Algoritmi e Strutture Dati

Foglio 3
07/03/2025

Esercizio 1. Applicare il Teorema Principale per determinare limiti asintotici stretti per le seguenti ricorrenze:

1. $T(n) = 2T(n/4) + 1$
2. $T(n) = 2T(n/4) + \sqrt{n}$
3. $T(n) = 2T(n/4) + n$
4. $T(n) = 2T(n/4) + n^2$

Esercizio 2. Si consideri l'algoritmo CRUEL descritto di seguito. L'input è un vettore $A[p..q]$ di n interi, con n potenza di 2. Qual è il tempo di esecuzione di CRUEL?

Algorithm CRUEL(A, p, q)

if $q - p + 1 > 1$ **then**
    CRUEL$(A, p, (q + p - 1)/2)$
    CRUEL$(A, (q + p - 1)/2 + 1, q)$
    CRUEL$(A, p, (q + p - 1)/2)$
    MERGE-SORT(A, p, q)
end if

Esercizio 3. Dato un vettore A di n interi distinti, un *minimo locale* di A è un intero x tale che:

- $x = A[i]$, per un certo $1 \leq i \leq n$;
- $A[i] < A[i-1]$ e $A[i] < A[i+1]$.

In parole, un minimo locale di A è un elemento di A che è più piccolo di entrambi gli elementi adiacenti (i due estremi del vettore, $A[1]$ e $A[n]$, hanno un solo elemento adiacente). Per esempio, il vettore $< 8, 9, 1, 2, 4, 3 >$ ha minimi locali 8, 1 e 3. Banalmente, è possibile trovare un minimo locale in $\Theta(n)$. Mostrare che si può fare di meglio.

Esercizio 4. Siano dati un vettore A di n interi ed un intero k. Progettare un algoritmo efficiente che trova, se esiste, una coppia di indici i, j con $i \neq j$ tali che $A[i] + A[j] = k$.

Esercizio 5. Un vettore A di dimensione n ha un *elemento preponderante* se più della metà dei suoi elementi sono uguali. Dato un vettore di dimensione n, si vuole progettare un algoritmo efficiente in grado di stabilire l'esistenza di un elemento preponderante e, nel caso, di trovarlo. Gli elementi del vettore non sono necessariamente numeri e quindi non siamo in grado di fare confronti del tipo $A[i] > A[j]$ (possiamo pensare gli elementi del vettore come dei file .gif, per esempio). Tuttavia, siamo in grado di dire se $A[i] = A[j]$ in $O(1)$ passi. Fornire un algoritmo che risolva tale problema in $O(n \log n)$ passi.

Esercizio 6. Utilizzate le seguenti idee per sviluppare un algoritmo non ricorsivo in tempo lineare per risolvere il problema del massimo sottoarray. Iniziate dall'estremità sinistra dell'array e procedete verso destra, registrando il massimo sottoarray trovato fino a quel punto. Conoscendo un massimo sottoarray di $A[1..j]$, aggiornate la soluzione considerando il massimo sottoarray che termina con l'indice $j + 1$

sulla base della seguente osservazione: un massimo sottoarray di $A[1..j+1]$ è un massimo sottoarray di $A[1..j]$ o un sottoarray della forma $A[i..j+1]$, per un certo $1 \leq i \leq j+1$. Determinate un massimo sottoarray della forma $A[i..j+1]$ in tempo costante, supponendo di conoscere un massimo sottoarray che termina con l'indice j.

Esercizio 7. Qual è il tempo di esecuzione di QUICKSORT quando tutti gli elementi dell'array A hanno lo stesso valore?

Esercizio 8. Considerate la variante di MERGE-SORT in cui una delle due chiamate ricorsive è sostituita da una chiamata a QUICKSORT. Qual è il tempo di esecuzione di questo nuovo algoritmo (nel caso peggiore)?