Fabrizio Luccio. Informatica per le Biotecnologie

4. Il problema dell'ordinamento

Il problema dell'ordinamento di dati (*sorting* in gergo informatico) consistente nel disporre in ordine "crescente" n elementi tra cui è definita una relazione di ordinamento totale indicata con $\leq$.

Memorizziamo gli elementi da ordinare in un vettore$A[0...n-1]$. Un algoritmo iterativo elementare detto INSERTION-SORT (ordinamento per inserzione), si basa sull'ipotesi che i primi i elementi contenuti nella porzione del vettore tra le posizioni 0 e $i-1$ siano ordinati. Si inserisce l'elemento $A[i]$ tra questi nella posizione che gli compete, ottenendo cosí l'ordinamento fino alla posizione i, e si ripete l'operazione a partire da $i+1$ finché tutti gli n elementi sono stati inseriti. Si inizia con $i=1$ (anziché $i=0$) poiché un sottoinsieme di un solo elemento $A[0]$ si considera ordinato.

```
INSERTION-SORT(A)
// Ordinamento iterativo di un vettore A[0...n − 1].
    for (i = 1; i ≤ n − 1; i++) {
        { k = A[i] ;
// ora si inserisce k nella posizione opportuna della sequenza A[0...i] che è ordinata
fino ad A[i − 1] confrontandolo k con gli elementi per valori decrescenti dell'indice
a partire da A[i − 1].
            j = i − 1;
            while ((j ≥ 0) && (A[j] > k)) {A[j + 1] = A[j]; j = j − 1};
// dopo l'istruzione A[j + 1] = A[j] le due celle contengono lo stesso valore; la A[j]
può essere quindi riutilizzata e si riparte con j = j − 1.
            A[j + 1] = k ; }
```

Questa procedura è riportata in forme diverse, ma tutte equivalenti, nei libri di algoritmica.

Esercizio 1. Simulare il funzionamento della procedura INSERTION-SORT su un breve vettore scelto a piacere per capire come funziona e studiare i casi ottimo (i dati in ingresso sono già ordinati) e pessimo (i dati sono ordinati in modo decrescente), verificando che nei due casi la procedura richiede rispettivamente tempo $\Theta(n)$ e $\Theta(n^2)$.

Un metodo alternativo per il sorting utilizza il paradigma ricorsivo in cui una procedura su dati di dimensione n chiama sé stessa su sottoinsiemi di tali dati finché questi raggiungono una dimensione costante (cioè indipendente da n) per cui il problema è risolto in modo diretto. Come sappiamo tale formulazione è spesso la più naturale per affrontare un problema, consente di dimostrare per induzione la correttezza del programma e si presta a uno studio matematico della complessità in tempo; ma diviene in genere difficile comprendere la sequenza di operazioni effettivamente

eseguita dal programma. Un algoritmo ricorsivo noto come MERGE-SORT (ordinamento per fusione) è il seguente: oltre al vettore A, in cui i dati vengono presentati e riordinati man mano, è necessario un vettore di appoggio $B[0...n-1]$.

MERGE-SORT(A, s, d)
// Ordinamento di un vettore A di n elementi tra le posizioni s,d. Nella chiamata iniziale $s = 0, d = n-1$. *Risulterà sempre* $s \leq d$.

if $(s < d)$ {

$m = \lfloor (s+d)/2 \rfloor$; *// m indica il punto medio tra s,d*

MERGE-SORT(A, s, m); MERGE-SORT$(A, m+1, d)$; FUSIONE(A, s, m, d);
}

FUSIONE(A, s, m, d)
// Fusione dei due sottovettori ordinati $A[s...m], A[m+1...d]$ *usando un vettore di appoggio B, per ottenere un sottovettore* $A[s...d]$ *completamente ordinato.*

$i = s;\ j = m+1;\ k = 0;$

while $((i \leq m)$ && $(j \leq d))$ {

if $(A[i] \leq A[j])$ $\{B[k] = A[i];\ i = i+1\}$

else $\{B[k] = A[j];\ j = j+1\}$;

$k = k+1$ };

// Ora se il secondo sottovettore di A è stato esaurito prima del primo sottovettore, gli elementi rimasti nel primo si trasferiscono nelle ultime posizioni del secondo.

if $(i \leq m)$

$\{j = d-(m-i)$; for $(\ ;\ i \leq m;\ i++;\ j++)\ A[j] = A[i]\}$;

// Ora si trasportano in A, a partire dalla posizione s, gli elementi posti in B che sono in totale k.

$i = s;\ j = 0;$ for $(\ ;\ j \leq k-1;\ i++;\ j++)\ A[i] = B[j];$

Anche questa procedura è riportata in forme diverse, ma tutte equivalenti, nei libri di algoritmica.

Esercizio 2. Esaminare attentamente la procedura FUSIONE per capire come funziona, eseguendone a mano una simulazione su un piccolo sottovettore (per esempio per $s = 5$ e $d = 8$). Convincersi che richiede tempo $\Theta(d-s)$, quindi tempo $\Theta(n)$ quando lavora sul vettore intero.

