Algoritmica 07/08
15. ANALISI AMMORTIZZATA

Abbiamo visto come la complessità di un algoritmo si studi in genere al **caso pessimo** tra tutte le distribuzioni possibili dei dati in ingresso, a pari loro dimensione n; o possa a volte essere studiata al **caso medio** tra tutte quelle distribuzioni, se è nota la probabilità con cui esse si presentano: caso questo del QUICKSORT randomizzato ove abbiamo ammesso che tutte le permutazioni dei dati d'ingresso siano equiprobabili. Esiste però un'altro tipo di analisi applicabile solo in determinate circostanze, nelle quali è più significativa delle precedenti.

Fissata una struttura di dati di dimensione n, si considera una catena di k operazioni su essa e si considera il tempo medio complessivo degli algoritmi impiegati per le diverse operazioni: posto che ognuno di questi algoritmi si trovi, quando richiamato, di fronte al caso pessimo. Si prende cioè, come complessità in tempo, la somma dei tempi di caso pessimo delle operazioni della catena e si divide tale somma per k. Il tempo così calcolato deve però tener conto che, per alcuni problemi, i dati non possono sempre presentarsi nelle condizioni più sfavorevoli per k volte consecutive, e si deve studiare l'effetto sulla sequenza peggiore di tutte. Questa analisi, detta **ammortizzata** (su k operazioni), e dunque un'analisi di caso pessimo sulla catena e non ha alcuna connotazione probabilistica.

Studieremo qui l'argomento su alcuni semplicissimi esempi. Per un'esposizione generale di questo metodo di analisi si vedano i testi C par. 3.4.3, e D par. 2.7.

k ricerche in un vettore. Dato un vettore A[0..n-1] contenente una permutazione arbitraria di n elementi, si devono eseguire su esso k operazioni di ricerca.
Metodo 1. Si eseguono k ricerche sequenziali a partire da A[0]. Nel caso pessimo ciascuna di esse richiede n passi (si cerca sempre l'ultimo elemento). Il tempo complessivo è $\Theta(kn)$, il tempo ammortizzato è $\Theta(kn/k) = \Theta(n)$.
Metodo 2. Si ordina il vettore in tempo $\Theta(n \log n)$; poi si eseguono su di esso k ricerche binarie. Nel caso pessimo ogni ricerca richiede log n passi (in caso non pessimo la ricerca binaria si ferma prima). Il tempo complessivo è $\Theta(n \log n + k \log n)$, il tempo ammortizzato è $\Theta(((n+k)\log n)/k)$. Si vede immediatamente che questo tempo è $\Theta(\log n)$ per $k=\Omega(n)$, è $\Theta(n)$ per $k=\Theta(\log n)$, è $\Theta(n \log n)$ per $k=\Theta(1)$.
Dunque se si devono eseguire "poche" ricerche conviene il metodo 1, per "molte" ricerche conviene il metodo 2, nei casi intermedi i due metodi possono essere equivalenti ($k=\Theta(\log n)$).

Tabelle di dimensioni variabili. Una tabella su cui si eseguono operazioni di inserzione e cancellazione di elementi è memorizzata in un vettore A[0..n-1]. Se richiesto, dopo un'operazione si procede all'adeguamento delle dimensioni del vettore in modo che questo sia sempre pieno per più di un quarto. Non è qui rilevante

conoscere gli algoritmi di inserzione e cancellazione di elementi, ma solo ammettere che richiedano tempo minore di n in ordine di grandezza: per semplicità ammettiamo che richiedano entrambi tempo costante. Per cambiarne le dimensioni, A si copia in un vettore di appoggio richiedendo quindi n passi. Le regole sono:
1. Se A è pieno e si vuole inserire un nuovo elemento, si raddoppia la dimensione di A che passa da n a 2n.
2. Se, dopo aver cancellato un elemento, A è pieno per n/4, si dimezza la dimensione di A che passa da n a n/2.
Le operazioni 1 e 2 costituiscono il caso pessimo per inserzione e cancellazione perché richiedono tempo $\Theta(n)$. In una catena di k operazioni, però, una volta presentatosi il caso 1 o 2 accorrono $\Theta(n)$ passi perché uno di questi possa ripresentarsi. Dunque il caso pessimo è che per ogni $k=\Theta(n)$ operazioni di tempo $\Theta(1)$ si richieda una operazione di tempo $\Theta(n)$, ottenendo il tempo ammortizzato: $(\Theta(1\times k)+ \Theta(n\times 1))/k = \Theta(1)$.

Incremento di un contatore binario. Un contatore di n bit viene incrementato dal valore 0 = 00...0 al valore 2^n = 11...1. Costo dell'operazione è il numero di bit cambiati a ogni passo. Per esempio per n=5:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1...
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0...
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0...
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1...
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0...

Il caso pessimo si ha quando gli n-1 bit meno significativi sono tutti 1, come per esempio nel passaggio tra il valore 15 = 01111 al valore 16 = 10000: in questo caso cambiano tutti gli n bit. Tale caso però si presenta solo una volta nella sequenza di operazioni. È facile vedere che una volta su due cambia un bit; una volta su quattro cambiano due bit; una volta su otto cambiano tre bit; ecc. Detto $N = 2^n$, il numero totale di bit cambiati è dato dalla somma:

$$S = 1N/2 + 2N/4 + 3N/8 + 4N/16 + ...$$
$$= N\{(1/2+1/4+1/8+1/16...) + (1/4+1/8+1/16 ...) + (1/8+1/16...)$$
$$\leq N(1 + 1/2 + 1/4 + 1/8...) = 2N.$$

Il costo per operazione, ammortizzato su k=N operazioni, è quindi $S/N \leq 2$.