Elementi di Algoritmi e Strutture Dati
Testo della prova scritta del 4 luglio 2008
docente: Luciano Gualà

Cognome:..... Nome:..... Matr.:..... Corso di Laurea:.....

Esercizio 1 [8 punti] Sia k una costante intera e, per ogni $i \in \{1, \ldots, k\}$, sia $f_i(n) : \mathbb{N} \to \mathbb{R}^+$ una funzione. Inoltre sia $g(n) : \mathbb{N} \to \mathbb{R}^+$ un'altra funzione. Dimostrare o confutare la seguente relazione

$$g(n) - \sum_{i=1}^{k} f_i(n) = \Omega(g(n))$$

quando

(a) $f_{i-1}(n) = \omega(f_i(n))$ per ogni $i = 2, \ldots, k$ e $g(n) = \omega(f_1(n))$.

(b) $\sum_{i=1}^{k} f_i(n) < g(n)$ per ogni $n \geq 0$.

Soluzione esercizio 1

(a) Vera. Infatti, si noti che per la proprietà transitiva della relazione $\omega(\cdot)$, per ogni $i \in \{1, \ldots, k\}$, abbiamo che $g(n) = \omega(f_i(n))$, da cui segue che $\lim_{n\to\infty} f_i(n)/g(n) = 0$. Quindi:

$$\lim_{n\to\infty} \frac{g(n) - \sum_{i=1}^{k} f_i(n)}{g(n)} = \lim_{n\to\infty} 1 - \frac{\sum_{i=1}^{k} f_i(n)}{g(n)} = \lim_{n\to\infty} 1 - \sum_{i=1}^{k} \frac{f_i(n)}{g(n)} = 1.$$

Che implica $g(n) - \sum_{i=1}^{k} f_i(n) = \Theta(g(n))$ e quindi la tesi.

(b) Falsa. Sia $k \geq 1$ una costante intera. Il seguente insieme di funzioni costituisce un controesempio: $g(n) = n+1$, $f_1(n) = n$ e $f_i(n) = 0$ per ogni $i = 2, \ldots, k$. Risulta $\sum_{i=1}^{k} f_i(n) = n < n+1 = g(n)$ per ogni n, ma $g(n) - \sum_{i=1}^{k} f_i(n) = 1 \neq \Omega(n+1)$.

Esercizio 2 [8 punti] Realizzare un algoritmo che, preso in input un vettore A di n valori distinti, e un intero k ($1 \leq k \leq n$), restituisce i k elementi più grandi di A. L'algoritmo deve avere complessità temporale $o(n \log n)$ quando $k = o(\log n)$. *Attenzione*: l'esercizio sarà valutato solo se corredato da adeguata descrizione del funzionamento dell'algoritmo, in base ai seguenti parametri: correttezza, efficienza e analisi di complessità.

Soluzione esercizio 2 L'approccio più intuitivo al problema è sicuramente quello di ordinare il vettore, per esempio in ordine crescente, e poi restituire gli ultimi k elementi. Questa soluzione ha chiaramente complessità $O(n \log n + k)$ e quindi non rispetta le specifiche richieste dal testo. Di seguito vengono proposte tre diverse soluzioni al problema. Per ognuna delle soluzioni verrà fornita l'idea principale e l'analisi della complessità temporale, mentre lo pseudocodice è lasciato come esercizio allo studente.

Soluzione 1. L'idea è quella di effettuare k diverse ricerche del massimo su A. L'unica accortezza da avere è quella di "eliminare" l'elemento dal vettore una volta che è stato trovato, per evitare di trovarlo nuovamente nelle successive ricerche. Un modo semplice per far questo (ma ce ne sono altri che non cambiano la complessità temporale dell'algoritmo) è quello di impostare a $-\infty$ la cella del vettore che contiene il massimo appena individuato. La complessità di questa soluzione è $O(kn)$, in quanto vengono effettuate k ricerche del massimo ognuna delle quali ha costo $O(n)$.

