ove k è una costante intera ≥ 1; $n_1, ..., n_k$ sono le dimensioni dei sottoinsiemi di dati su cui operano le diverse chiamate ricorsive dell'algoritmo; $a_1, ..., a_k$ sono costanti intere, $a_1, ..., a_{k-1} \geq 0$, $a_k \geq 1$; e $f(n)$ è una funzione non decrescente che rappresenta il lavoro svolto dall'algoritmo al di fuori delle chiamate ricorsive, in particolare per dividere i dati nei sottoinsiemi cui tali chiamate si riferiscono e per combinarne i risultati. Trattiamo qui solo le relazioni più comuni nell'analisi di algoritmi, che hanno la forma della (1) e in cui, per ogni $1 \leq i \leq k$, si ha uno dei due casi:

1. $n_i = n/b_i$, con b_i costante, $1<b_i<n$, e n/b_i interpretato come $\lfloor n/b_i \rfloor$ o $\lceil n/b_i \rceil$;
2. $n_i = n - b_i$, con b_i costante intera, $1 \leq b_i < n$.

Notiamo che $n_1, ..., n_k$ sono funzioni di n e poniamo $n > n_1 > ... > n_k$, quindi $b_1 < ... < b_k$. Le relazioni del primo e secondo tipo saranno dette rispettivamente *realzioni bilanciate* e *relazioni di ordine k*, anche se questa terminologia non è impiegata da tutti.

Per quanto riguarda le condizioni iniziali (2) notiamo che i valori $T(i)$, $1 \leq i \leq h$, sono costanti in quanto indipendenti da n, e che il loro numero h deve essere sufficiente a far partire la successione per n generico, come vedremo in seguito. In particolare avremo in genere $h = 1$, ma potrà a volte essere comodo considerare anche la condizione iniziale $T(0)$ che corrisponde a operare su un sottoinsieme vuoto di dati.

1 Due semplici esempi

ESEMPIO 1. In un torneo di tennis competono n giocatori $G[1], ..., G[n]$, con n potenza di 2. L'algoritmo classico di organizzazione degli incontri può essere definito ricorsivamente determinando il vincitore H_1 tra $G[1], ..., G[n/2]$ e il vincirore H_2 tra $G[n/2+1], ..., G[n]$, e facendo poi incontrare H_1 e H_2 nella finale. Quando dividendo l'insieme in due parti si incontra un sottoinsieme limite composto da un solo giocatore questi è considerato il vincitore in quel sottoinsieme. Posto $l = 1$ e $r = n$ lo schema è il seguente:

algorithm TOURN(l, r)
    if $(l = r)$ **return** $G[l]$;
    $c = \lfloor (l + r)/2 \rfloor$;
    $H_1 =$ TOURN(l, c); $H_2 =$ TOURN$(c + 1, r)$; **return** WINNER(H_1, H_2).

Per esempio immaginando che i giocatori siano indicati da interi positivi che ne rappresentano la bravura tennistica (quindi la procedura WINNER si riduca a calcolare il massimo tra due interi), per $n = 16$ e per il vettore:

$G:$ 3 12 13 6 8 9 1 10 15 4 16 11 2 5 14 7

gli incontri avverrebbero nell'ordine:

$3-12,\ \ 13-6,\ \ 12-13,\ \ 8-9,\ \ 1-10,\ \ 9-10,\ \ 13-10,\ \ 15-4,$

$$16-11,\ \ 15-16,\ \ 2-5,\ \ 14-7,\ \ 5-14,\ \ 16-14,\ \ 13-16$$

decretando vincitore 16 nella finale contro 13. Il tempo $T(n)$ richiesto dall'algoritmo è espresso dalla relazione bilanciata:

$$T(n) = 2\,T(n/2) + c, \quad \text{con } T(1) = c_1, \tag{3}$$

ove c_1 è il tempo necessario a proclamare vincitore il giocatore di un sottoinsieme composto di un solo elemento (condizione $l = r$); e c è il tempo (costante) necessario a dividere l'insieme in due parti e a eseguire il confronto tra i due vincitori nella procedura WINNER, dopo le chiamate ricorsive. Con la notazione precedente abbiamo $a_1 = 2$, $b_1 = 2$, $f(n) = c$ costante. Risolveremo la relazione (3) nel prossimo paragrafo 2.

