Gli algoritmi e la loro complessità

Matteo SONZA REORDA
Dip. Automatica e Informatica
Politecnico di Torino

1 a.a. 2001/2002

Algoritmo

Un algoritmo è una procedura di calcolo (eventualmente composta da un certo numero di passi) che risolve un certo problema, operando su un insieme di valori di ingresso e producendo un insieme di valori di uscita.

2 a.a. 2001/2002

Esempio

Il problema dell'ordinamento è così definito:

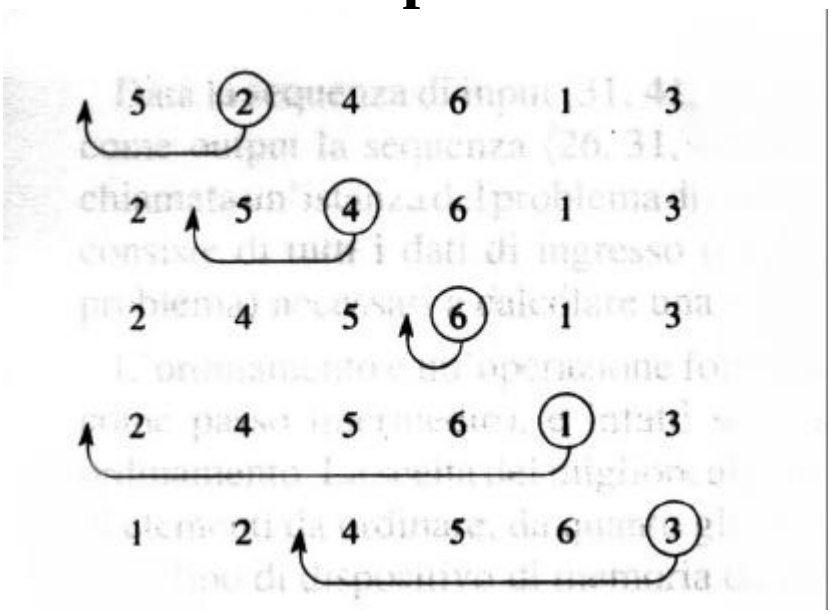
Insieme di ingresso: sequenza di numeri $<a_1, a_2, \ldots, a_n>$

Insieme di uscita: permutazione $<a'_1, a'_2, \ldots, a'_n>$ della sequenza di ingresso tale per cui $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Insertion Sort

```
INSERTION-SORT(A)
1  for j ← 2 to length[A]
2  do  key ← A[j]
3      ▷ Si inserisce A[j] nella sequenza ordinata A[1 . . j − 1]
4      i ← j − 1
5      while i > 0 e A[i] > key
6          do A[i + 1] ← A[i]
7             i ← i − 1
8      A[i + 1] ← key
```

Esempio



Analisi degli algoritmi

È l'operazione tramite la quale si tenta di prevedere la quantità di risorse (I/O, memoria, tempo) richieste dall'algoritmo per la sua esecuzione.

Tale analisi dovrebbe essere indipendente dal tipo e dalle caratteristiche della macchina che esegue l'algoritmo.

In questo corso si considereranno sempre elaboratori di tipo tradizionale (noti come Random Access Machine o RAM).

Dimensione dell'ingresso

L'analisi degli algoritmi è normalmente svolta rispetto ad uno o più parametri che caratterizzano la dimensione del problema.

Esempi

Nel caso dell'ordinamento tale parametro corrisponde al numero di valori nella sequenza di ingresso.

Nel caso del problema della moltiplicazione tra numeri interi, la dimensione dell'ingresso è data dal numero di bit su cui sono rappresentati gli operandi.

Ipotesi

- **Per eseguire ciascuna riga dello pseudocodice è richiesto un tempo costante**
- **Ciascuna riga ha un tempo di esecuzione diverso.**

Analisi dell'insertion sort

```
INSERTION-SORT(A)                                   costo   n° di volte
1  for j ← 2 to length[A]                           c1      n
2      do  key ← A[j]                               c2      n − 1
3          ▷ Si inserisce A[j] nella
           ▷ sequenza ordinata A[1 . . j − 1]       0       n − 1
4          i ← j − 1                                c4      n − 1
5          while i > 0 e A[i] > key                 c5      Σ_{j=2}^{n} t_j
6              do  A[i + 1] ← A[i]                  c6      Σ_{j=2}^{n} (t_j − 1)
7                  i ← i − 1                        c7      Σ_{j=2}^{n} (t_j − 1)
8          A[i + 1] ← key                           c8      n − 1
```


Tempo di esecuzione dell'insertion sort

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .$$

Caso migliore

Se il vettore è già ordinato, allora $t_j = 1$ per qualsiasi j.