Esercizio 3. Eseguire a mano una simulazione di MERGE-SORT su un vettore di otto elementi per comprendere i passi eseguiti da un algoritmo ricorsivo. A tale proposito dare per aquisito che FUSIONE fonde due sottovettori ordinati per formarne uno complessivo, senza ripeterne la simulazione già vista nell'esercizio 2.

Per calcolare il tempo $T(n)$ richiesto da MERGE-SORT notiamo che:

- per $n = 1$ (cioè per $s = d$) la procedura richiede tempo costante b per eseguire il test "if($s < d$)", poi termina: abbiamo dunque $T(1) = b$;
- per $n > 1$ (cioè per $s < d$) la procedura richiede tempo costante c_1 per eseguire il test e calcolare m; poi chiama due volte sé stessa su $n/2$ dati; infine esegue FUSIONE in tempo $\leq c_2 n$ per un opportuna costante c_2 (vedi esercizio 2): abbiamo dunque $T(n) = 2T(n/2) + c_1 + c_2 n \leq 2T(n/2) + cn$ inglobando la costante c_1 nel termine lineare cn, per un opportuno valore costante $c > c_2$.

In conclusione la funzione $T(n)$ soddisfa la seguente *equazione di ricorrenza*:

$$T(1) = b; \quad T(n) \leq 2T(n/2) + cn, \quad \text{per } n > 1;$$

con b, c costanti. Sviluppando con la stessa formula it termine $T(n/2)$ e proseguendo allo stesso modo per i termini via via generati abbiamo $T(n/2) \leq 2T(n/4) + cn/2$, $T(n/4) \leq 2T(n/8) + cn/4$, ecc. Ponendo per semplicità di calcolo che n sia una potenza di 2, cioè $n = 2^t$ e quindi $t = \log_2 n$, otteniamo lo sviluppo:

$$\begin{aligned} T(n) &\leq 2T(n/2) + cn \leq 2(2T(n/4) + cn/2) + cn = 4T(n/4) + 2cn \\ &\leq 4(2T(n/8) + cn/4) + 2cn = 8T(n/8) + 3cn = 2^3 T(n/2^3) + 3cn \\ &\leq .... = 2^t T(n/2^t) + tcn = nT(1) + cn \log_2 n = cn \log_2 n + bn, \end{aligned}$$

ovvero $\boldsymbol{T(n)}$ è di ordine $\mathbf{O(n \log n)}$. Dunque MERGE-SORT è molto più efficiente di INSERTION-SORT che richiedeva tempo quadratico nel caso pessimo (si noti che la funzione $n \log n$ è assai più prossima a n che a n^2).[1]

Vediamo ora di stabilire un limite inferiore alla complessità in tempo del problema determinando il numero minimo di confronti $min(n)$ che devono essere necessariamente eseguiti per ordinare un insieme. Come discusso per il problema della ricerca di un dato in un insieme, $min(n)$ costituisce un limite inferiore: se si potesse dimostrare che altre operazioni, diverse dal confronto, devono essere eseguite in numero maggiore di quello dei confronti si potrebbe stabilire un limite inferiore più alto e quindi più significativo: vedremo però che il limite sul numero di confronti è sufficiente per l'analisi del sorting.

Utilizziamo l'albero di decisione già illustrato per il problema della ricerca. Partendo dalla sequenza iniziale dei dati nel vettore, le $S(n)$ possibili soluzioni del problema sono rappresentate dalle permutazioni di tale sequenza: infatti la sequenza ordinata che si vuole ottenere è una delle possibili permutazioni della sequenza originale, che sono $n!$. Utilizziamo la *formula di Stirling* che dà un'approssimazione della funzione fattoriale: $S(n) = n! \approx \sqrt{2\pi n}(n/e)^n$ (vedi dispensa 2.2). Allocando

[1] Nel caso generale in cui n non sia una potenza di due, ovvero $2^{t-1} < n < 2^t$, lo studio di complessità si può eseguire immaginando di allungare il vettore A portandolo a $A[0...2^t - 1]$, allocare i dati da ordinare nelle celle da 0 a $n-1$ e allocare un valore ∞ (cioè il massimo contenibile in una cella di A) in tutte le celle da n a $2^t - 1$. Applicando MERGE-SORT al nuovo vettore si ottiene nelle prime n celle di A l'ordinamento voluto e il calcolo della complessità genera il medesimo risultato in ordine di grandezza.

queste soluzioni nelle foglie di un albero di decisione ternario che rappresenta percorsi di computazione di lunghezza massima t abbiamo $3^t \geq \sqrt{2\pi n}(n/e)^n$. Applicando il logaritmo a base 3 ai due membri della diseguaglianza otteniamo:

$$t \geq (1/2)\log_3(2\pi n) + n(\log_3 n - \log_3 e) > n(\log_3 n - \log_3 e)$$

ove il termine $\log_3 n$ è prevalente rispetto al termine costante $\log_3 e$, e dunque t è di ordine $\Omega(n \log n)$.

Ricordando che, nel caso pessimo, t rappresenta il numero di confronti successivi uno all'altro nella computazione più lunga, abbiamo $min(n) = t$, dunque il limite inferiore alla complessità del sorting è $\Omega(n \log n)$. Possiamo perciò concludere che MERGESORT è un **algoritmo ottimo** (mentre INSERTION-SORT non lo è).