ESEMPIO 2. Nella pallacanestro, fino ad alcuni anni fa, i canestri potevano valere uno o due punti. Posto che una squadra abbia realizzato n punti, si devono costruire tutte le sequenze di canestri che possono aver generato tale punteggio. Anzitutto il **numero** R_n di tali sequenze si ottiene notando che $R_0 = 1$ (esiste solo la sequenza vuota), $R_1 = 1$ (esiste solo la sequanza composta da un canestro da 1), e in genere $R_n = R_{n-1} + R_{n-2}$ perché la sequenza può cominciare con 1 o con 2, seguiti rispettivamente da tutte le sequenze con punteggio complessivo $n-1$ o $n-2$. Come si vede si tratta della successione di Fibonacci, anche se non risulta che il matematico pisano conoscesse il basket.

Un algoritmo per risolvere il problema registrerà le soluzioni una dopo l'altra in un vettore $S[1..n]$ inizialmente azzerato, sotto forma di sequenze di 1 e 2 a partire da $S[1]$ (la sequenza di soli 1 termina in $S[n]$, le altre terminano prima, quando cominciano gli 0). Considerando che il primo canestro sia stato da 1 o 2 punti, si porrà alternativamente $S[1] = 1$ e $S[1] = 2$ e si costruiranno poi ricorsivamente le due sequenze residue che raggiungono punteggio di $n-1$ e $n-2$. Ponendo $i = 1$ e $j = n$ lo schema è il seguente (invitiamo a esaminarlo attentamente perché è semplice ma non elementare):

algorithm BASKET(i, j)
  if $(j = 0)$ **return** $S[1..n]$;
  $S[i] = 1$; **if** $(j \geq 1)$ BASKET$(i+1, j-1)$;
  if $(j \geq 2)$ $\{S[i] = 2;$ BASKET$(i+1, j-2)\}$.

Per esempio per $n = 5$ l'algoritmo genera nell'ordine le sequenze

1 1 1 1 1 , 1 1 1 2 0 , 1 1 2 1 0 , 1 2 1 1 0 , 1 2 2 0 0 ,
2 1 1 1 0 , 2 1 2 0 0 , 2 2 1 0 0 .

Le sequenze generate sono $R(5) = 8$, pari al numero di Fibonacci di uguale indice. Il tempo $T(n)$ richiesto dall'algoritmo è espresso dalla relazione di ordine 2:

$$T(n) = T(n-1) + T(n-2) + \alpha\, n + \beta, \quad \text{con } T(0) = c_0, T(1) = c_1, \tag{4}$$

ove c_0 è il tempo costante richiesto dal test iniziale su j; c_1 è il tempo costante per eseguire due test su j, registrare in $S[1]$ un canestro da un punto e invocare la procedura BASKET$(i+1, j-1=0)$; $\alpha\, n+\beta$, con α e β costanti, è il tempo costante per eseguire test e assegnamenti di valore, più il tempo lineare in n per trasmettere all'esterno il vettore S. Con la notazione precedente abbiamo $a_1 = a_2 = 1$ e $f(n) = \Theta(n)$. Discuteremo la soluzione della relazione (4) nel prossimo paragrafo 3.

2 Relazioni bilanciate

Un noto metodo per risolvere un'importante classe di relazioni bilanciate fu proposto nel 1980 da Bentley, Haken e Saxe [1]. Il loro *teorema principale* è apparso successivamente in molti libri in forma ridotta, cioè atta a risolvere una classe più ristretta di relazioni: in particolare nel notissimo testo di Cormen, Leiserson e Rivest [2], e, in forma anche più ridotta, nelle dispense di Jeff Erickson. Proponiamo qui una versione elementare del teorema utile per l'analisi della maggior parte degli algoritmi ricorsivi, che permette di risolvere in modo più semplice le stesse relazioni considerate da Erickson in [3] e ivi proposte in una forma solo apparentemente più generale. Sul confronto tra il teorema qui proposto e le versioni precedentisi si veda il paragrafo 2.2. Altri ragionamenti ci consentiranno poi di impiegare il teorema anche per risolvere relazioni più generali. Diversi metodi che non sono basati sul teorema principale nelle sue varie forme sono riportati in un'altra dispensa.

Le relazioni bilanciate considerate qui hanno la forma:

$$T(n) = a\,T(n/b) + c\,n^e, \quad \text{con } T(1) = c_1, \tag{5}$$

con a, b, c, e, c_1 costanti; a intera ≥ 1, b intera > 1, $c > 0$, $e \geq 0$, $c_1 \geq 0$.