Quindi

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \; . \end{aligned}$$

che può essere espresso come T(n) = an + b.

Caso peggiore

Se il vettore è ordinato in ordine inverso al j-esimo passo si devono eseguire j confronti e j scambi: $t_i = j$.

Si ricordi che

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

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

Caso peggiore (2)

Quindi

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + \\
&\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\
&= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n + \\
&\quad - (c_2 + c_4 + c_5 + c_8) .
\end{aligned}
$$

Nel caso peggiore si ha quindi che

$$T(n) = an^2+bn+c$$

Caso medio

L'analisi del caso peggiore è particolarmente importante in quanto fornisce un limite superiore alle risorse richiedte dall'algoritmo.

In alcuni casi può essere importante anche l'analisi del caso medio (o atteso).

Importanza dell'analisi della complessità

L'analisi delle risorse richieste da un algoritmo (anche detta analisi della complessità) permette di determinare soprattutto l'andamento di tali esigenze in funzione della dimensione del problema.

La scelta di un algoritmo con complessità inferiore è vantaggiosa indipendentemente dalla tecnologia utilizzata, e può largamente compensare la mancanza di hardware potente.

Esempio

Si supponga che per la risoluzione di un certo problema siano disponibili due algoritmi:

- **Uno con complessità $T(n) = 2n^2$**
- **Un altro con complessità $T(n) = 50n \lg_2 n$**

Si supponga di utilizzare un supercalcolatore (capace di eseguire 100 Mistruzioni/secondo) per eseguire il primo algoritmo, ed un PC (capace di eseguire 1 Mistruzioni/secondo).

I tempi per la soluzione del problema nei due casi allorché n = 1M sono

Supercalcolatore: 5,56 ore

PC: 16.67 minuti

Notazione asintotica

Per l'analisi della complessità degli algoritmi si fa spesso uso della cosiddetta notazione asintotica, ossia dell'analisi della complessità dell'algoritmo quando la dimensione dell'ingresso tende a diventare molto grande.

La notazione asintotica si basa su tre notazioni:

- **Notazione Θ**
- **Notazione O**
- **Notazione Ω.**

Notazione Θ

Sia dato un algoritmo con complessità T(n).

Si scrive che $T(n) = \Theta(g(n))$ se e solo se esistono tre costanti positive c_1, c_2 e n_0 tali per cui

$$0 \leq c_1 g(n) \leq T(n) \leq c_2 g(n)$$

per ogni $n \geq n_0$.

Si dice in questo caso che g(n) è un limite asintotico stretto per T(n).

Notazione O

Si scrive che T(n) = O(g(n)) se e solo se esistono due costanti positive c e n_0 tali per cui

$$0 \leq T(n) \leq cg(n)$$

per ogni $n \geq n_0$.

Si dice in questo caso che g(n) è un limite asintotico superiore per T(n).

Notazione Ω

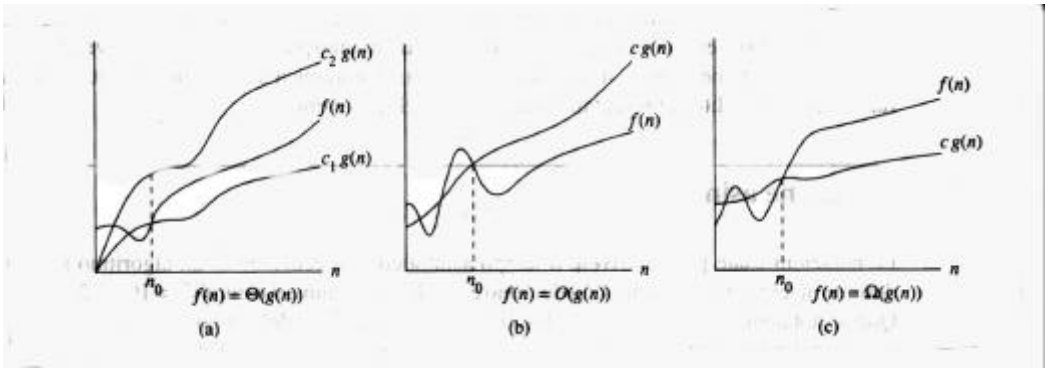
Si scrive che T(n) = Ω(g(n)) se e solo se esistono due costanti positive c e n_0 tali per cui

$$0 \leq cg(n) \leq T(n)$$

per ogni $n \geq n_0$.

Si dice in questo caso che g(n) è un limite asintotico inferiore per T(n).

Significato



Teorema

Date due funzioni g(n) e T(n), T(n) = Θ(g(n)) se e solo se

- **T(n) = O(g(n)) e**
- **T(n) = Ω(g(n)).**