INFORMATICA GENERALE - I-Z

Claudia Malvenuto - Daniele A. Gewurz
Scheda esercizi n. 3

1. Se con $\lfloor x \rfloor$ indichiamo la parte intera inferiore di x, cioè il massimo intero m che sia minore o uguale di x, dimostrare che, presi comunque tre numeri naturali a, b e n maggiori di 0, si ha sempre:
$$\lfloor \lfloor n/a \rfloor / b \rfloor = \lfloor n/ab \rfloor.$$
(Questa uguaglianza è utile perché garantisce che se, nel calcolo del tempo di esecuzione di un algoritmo, si approssima $\lfloor n/a \rfloor$ con n/a in iterazioni successive, l'errore che si accumula rimane limitato.)

2. Svolgere gli esercizi 4.3-1, 4.3-2 e 4.3-3 a pag. 73 del Cormen (risoluzione di alcune ricorrenze fondamentali).

3. Dati due array A e B composti da n numeri ognuno, l'algoritmo ricorsivo qui riportato determina se A e B hanno almeno un elemento in comune.

Algoritmo 1 elementoInComune(array A, B; interi n, i, j) $\rightarrow$ booleano

```
if i = j then
    for k = 0 to n − 1 do
        if A[k] = B[i] then
            return true
        end if
    end for
    return false
else
    m ← ⌊(i + j)/2⌋
    if elementoInComune (A, B, n, i, m) then
        return true
    else
        return elementoInComune(A, B, n, m + 1, j)
    end if
end if
```

La chiamata iniziale è elementoInComune(A, B, n, 0, $n-1$) (assumendo che gli indici degli array vadano da 0 a n). Qual è il tempo di esecuzione del passo base? Qual è il tempo di esecuzione totale? Si discutano il caso migliore e quello peggiore, assumendo che il passaggio di parametri costi tempo $O(1)$.

(Suggerimento: Si consideri il tempo $T(n, b)$ di esecuzione dell'algoritmo, in funzione della lunghezza n dell'array e della lunghezza $b = j - i + 1$ della porzione di array considerata in una generica chiamata dell'algoritmo.)

4. Scrivere un algoritmo ricorsivo in pseudocodice che, dato un vettore ordinato A, vi cerchi un elemento usando la *ricerca ternaria*. Mentre nella ricerca binaria ad ogni chiamata la ricerca

prosegue in uno solo di due sottovettori, nella ternaria la ricerca deve proseguire in uno solo di tre sottovettori. Studiare il tempo di esecuzione dell'algoritmo nel caso peggiore.

5. Supponiamo di avere due algoritmi; il tempo di esecuzione di uno, in funzione di un parametro n è definito ricorsivamente da $T(n) = 5T(n/6) + 2n^2$; quello dell'altro è dato da $T(n) = 7T(n/6) + n$. In entrambi casi il tempo di esecuzione del caso base è $O(1)$. Per n sufficientemente grande, quale dei due algoritmi è più veloce?

6. Sia A un vettore di lunghezza n. Si considerino le seguenti funzioni in pseudo-codice:

Algoritmo 2 Algo1(array A; interi i, j)

```
k ← i
while k ≤ j do
    h ← i
    while h ≤ j do
        Istr
        h ← h + 1
    end while
    k ← k + 1
end while
```

Algoritmo 3 Algo2(array A; interi i, j)

```
if i < j then
    n ← j − i + 1
    Algo2(A, i, i + n/4)
    Algo2(A, j − n/4, j)
    Algo1(A, i, j)
end if
```

Nella chiamata iniziale di Algo2, si ha $i = 1$ e $j = n$. Sapendo che Istr è un blocco di istruzioni il cui tempo di esecuzione è O(1), studiare il tempo di esecuzione di Algo2 nel caso peggiore. Per semplicità di analisi, è possibile non considerare gli effetti dovuti agli arrotondamenti (parti intere inferiori e superiori).

7. Dato un vettore contenente numeri interi (positivi e negativi), il problema del *massimo sottovettore* chiede di trovare quale sia la massima somma possibile dei componenti di un sottovettore composto da elementi consecutivi. Più formalmente, dato un vettore A di lunghezza n (con indici da 0 a $n-1$), si cerca il massimo di $\sum_{i=k}^{l} A[i]$ (detto valore del sottovettore) al variare di k e l. Scrivere lo pseudocodice e studiare il tempo di esecuzione per alcuni possibili algoritmi per risolvere questo problema:

   (a) l'algoritmo che per "forza bruta" prende in esame con due cicli annidati ogni possibile sottovettore, per ognuno calcola il valore sommando - con un ciclo - tutti i termini del sottovettore e confronta questo valore con il valore più alto trovato fino ad allora;

   (b) l'algoritmo composto da un ciclo su i che va da 0 a $n-1$, all'interno del quale con un ciclo da i a $n-1$ considera il sottovettore $(A[i], \ldots, A[j])$ e (quando $j > i$) ne calcola il valore sommando $A[j]$ al valore già calcolato di $(A[i], \ldots, A[j-1])$ (si evita rispetto al precedente di ricalcolare più volte le somme parziali dei vari sottovettori);

   (c) l'algoritmo che in una prima fase, con un singolo ciclo, calcola i valori solo dei sottovettori della forma $(A[0], \ldots, A[m])$ al variare di m, chiamando $v[m]$ il valore corrispondente; poi usa questo nuovo vettore per calcolare, con due cicli annidati, i valori di tutti i sottovettori di A, usando il fatto che il valore di $(A[i], \ldots, A[j])$ è uguale a $v[j] - v[i-1]$.

Provate a considerare anche un algoritmo "divide et impera", tenendo conto del fatto che, se si divide A in due vettori di lunghezza metà A_1 e A_2, non è detto che per trovare un sottovettore massimo di A basti confrontare i sottovettori massimi di A_1 e di A_2: quello per A potrebbe trovarsi a cavallo del punto in cui abbiamo diviso il vettore.

Infine, capite perché l'algoritmo riportato di seguito funziona, e perché ha tempo $O(n)$:

Algoritmo 4 massvettlineare(array A)

$massimofinora \leftarrow 0$
$massimonuovo \leftarrow 0$
for $i = 0$ **to** $n - 1$ **do**
  $massimonuovo \leftarrow \max(massimonuovo + A[i], 0)$
  $massimofinora \leftarrow \max(massimofinora, massimonuovo)$
end for

Ovviamente per questo problema un tempo lineare è ottimale, perché già solo la lettura di n dati richiede un tempo $O(n)$.

(Per i più curiosi: è tuttora aperto il problema di trovare un algoritmo ottimale per l'analogo bidimensionale di questo problema: dato un array contenente $n \times n$ numeri, trovare la massima somma su tutti i sottoarray rettangolari.)