Teorema 1. Posto $n = b^m$ con m intero > 1, la relazione (5) ha soluzione:

1. $T(n) = \Theta(n^e)$, se $a < b^e$;
2. $T(n) = \Theta(n^e \log n)$, se $a = b^e$;
3. $T(n) = \Theta(n^{\log_b a})$, se $a > b^e$.

Dimostrazione. Ricordiamo anzitutto alcune relazioni aritmetiche elementari.

$$n = b^m \;\Rightarrow\; m = \log_b n; \quad a^{\log_b n} = n^{\log_b a}. \tag{6}$$

Per $S = \sum_{i=0}^{m-1} r^i$ con r costante:

$$S = m \text{ , per } r = 1; \quad S = \tfrac{1-r^m}{1-r} \text{ , per } r \neq 1; \tag{7}$$

quindi:

$$S = \Theta(1) \text{ , per } r < 1; \quad S = \Theta(r^{m-1}) = \Theta(r^m) \text{ , per } r > 1. \tag{8}$$

Sviluppiamo ora la (5) notando che il termine $T(n/b)$ si può calcolare con la stessa (5) sostituendovi n/b al posto di n per ottenere $T(n/b) = aT(n/b^2) + c(n/b)^e$. Proseguendo con successive sostituzioni, e ricordando che $n = b^m$, $T(1) = c_1$, abbiamo:

$$
\begin{aligned}
T(n) &= a\,T(n/b) + c\,n^e = a^2T(n/b^2) + ac(n/b)^e + cn^e \\
&= a^3T(n/b^3) + a^2c(n/b^2)^e + ac(n/b)^e + cn^e = \ldots \\
&= a^mT(n/b^m) + a^{m-1}c(n/b^{m-1})^e + \ldots + a^2c(n/b^2)^e + ac(n/b)^e + cn^e \\
&= c_1a^m + cn^e \textstyle\sum_{i=0}^{m-1}(a/b^e)^i = c_1n^{\log_b a} + cn^e \sum_{i=0}^{m-1}(a/b^e)^i \qquad (9)
\end{aligned}
$$

ove l'ultimo passaggio deriva dalle (6). Per i tre casi del teorema abbiamo dunque:

1. $a < b^e$, quindi $\log_b a < e$; inoltre $\sum_{i=0}^{m-1}(a/b^e)^i = \Theta(1)$ per la prima delle (8). Quindi $T(n) = \Theta(n^e)$.

2. $a = b^e$, quindi $\log_b a = e$; inoltre $\sum_{i=0}^{m-1}(a/b^e)^i = m = \log_b n$ per la prima delle (7). Quindi $T(n) = \Theta(n^e \log n)$.

3. $a > b^e$, quindi $\log_b a > e$; inoltre $\sum_{i=0}^{m-1}(a/b^e)^i = \Theta((a/b^e)^m)$ per la seconda delle (8). La (9) diviene quindi: $T(n) = c_1n^{\log_b a} + \Theta(n^e(a/b^e)^m)$ e si ha: $n^e(a/b^e)^m = a^m(n/b^m)^e = a^{\log_b n}(1)^e = n^{\log_b a}$. Quindi i due termini della (9) hanno lo stesso valore asintotico e si ha: $T(n) = \Theta(n^{\log_b a})$. Q.E.D.

Per esempio la relazione (3) relativa al torneo ha la forma (5) con $a = b = 2$, $e = 0$; dunque $b^e = 1$, $\log_b a = 1$, $a > b^e$. Per n potenza di $b = 2$ si applica il teorema 1, caso 3, e si ha $T(n) = \Theta(n)$. Nell'esempio svolto sopra, con $n = 16$ quindi $m = 4$, abbiamo visto che si eseguono 15 incontri tra giocatori: si può facilmente controllare che l'algoritmo TOURN richiede $n - 1$ incontri e il numero di altre operazioni è ad esso proporzionale. Vediamo ora un'estensione immediata del teorema 1.

Corollario 1.1. Il teorema 1 è valido anche se nella ricorrenza (5) il termine cn^e è sostituito con $\Theta(n^e)$.
Dimostrazione. Ogni funzione in $\Theta(n^e)$ si può scrivere come: $cn^e + o(n^e)$. Sostituendo questa forma nello sviluppo per $T(n)$ indicato nella dimostrazione del teorema 1, all'espressione finale (9) si aggiungono termini di valore $o(n^e \sum_{i=0}^{m-1}(a/b^e)^i)$, e il risultato non cambia in ordine di grandezza. Q.E.D.

