18/10/2010

COMPLESSITÀ DEGLI ALGORITMI

Un algoritmo coincide con una descrizione in pseudocodice che, tradotto il linguaggio macchina (implementazione), diventa un programma. Questo programma può essere eseguito su una piattaforma opportuna con costi che variano a seconda della risorsa utilizzata: quella più critica è il tempo di calcolo. Il tempo di calcolo è influenzato da:

- la dimensione dell'input: cresce all'aumentare della dimensione dell'input;
- l'efficienza dell'algoritmo: pere ottenere un algoritmo il più efficiente possibile bisogna fare una previsione del tempo di calcolo delle implementazioni di un algoritmo (detta *analisi*).
- L'hardware; il tipo di linguaggio e compilatore;
- Il tipo di programmatore.

Analisi degli algoritmi

Gli strumenti che utilizziamo per questa analisi sono il tempo di esecuzione di ogni istruzione e l'analisi asintotica delle funzioni. In particolare si preferisce studiare il caso peggiore rispetto al caso migliore (che è di interesse puramente teorico) per avere un errore per eccesso piuttosto che per difetto, senza contare che il caso media si avvicina più al caso peggiore piuttosto che a quello migliore. Per la stima del tempo si possono adottare due strategie: una più onerosa, nel quale si parte dallo pseudocodice studiando il comportamento asintotico del tempo di calcolo dell'implementazione, e una strategia più efficiente in cui viene studiato il comportamento asintotico di ogni istruzione dello pseudocodice e mettendoli insieme.

esempio di calcolo di T(n)

INVERTI-ARRAY (A)	costo	n° di volte
1. **for** i = 0 **to** A.length-2 **do**	c_1	n
2. memo = A[i]	c_2	n-1
3. A[i] = A[A.length-1]	c_3	n-1
4. **for** j = A.length-1 **down to** i+2 **do**	c_4	(n-1)n/2
5. ▷ traslo in avanti A[i+1..]	0	(n-1)n/2-1
6. A[j] = A[j-1]	c_6	(n-1)n/2-1
7. A[i+1] = memo	c_7	n-1

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1)n/2 + c_6((n-1)n/2-1) + c_7(n-1)$$
$$= n^2(c_4/2+c_6/2) + n(c_1+c_2+c_3-c_4/2-c_6/2+c_7) - (c_2+c_3+c_6+c_7)$$
$$T(n) = O(n^2)$$

Righe:

1) il test viene eseguito n volte (n−1 + l'ultima per uscire dal ciclo);

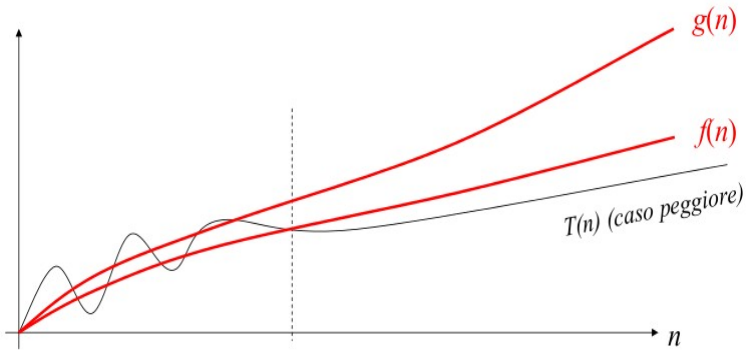
2)3)7) esecuzione per ogni ciclo di for (n−1);

4) il test viene eseguito per ogni i = 0, …, n−2 -> $\Sigma(n-1) \rightarrow n(n-a)/2$

5)6) lo stesso tranne −1.

