Introduzione alla Complessità

corso regionale di preparazione per le OII

Giorgio Audrito

Università degli Studi di Torino

30 Settembre 2010

È meglio il mio algoritmo o il tuo?

Tra due algoritmi che risolvono lo stesso problema, viene considerato migliore quello che consente di risolvere casi più complessi.
Poiché vogliamo confrontare gli algoritmi e non le macchine, cerchiamo una misura indipendente dalla potenza della macchina su cui viene eseguito.

Quali sono i casi più complessi?

- I casi più "grandi",
- dove la "grandezza" di un caso è la dimensione (p. es. in *byte*) del file di input che lo descrive: $|I| = n$

È meglio il mio algoritmo o il tuo?

La quantità di memoria e di tempo richiesta da un algoritmo normalmente cresce al crescere di n.

Definizione

- Chiamiamo $T(n)$ il tempo entro il quale l'algoritmo riesce a risolvere una qualunque istanza di dimensione n.
- Chiamiamo $S(n)$ la quantità di memoria entro la quale l'algoritmo riesce a risolvere una qualunque istanza di dimensione n.

Le funzioni $T(n)$ e $S(n)$ sono dette *complessità* temporale e spaziale dell'algoritmo.

È meglio il mio algoritmo o il tuo?

Obbiettivi

- Vogliamo ottenere una misura approssimata e indipendente dalla macchina considerata.
- Un algoritmo può in pratica risolvere un istanza fin tanto che il tempo $T(n)$ e la memoria $S(n)$ di cui ha bisogno sono disponibili sulla macchina considerata.
- Per capire quanto buono è un algoritmo, ci interessa come si comportano $T(n)$ e $S(n)$ solo quando n è molto grande.
- Cambiando macchina, il tempo e la memoria disponibile cambiano di un fattore costante (per esempio, raddoppiano o dimezzano), ma non vogliamo che la misura della "bontà" dell'algoritmo dipenda dalla macchina considerata quindi ignoriamo i coefficienti moltiplicativi.

È meglio il mio algoritmo o il tuo?

Definizione

Diciamo che $T(n) = O(f(n))$ se per $n \to \infty$ $T(n)$ è al massimo proporzionale a $f(n)$.
In formule:

$$T(n) = O(f(n)) \iff \lim_{n \to \infty} \frac{T(n)}{f(n)} = c < \infty$$

Ovviamente vale la stessa definizione anche per $S(n)$.

È meglio il mio algoritmo o il tuo?

Esempi di complessità

- $\log_{13} n = O(\log n)$ – complessità logaritmica.
- $2n = O(n)$ – complessità lineare.
- $n = O(2n + 3)$ – complessità lineare.
- $\log n = O(n^2)$ – complessità quadratica.
- $5n^2 + n + \log n = O(n^2)$ – complessità quadratica.
- $2^n + n^{100} = O(2^n)$ – complessità esponenziale.

Stime iterative

Per calcolare la complessità in memoria $S(n)$ di un algoritmo, basta stimare quante variabili primitive sono utilizzate contemporaneamente dall'algoritmo.

Esempio

```
long long V[M][N];
int C[N];
char Inp[50];
int i, j, k;
```

Il programma precedente ha complessità $S(M, N) = O(MN + N + 53) = O(MN)$.

Stime iterative

Per calcolare la complessità in tempo $T(n)$ di un algoritmo, basta stimare quante istruzioni primitive vengono effettuate.
Se l'algoritmo è iterativo questo può essere fatto direttamente.

Esempio

```
c = 15;
for ( i=N; i>0; i-- ) {
  c *= i;
  for (j=1; j<N; j++) c += 2*j;
}
return c;
```

Il programma precedente ha complessità
$T(N) = O(2 + N(1 + (N-1) \cdot 2)) = O(N^2)$.

Stime ricorsive

Per esprimere la complessità in tempo $T(n)$ di un algoritmo ricorsivo, prima si analizza la forma della ricorsione per scrivere un'equazione per ricorrenza in $T(n)$, e poi si procede a risolvere l'equazione.

Esempio

```
int fai(int n, char c) {
  r = 10;
  r += 3*fai(n-1,c)+fai(n-1,1-c)
  return r == 13+c;
}
```

Il programma precedente soddisfa $T(n) = 2T(n-1) + O(1)$.
Si trova facilmente che la soluzione è $T(n) = O(2^n)$.

Stime ricorsive

Divide et Impera

In una classe notevole di algoritmi, comprendente tutti quelli di Divide et Impera, la funzione ricorsiva $g(n)$ richiama un certo numero a di volte sé stessa con parametro $\frac{n}{b}$.

Equazione notevole

$$T(n) = aT(\frac{n}{b}) + O(r(n))$$

La funzione ricorsiva con parametri di dimensione n richiama a volte se stessa, con parametri ogni volta di dimensione $\frac{n}{b}$, e in più impiega tempo $r(n)$ per ricombinare i risultati ottenuti.