Per esempio il noto algoritmo MERGESORT ordina un insieme di n elementi dividendolo in due sottoinsiemi di $n/2$ elementi ciascuno, che vengono ordinati ricorsivamente e poi fusi tra loro in tempo lineare. La ricorrenza che governa questo procedimento è: $T(n) = 2T(n/2) + \Theta(n)$, con $T(1) = c_1$. Si ha $a = b = 2$, $e = 1$, quindi $a = b^e$. Per n potenza di 2 si applica il teorema 1, caso 2, e si ha il noto risultato $T(n) = \Theta(n \log n)$.

2.1 Altre due estensioni del teorema 1.

Perché il teorema 1 e il successivo corollario possano essere utilizzati in pratica abbiamo bisogno di altre due estensioni. È bene sottolineare che queste non dipendono dalla forma semplificata da noi adottata per la ricorrenza, e sono necessarie in tutte le forme del terema principale indicate nella letteratura.

Anzitutto una procedura ricorsiva non deve necessariamente raggiungere sottoinsiemi limite di un elemento, ma può richiedere che il problema sia risolto direttamente quando il parametro raggiunge un valore costante $\leq h$. Abbiamo allora:

Corollario 1.2. Il teorema 1 è valido anche se la ricorrenza (5) è espressa per $n > h$, con $h > 1$ costante e condizioni iniziali $T(1) = \Theta(1), ..., T(h) = \Theta(1)$.
Dimostrazione. Maggioriamo h come la più piccola costante $h' \geq h$ che sia potenza di b, quindi $n = b^g h'$; e poniamo che tutte le chiamate ricorsive terminino sul valore limite h'. Lo sviluppo della (9) si arresta allora per $T(n/b^g){=}T(h'){=}\Theta(1)$. Notando che $a^g = \alpha\, a^{\log_b n}$ con $\alpha = 1/a^{\log_b h'}$ costante, si vede facilmente che i termini della (9) cambiano solo per una costante moltiplicativa. Poiché la soluzione del teorema è espressa in ordine di grandezza e non cambia se la ricorrenza termina sui i valori limite $T(1)$ e $T(h')$, tale soluzione rimane valida per ogni $T(1), ..., T(h)$. Q.E.D.

Si noti che i risultati dei corollari 1.1 e 1.2 valgono anche combinando le rispettive condizioni.

Un aspetto più delicato riguarda il valore di n. Anzitutto notiamo che l'ipotesi $n = b^m$ con m intero, posta nel teorema 1, implica che anche b sia una costante intera poiché n è intero. In effetti in genere si sceglie b intero poiché una procedura ricorsiva si chiama su sottoinsiemi contenenti metà dei dati ($b = 2$), o un terzo dei dati ($b = 3$), ecc., e più difficilmente su sottoinsiemi che ne contengono due terzi ($b = 3/2$), ecc. Ma, pur se b è intero, solo una piccola parte dei valori possibili di n risulterà potenza di b (per ogni intero $N \geq 2$, solo $\lceil \log_b N \rceil$ interi compresi tra 2 e N sono potenze di b). Dobbiamo quindi stabilire come si costruisce un algoritmo per tener conto di tutti i possibili valori di n, e formulare poi un'altra estensione del teorema 1 per le nuove ricorrenze che nascono.

Per chiarire le idee consideriamo un esempio. L'algoritmo di MERGESORT per n dati comprende due chiamate ricorsive su $n/2$ dati (cioè $b = 2$); ma se n non è una potenza di due, nel corso del calcolo alcune chiamate si imbatteranno inevitabilmente in valori dispari del parametro: quindi l'algoritmo standard si formula con le due chiamate su $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$, formulazione valida per n pari o dispari ma che dà luogo alla ricorrenza $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$, diversa da quella risolta nel teorema 1.