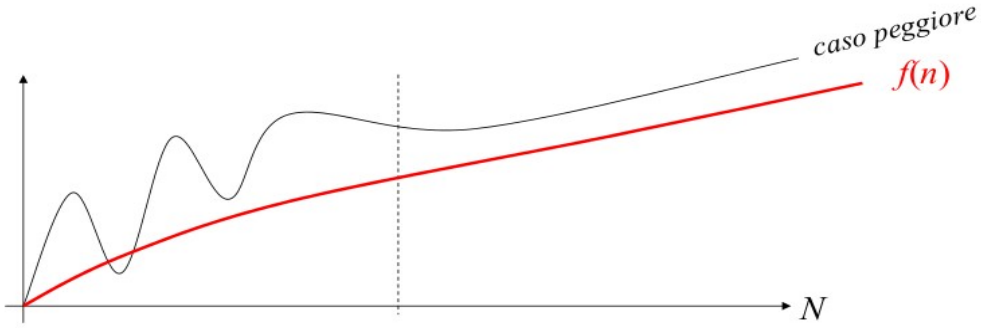
Algoritmi e complessità O(f(n))

L'algoritmo ha complessità temporale O(f(n)) se il tempo di esecuzione di esso è uguale alla complessità. Il tempo di esecuzione dell'algoritmo è al più f(n) (limite superiore). Se un algoritmo ha complessità f(n), allora ha anche la complessità g(n) per ogni g(n) tale che f(n) ∈ O(g(n)). La complessità espressa tramite la notazione O-grande diventa tanto più significativa quanto più f(n) è piccolo.



Algoritmi e complessità Ω(f(n))

L'algoritmo ha complessità temporale Ω(f(n)) se il tempo di esecuzione di esso è uguale alla complessità. Il tempo di esecuzione dell'algoritmo è al più f(n) (limite inferiore).



Algoritmi e complessità Θ(f(n))

L'algoritmo ha complessità temporale Θ(f(n)) se ha complessità temporale O(f(n)) e Ω(f(n)). Il tempo di esecuzione è f(n), che corrisponde sia al limite inferiore che a quello superiore; è la quantità di tempo sufficiente e necessaria per l'esecuzione dell'algoritmo.

Calcolo efficiente del costo asintotico

- **istruzioni semplici**: tempo di esecuzione costante $O(1)$;
- **sequenza di istruzioni semplici**: tempo di esecuzione costante $O(1)$;
- **sequenza di istruzioni generiche**: somma tempi esecuzione di ciascuna istruzione;
- **istruzioni condizionali**: somma dei costi tra la valutazione della condizione e il costo massimo tra <parte-then> e <parte-else>;
- **istruzioni ripetitive**: occorre determinare un limite superiore $O(f(n))$ al numero di iterazione del ciclo e uno al tempo di esecuzione di ogni iterazione; il costo del ciclo sarà $O(g(n)*f(n))$.

ESEMPIO

```
FACT(n)
1. f = 1
2. k = n
3. while k > 0
4.     do f = f * k
5.        k = k - 1
6. return f
```

Il numero di iterazioni del ciclo while è $O(n)$; il costo di una singola iterazione è $O(1)$. Il costo complessivo del ciclo while è $O(n*1) = O(n)$, mentre il costo complessivo della procedura è $O(n)$. Se la misura dell'input è il numero di bit k necessari per rappresentare n in binario avremmo $k=\lceil \log_2 n \rceil$ e costo complessivo $O(2^k)$.

Chiamata a funzione o procedura

Supponiamo che un programma P invochi una procedura Q con complessità $T_q(n) \in O(f_n))$.

ESEMPIO

```
SUM-OF-FACT(n)
1.  sum = 0
2.  m = n
3.  while m > 0
4.      do  sum = sum + FACT(m)
5.          m = m - 1
6.  return sum
```

Il corpo del ciclo while ha complessità $O(1) + O(m) = O(m)$; il ciclo viene eseguito n volte per i valori di m da n a 1 $(O(n) + O(n) + \dots + O(n))$:

$O(m) = O(n) + O(n-1) + \dots + O(2) + O(1)$.

Il costo complessivo è $O(n) + O(n-1) + \dots + O(2) + O(1) = (O(n) + O(n) + \dots + O(n) = n*O(n) = O(n^2)$.