Stime ricorsive

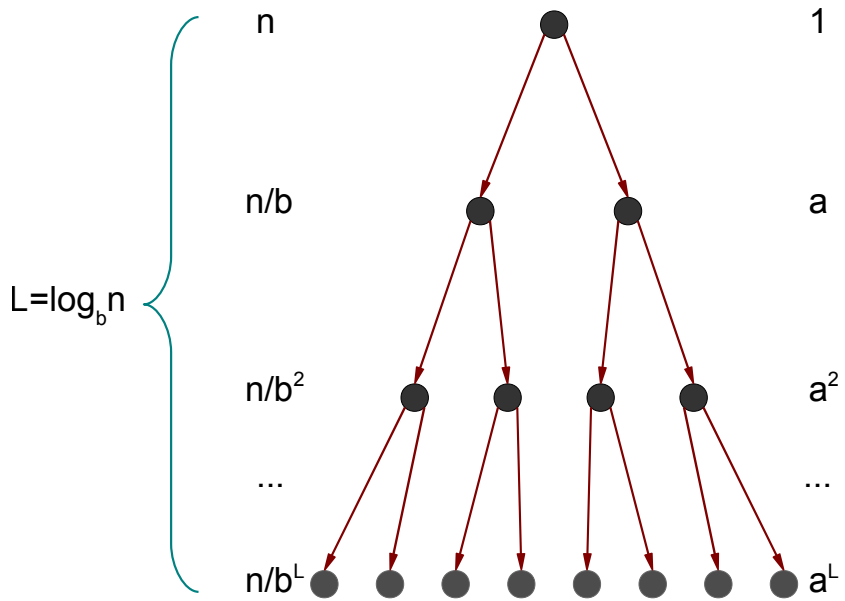
Esempi

- Potenza veloce: $T(n) = T(\frac{n}{2}) + O(1)$
- Ricerca binaria: $T(n) = T(\frac{n}{2}) + O(1)$
- Ricerca in un array non ordinato: $T(n) = 2T(\frac{n}{2}) + O(1)$
- Merge Sort: $T(n) = 2T(\frac{n}{2}) + O(n)$
- Prodotto di matrici veloce: $T(n) = 7T(\frac{n}{2}) + O(n^2)$

Per risolvere questa equazione notevole, proviamo ad analizzarne l'albero di ricorsione.

Complessità ricorsiva

Albero di ricorsione

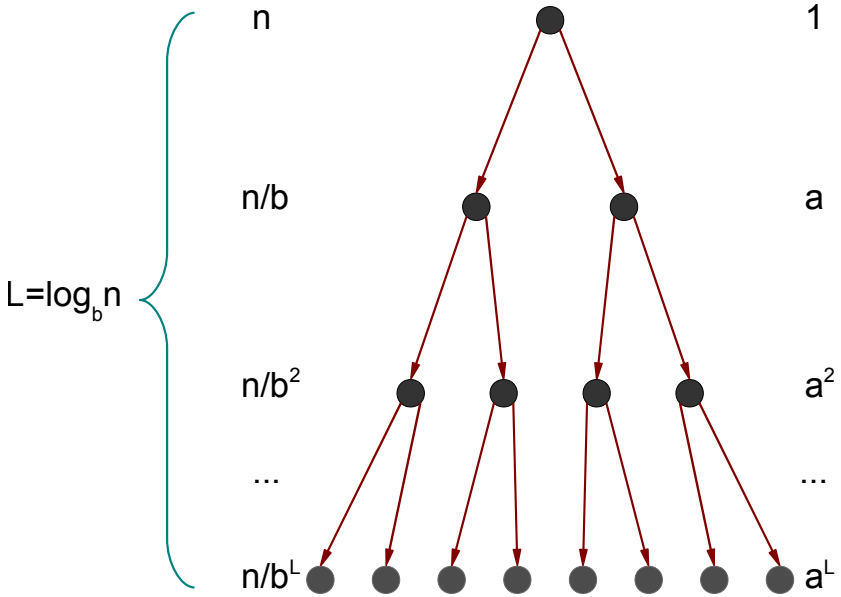


La complessità $T(n)$ dell'algoritmo è pari alla somma di:

- $O(r(n))$ – tempo impiegato nel primo livello (la *radice*).
- $O(i(n))$ – tempo impiegato nei livelli intermedi
- $O(f(n))$ – tempo impiegato nell'ultimo livello (le *foglie*)