Soluzione 2. Scorriamo il vettore A con un indice i che va da 1 a n. Ad ogni passo i teniamo traccia dei k valori più grandi fra quelli in $A[1, i]$ in una qualche struttura dati (all'inizio la struttura contiene k elementi di valore $-\infty$). Quando i è uguale a n restituiamo i k valori contenuti nella struttura. A questo punto è necessario solo stabilire quale struttura dati utilizzare ed esplicitare il passo generico dell'algoritmo. Una struttura dati utile a questo scopo può essere un albero AVL. All'inizio di ogni passo i l'albero AVL contiene i k elementi più grandi fra gli $i-1$ elementi precedentemente visti (i primi $i-1$ elementi di A). Poi l'algoritmo considera l'elemento $A[i]$: viene effettuata una ricerca di minimo sull'albero AVL. Se tale minimo è maggiore di $A[i]$ non si fa niente, altrimenti si elimina il minimo dall'albero e si inserisce l'elemento $A[i]$. All'inizio, come passo di inizializzazione, vengono inseriti nell'albero vuoto k elementi con chiave $-\infty$. Per quanto riguarda la complessità di questa soluzione possiamo notare che il passo di inizializzazione ha costo $O(k \log k)$ (k inserimenti di costo $O(\log k)$). Poi ad ogni passo vengono effettuate al più una operazione di ricerca di minimo nell'albero (che ha k elementi), una di cancellazione e una di inserimento. Ognuna di queste operazioni ha costo $O(\log k)$, il che implica che la complessità temporale dell'intero algoritmo è $O(k \log k + n \log k) = O(n \log k)$.

Soluzione 3. La terza soluzione che proponiamo è semplice ed efficiente. A partire dal vettore A, si costruisce un heap attraverso la procedura `Heapify` e poi si effettuano k estrazioni di massimo dall'heap. La procedura `Heapify` ha complessità $O(n)$ e ogni estrazione costa $O(\log n)$, quindi la complessità dell'intero algoritmo risulta essere $O(n + k \log n)$.

Esercizio 3 [8 punti]

(a) Si calcoli la distanza fra le stringhe X e Y utilizzando l'algoritmo di programmazione dinamica `distanzaStringhe`, con $X =$ `sodoma` e $Y =$ `gomorra`. Si mostri, inoltre, la sequenza minima di operazioni che trasforma X in Y.

(b) Si risolva la seguente equazione di ricorrenza: $T(n) = T(n/5) + T(4n/5) + n$, $T(1) = 1$.

Soluzione esercizio 3

(a) Di seguito viene riportata la matrice calcolata dall'algoritmo per $X =$ `sodoma` e $Y =$ `gomorra`. La distanza fra X e Y si trova nella cella della matrice in basso a destra ed è quindi 4. In grassetto viene indicata una delle possibili sequenze di operazioni che permette di trasformare X in Y, ovvero: sostituisci S con G, mantieni O, sostituisci D con M, mantieni O inserisci R, sostituisci M con R, mantieni A.

		G	O	M	O	R	R	A
	0	1	2	3	4	5	6	7
S	1	**1**	2	3	4	5	6	7
O	2	2	**1**	2	3	4	5	6
D	3	3	2	**2**	3	4	5	6
O	4	4	3	3	**2**	**3**	4	5
M	5	5	4	3	3	3	**4**	5
A	6	6	5	4	4	4	4	**4**

(b) Risolviamo la relazione di ricorrenza utilizzando il metodo dell'albero della ricorsione. Disegnando l'albero della ricorsione relativo alla relazione in esame, è facile vedere che la somma della dimensione dei nodi al livello i è al più n. Poiché per risolvere un nodo di dimensione x spendiamo esattamente x, ogni livello ci costa al più n. Dato che i livelli sono $O(\log_{5/4} n) = O(\log n)$, possiamo affermare che $T(n) = O(n \log n)$.

Esercizio 4 [8 punti] Si descriva in modo sintetico e preciso cosa è un Heap e come può essere mantenuto attraverso un vettore. Poi si mostri che un Heap di n elementi ha altezza $O(\log n)$.

Soluzione esercizio 4 Si consulti il libro di testo.