Più complicato è il caso in cui n si divida per $b \neq 2$: per esempio si può riformulare MERGESORT richiamando ricorsivamente la procedura su tre gruppi di $n/3$ dati e fondendo poi tra loro i tre gruppi ordinati. Per n potenza di tre la ricorrenza diviene $T(n) = 3T(n/3) + \Theta(n)$, e il corollario 1.1 dà ancora soluzione $T(n) = \Theta(n \log n)$; ma per n qualsiasi le cose si complicano. Infatti per operare su tutti i dati le tre chiamate dovranno essere eseguite su $\lfloor n/3 \rfloor$ e $\lceil n/3 \rceil$; ma in dipendenza del valore di n, che cambia continuamente nei diversi livelli di ricorsione, si dovranno eseguire una o due chiamate su $\lfloor n/3 \rfloor$ e, rispettivamente, due o una chiamate su $\lceil n/3 \rceil$. Per esempio ponendo $n = 38$ la procedura viene chiamata due volte per $\lceil n/3 \rceil = 13$ e una volta per $\lfloor n/3 \rfloor = 12$ (infatti $13 + 13 + 12 = 38$). La prima chiamata su $n = 13$ si risolve poi con una chiamata per $\lceil n/3 \rceil = 5$ e due per $\lfloor n/3 \rfloor = 4$, dunque i due livelli di ricorsione dovrebbero essere valutati con ricorrenze diverse. Strettamente parlando, pur non essendovi alcuna difficoltà a formulare l'algoritmo, non è possibile scrivere la ricorrenza che ne descrive il funzionamento. Notiamo anche che l'algoritmo termina su due casi limite diversi con $n = 2$ e $n = 1$, nei quali non si applica la fusione di tre gruppi, e richiede quindi le due condizioni iniziali $T(1) = c_1$, $T(2) = c_2$ come indicato nel corollario 1.2.

Per risolvere queste situazioni consideriamo il seguente fatto:

Corollario 1.3. Il risultato del teorema 1 rimane identico se il teorema si applica ai due valori $n_1 = b^m$, $n_2 = b^{m+1}$.
Dimostrazione. Nella dimostrazione del teorema 1 poniamo $n = n_1$ e consideriamo il risultato nella espressione (9) e successivi casi 1, 2 e 3. Eseguiamo nuovamente il calcolo per $n = n_2$. Notando che $n_2^{\log_b a} = b\, n_1^{\log_b a}$, e che $\sum_{i=0}^{m-1}(a/b^e)^i$ e $\sum_{i=0}^{m}(a/b^e)^i$ hanno, nei tre casi, lo stesso valore in ordine di grandezza, l'assunto è dimostrato. Q.E.D.

Con una certa forzatura nella notazione, porremo di qui in avanti che:

$$T(n) = a\,T(n/b) + c\,n^e \qquad (10)$$

indichi una ricorrenza in cui il termine $T(n/b)$ appare a volte con parametro $\lceil n/b \rceil$, o $\lfloor n/b \rfloor$, applicati questi anche in numero diverso di volte nei vari livelli della ricorrenza. Posto $n_1 = b^m < n < n_2 = b^{m+1}$, possiamo affermare che la soluzione della (10) è limitata inferiormente e superiormente dalle soluzioni della (5) con parametri n_1 e n_2. Possiamo allora combinare i risultati del teorema 1 e dei corollari 1.1, 1.2, 1.3 esprimendoli nella forma generale:

Teorema 2. Per ogni valore di $n > h$, la ricorrenza:

$$T(n) = a\,T(n/b) + \Theta(n^e), \quad \text{con } T(i) = c_i \text{ per } 1 \leq i \leq h,$$

ha soluzione:

1. $T(n) = \Theta(n^e)$, se $a < b^e$;
2. $T(n) = \Theta(n^e \log n)$, se $a = b^e$;
3. $T(n) = \Theta(n^{\log_b a})$, se $a > b^e$.

2.2 Se la funzione $f(n)$ non è un polinomio.

La forma generale (1) delle relazioni di ricorrenza include una funzione $f(n)$ che rappresenta il lavoro di divisione dei dati e ricombinazione dei risultati. Il teorema principale enunciato in [1], e la sua formulazione ridotta in [2], ammettono in linea di principio che $f(n)$ abbia forma arbitraria, anche se non sempre il teorema si può poi applicare. La formulazione del teorema data in [3], invece, considera relazioni con $f(n)$ di forma arbitraria ma poi impone che la funzione *scali*, il che, come noto, avviene **solo** per le *leggi a potenza* che hanno forma cn^e: dunque questa formulazione di fatto si applica solo alle relazioni come la (5) (né si applica alla generalizzazione posta nel corollario 1.1 o nel teorema 2).