Stime ricorsive

Albero di ricorsione

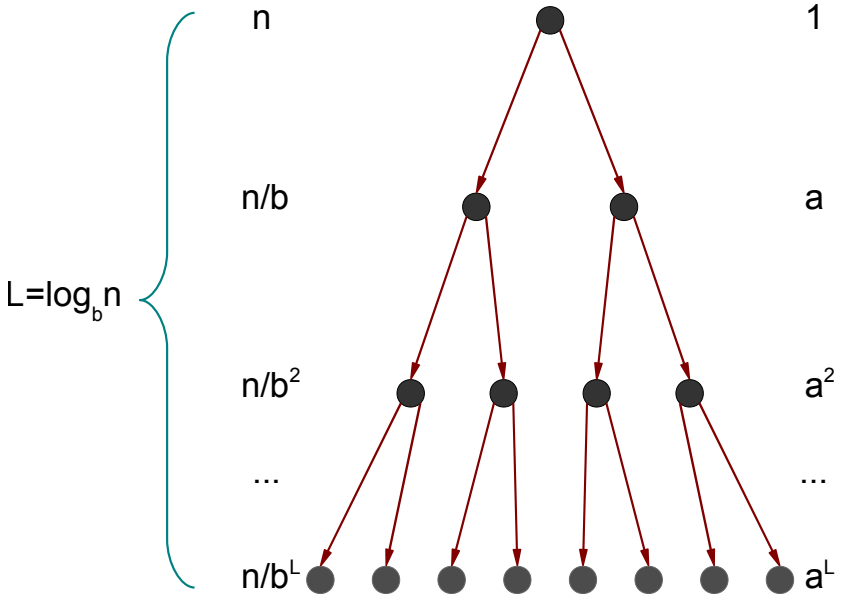


Il numero di livelli è $\frac{n}{b^L} = 1 \Rightarrow L = \log_b n$. Solitamente ogni livello intermedio impiega tempo simile a quello della radice, quindi

$$i(n) = r(n) \log n$$

Stime ricorsive

Albero di ricorsione

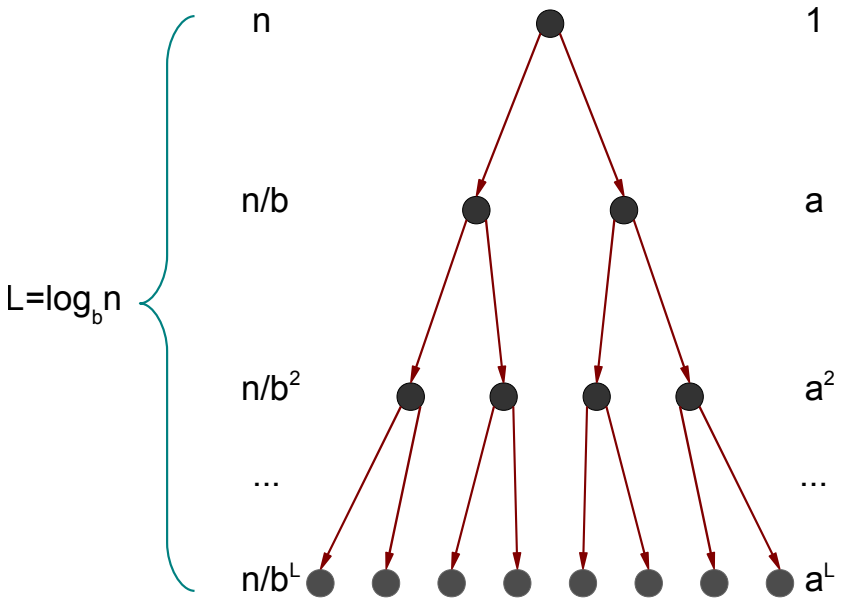


Ogni foglia richiede tempo $O(1)$, quindi $f(n)$ è pari al numero di foglie:

$$a^L = a^{\log_b n} = b^{\log_b a \log_b n} = b^{\log_b (n^{\log_b a})} = n^{\log_b a}$$

Stime ricorsive

Albero di ricorsione



In totale il tempo di esecuzione sarà quindi:

$$T(n) = O(r(n) + i(n) + f(n)) = O(r(n) \log n + n^{\log_b a})$$

Stime ricorsive

Esempi

- Potenza veloce:
  $T(n) = T(\frac{n}{2}) + O(1) = O(1 \log n + n^0) = O(\log n)$
- Ricerca binaria:
  $T(n) = T(\frac{n}{2}) + O(1) = O(1 \log n + n^0) = O(\log n)$
- Ricerca in un array non ordinato:
  $T(n) = 2T(\frac{n}{2}) + O(1) = O(1 \log n + n^1) = O(n)$
- Merge Sort:
  $T(n) = 2T(\frac{n}{2}) + O(n) = O(n \log n + n^1) = O(n \log n)$
- Prodotto di matrici veloce:
  $T(n) = 7T(\frac{n}{2}) + O(n^2) = O(n^2 \log n + n^{log_2 7}) \simeq O(n^{2.8})$

Stime ricorsive

Esempio

```
int fai(int n, int j) {
  r = j;
  r += fai(n/3,j)*fai(n/3,j+1);
  for (r=r+1; n>1; n /= 2) r += j;
  return r;
}
```

Il programma precedente soddisfa $T(n) = 2T(n/3) + O(\log n)$.
La soluzione è $T(n) = O(\log n \log n + n^{\log_3 2}) \simeq O(n^{0.63})$.