I nostri risultati sono relativi a $f(n) = \Theta(n^e)$. Tuttavia si possono estendere a funzioni diverse espresse in termini assoluti o in ordine di grandezza Θ oppure O, dando per la $T(n)$ risultati in ordine Θ oppure O. La tecnica di risoluzione è molto semplice: si cerca un valore t tale che la $f(n)$ sia limitata superiormente da un polinomio di grado t (sarà opportuno cercare il valore di t più piccolo possibile), e si ha:

Teorema 3. Sia $f(n) \neq \Theta(n^e)$ per qualsiasi valore di e, e sia $t > 0$ una costante tale che $f(n) = O(n^t)$. Per ogni valore di $n > h$, la ricorrenza:

$$T(n) = a\,T(n/b) + \Theta(f(n)), \quad \text{con } T(i) = c_i \text{ per } 1 \leq i \leq h,$$

ha soluzione:

1. $T(n) = O(n^t)$, se $a < b^t$;
2. $T(n) = O(n^{\log_b a} \log n)$, se $a = b^t$;
3. $T(n) = \Theta(n^{\log_b a})$, se $a > b^t$.

Dimostrazione. Considerando la ricorrenza $T_t(n) = a\,T_t(n/b) + \Theta(n^t)$, abbiamo, per ogni valore di n: $T(n) = O(T_t(n))$. Applicando alla ricorrenza il teorema 2 nei tre casi previsti nel presente teorema, la dimostrazione di questo segue immediatamente. Q.E.D.

Abbiamo inoltre immediatamente:

Corollario 3.1. Il risultato del teorema 3 non cambia se nella ricorrenza il termine $\Theta(f(n))$ vierne sostituito con $O(f(n))$.

Consideriamo per esempio il calcolo della somma S di n interi positivi, ciascuno dei quali sia contenuto in una cella di memoria, ovvero sia rappresentato con un numero costante c di bit. Per n crescente il valore di S cresce illimitatamente come cn e la lunghezza della sua rappresentazione diviene $\Theta(\log n)$ (quindi non è più possibile rappresentare S in una cella di memoria). Impiegando un algoritmo che calcola ricorsivamente S_1 e S_2 come somma dei primi e dei secondi $n/2$ dati, e calcola poi $S = S_1 + S_2$, il tempo di funzionamento è dato da:

$$T(n) = 2\,T(n/2) + \Theta(\log n). \tag{11}$$

Possiamo applicare il teorema 3 scegliendo t piccolo a piacere, perché si ha $\log n = o(n^t)$ per ogni $t > 0$. Si ha inoltre $a = 2$, $b = 2$, quindi $a > b^t$, e il caso 3 del teorema dà la soluzione $T(n) = \Theta(n^{\log_b a})$, ovvero $T(n) = \Theta(n)$. Si noti che il risultato non è banale perché il numero di addizioni eseguite è comunque $n-1$, ma, al crescere del valore della somma, queste addizioni richiedono tempo progressivamente crescente fino a divenire $\Theta(\log n)$ per i valori più grandi. Il motivo per cui il tempo totale rimane $\Theta(n)$ (anziché $\Theta(n \log n)$) è che le addizioni tra valori grandi sono proporzionalmente "poche".

Trasformiamo ora l'algoritmo precedente sostituendo l'operazione di addizione con quella di moltiplicazione: calcoleremo così il prodotto P degli n interi. Al crescere di n il valore di P cresce come c^n ed è rappresentato con $\Theta(n)$ bit. Il calcolo del prodotto tra due interi di n bit si esegue in tempo $O(n^2)$ con il metodo ordinario, o in tempo $O(n \log n \log \log n)$ con un metodo molto sofisticato (Schönhage - Strassen 1971). Utilizzando questo abbiamo la ricorrenza:

$$T(n) = 2\,T(n/2) + O(n \log n). \tag{12}$$

Possiamo applicare il teorema 3 nella versione del corollario 3.1, scegliendo $t = 1 + \epsilon$ con $\epsilon > 0$ piccolo a piacere. Abbimo $a = 2$, $b^t = 2^{1+\epsilon}$, e il caso 1 del teorema dà la soluzione: $T(n) = O(n^{1+\epsilon})$. Sorprendentemente il risultato della (12) supera quello della (11) per un fattore polinomiale con esponente piccolo a piacere.

2.3 Chiamate ricorsive su sottoinsiemi di diverse dimensioni.

La forma generale (1) delle ricorrenze prevede che l'algoritmo contenga chiamate per diversi valori $n_1, ..., n_k$. Nelle relazioni bilanciate il caso si presenta in algoritmi particolari, spesso per valori di b_i non interi: per esempio MERGESORT potrebbe essere chiamato su due sottoinsiemi contenenti rispettivanmente un terzo e due terzi dei dati dando luogo alla ricorrenza: $T(n) = T(2n/3) + T(n/3) + \Theta(n)$, con valori $a_1 = a_2 = 1$, $b_1 = 3/2$, $b_2 = 3$.

Per risolvere relazioni di questo tipo non esistono semplici estensioni del teorema principale che permettano di trovare una soluzione sufficientemente accurata, e si deve ricorrere ad altri metodi esposti in una dispensa a parte.

3 Relazioni di ordine k

Come risulta da quanto detto nell'introduzione, queste relazioni di ricorrenza possono essere espresse nella forma:

$$T(n) = a_1 T(n-1) + ... + a_k T(n-k) + f(n), \tag{13}$$

con $k, a_1, ..., a_k$ costanti intere, $k \geq 1$, $a_1, ..., a_{k-1} \geq 0$, $a_k \geq 1$, e almeno k condizioni iniziali perché possa partire il calcolo della successione, assegnate spesso con i valori costanti di $T(0), ..., T(k-1)$.

Uno studio generale di queste relazioni è cruciale nell'analisi di strutture combinatorie: la relativa teoria è esposta in molti testi, si veda per esempio [4]. Nella grande maggioranza dei casi il valore della funzione $T(n)$ cresce esponenzialmente con n. Nell'analisi di algoritmi raggrupperemo tutti questi casi in uno solo senza preoccuparci di conoscere la forma esatta della soluzione, e studieremo invece con precisione le ricorrenze che hanno soluzioni polinomiali.

Questa posizione è giustificata dal fatto che tutti gli algoritmi di costo esponenziale sono comunque inutilizzabili al crescere di n: l'analisi ci permetterà di stabilire a priori se un algoritmo progettato per un dato problema abbia questo tipo di comportamento per scartarlo e ricercarne un altro, di costo polinomiale, che risolva lo stesso problema (naturalmente quest'ultimo potrebbe non esistere, e il problema sarebbe dichiarato intrattabile). Vedremo in particolare che la soluzione è polinomiale in ordine O se tutte le costanti $a_1, ..., a_{k-1}$ sono nulle e $f(n)$ è polinomiale in ordine O; altrimenti la soluzione è certamente esponenziale.

Constatiamo anzitutto che, se $a_1 = ... = a_{k-1} = 0$, $a_k = 1$, la relazione risultante:

$$T(n) = T(n-k) + f(n), \quad k \geq 1 \tag{14}$$

ha soluzione: $T(n) = T(n-k)+f(n) = T(n-2k)+f(n-k)+f(n) = ... = O(nf(n))$ ed è quindi polinomiale in ordine O se anche $f(n)$ è polinomiale in ordine O. Vedremo sotto che se almeno una delle costanti $a_1, ..., a_{k-1}$ è $\neq 0$, o se $a_k > 1$, la soluzione è certamente esponenziale, quindi la relazione (14) è l'unica che possa avere risultato polinomiale. Possiamo però raffinare questo valore. Per ogni $e \geq 0$ costante (non necessariamente intera) poniamo:

Teorema 4. Per ogni $n \geq k$ la ricorrenza:

$$T(n) = T(n-k) + \Theta(n^e), \quad \text{con } T(i) = c_i \text{ per } 0 \leq i \leq k-1, \tag{15}$$

ha soluzione: $T(n) = \Theta(n^{e+1})$.

Dimostrazione. Notiamo anzitutto che vale la relazione: $\int_0^m x^e dx < \sum_{x=1}^m x^e < \int_0^m x^e dx + m^e$, da cui risulta immediatamente:

$$\textstyle\sum_{x=1}^m x^e = \Theta(m^{e+1}). \tag{16}$$

Poniamo $n = km+r$ con r costante intera, $0 \leq r \leq k-1$ (quindi $m \geq 1$ intero). Nella parte $\Theta(n^e)$ della relazione (15) consideriamo solo il termine di grado massimo n^e, ottenendo una nuova ricorrenza $T^*(n) = T^*(n-k)+n^e = T^*(n-2k)+(n-k)^e+n^e = ... = T(r) + \sum_{i=0}^{m-1}(n-ik)^e = c_r + \sum_{i=0}^{m-1}(k(m-i)+r)^e = c_r + \sum_{i=1}^{m}(ki+r)^e$. Posto $S = \sum_{i=1}^m i^e$ abbiamo quindi: $c_r + k^e S \leq T^*(n) < c_r + (k+1)^e S$, e, dalla relazione (16), $T^*(n) = \Theta(k^e m^{e+1}) = \Theta(n^{e+1})$. Poiché il calcolo di $T^*(n)$ è stato eseguito per il termine di grado massimo della funzione che appare in $T(n)$, quest'ultima è dello stesso ordine Θ di $T^*(n)$. Q.E.D.

Consideriamo per esempio di ordinare n elementi contenuti in un vettore V eseguendo una scansione di V per trovare la posizione del massimo; scambiare poi questo con l'elemento in ultima posizione; ripetere ricorsivamente l'algoritmo sui primi $n-1$ elementi finchè il sottovettore su cui si ricerca il massimo diviene vuoto. La ricorrenza che esprime il tempo di funzionamento è: $T(n) = T(n-1) + \Theta(n)$ con $T(0) = c$, e si applica il teorema 4 con $k = 1$, $e = 1$ ottenendo: $T(n) = \Theta(n^2)$.

Consideriamo ore il seguente:

Lemma 1. Per ogni $n \geq k$ la ricorrenza:

$$T(n) = 2T(n-k), \quad \text{con } T(i) = c_i \text{ per } 0 \leq i \leq k-1, \tag{17}$$

ha soluzione: $T(n) = \Theta(2^{n/k})$.

Dimostrazione. Poniamo $n = km + r$ con $0 \leq r \leq k-1$. Abbiamo:

$$T(n) = 2T(n-k) = 4T(n-2k) = ... = 2^i T(n-ik) = ... = 2^m T(n-mk) = c_r 2^m.$$

Poichè $2^{km} \leq 2^n < 2^{k(m+1)}$, segue $T(n) = \Theta(2^{n/k})$. Q.E.D.

Ricordando che k è costante e n è crescente, il lemma 1 indica che $T(n)$ cresce esponenzialmente con n. Notiamo inoltre che per ogni funzione $T(n)$ espressa dalla ricorrenza (13) con almeno una delle $a_i \neq 0$, $0 \leq i \leq k-1$, o con $a_k > 1$, si ha: $T^*(n) = 2T^*(n-k) \leq T(n)$ ponendo opportune condizioni iniziali. Dunque dal lemma 1 discende immediatamente:

Teorema 5. Ogni relazione di ricorrenza di ordine k con almeno una delle $a_i \neq 0$, $0 \leq i \leq k-1$, o con $a_k > 1$, ha soluzione esponenziale in n. Q.E.D.

Consideraimo il notissimo esempio del calcolo dei numeri di Fibonacci definiti dalla relazione: $F_n = F_{n-1} + F_{n-2}$, con $F_0 = 0, F_1 = 1$. La relazione stessa può essere trasformata direttamente in un algoritmo ricorsivo di calcolo il cui tempo di funzionamento è dato da: $T(n) = T(n-1) + T(n-2)$, con $T(0) = c_0$, $T(1) = c_1$, che ha soluzione esponenziale per il teorema 5. Se gli stessi numeri si calcolano in sequenza per valori crescenti dell'indice, cioè come: $F_2 = 0 + 1 = 1$, $F_3 = 1 + 1 = 2$, $F_4 = 2 + 1 = 3$, $F_5 = 3 + 2 = 5$, ..., l'algoritmo richiede tempo lineare. Lo stesso problema si riscontra per esempio per l'algoritmo BASKET del paragrafo 1, che l'espressione (4) indica come inutilmente esponenziale: anche per quel problema la formulazione ricorsiva dell'algoritmo è estremamente dannosa e il teorema 5 ci permette di rendercene immediatamente conto.

4 Bibliografia.

[1] J.L. Bentley, D. Haken, J.B. Saxe. A general method for solving divide-and-conquer recurrences. *SIGACT News* 12-3 (1980) 36-44.

[2] T.H. Cormen, C.E. Leiserson, L.R. Rivest. *Introduction to Algorithms. Cap. IV.* MIT Press, Cambridge 1990.

[3] J. Erickson. *Algorithms notes and exercises - Solving recurrences.* Disponibile su: http://compgeom.cs.uiuc.edu/ jeffe/.

[4] R.L. Graham, D.E. Knuth, O. Patashnik. *Concrete Mathematics.* Addison-Wesley, Reading 1989.