Elementi di Algoritmi e Strutture Dati
Testo della prova scritta del 9 luglio 2007
docente: Luciano Gualà

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

Esercizio 1 [8 punti] Siano $f(n), g(n) : \mathbb{N} \to \mathbb{R}^+$ due funzioni. Dimostrare o confutare la seguente relazione

$$n^4 \cdot (f(n))^2 = O(g(n) \cdot f(n)),$$

quando:

(a) $g(n) = \Omega(n^4 \cdot f(n))$.

(b) $g(n) = \omega(n^{134} \cdot 2^{2^n})$.

Soluzione esercizio 1 Dimostriamo che la (a) è vera. Per definizione di $O(\cdot)$, dobbiamo cercare due costanti positive c e n_0, per cui valga la relazione $n^4 f(n) f(n) \leq c\, g(n) f(n), \forall n \geq n_0$. Per ipotesi sappiamo che (applicando la definizione di $\Omega(\cdot)$) esistono due costranti c' e n_0' per cui vale $g(n) \geq c' n^4 f(n), \forall n \geq n_0'$, da cui segue che $n^4 f(n) \leq \frac{1}{c'} g(n), \forall n \geq n_0'$. Quindi, è sufficiente scegliere $c = \frac{1}{c'}$ e $n_0 = n_0'$.

La (b) è falsa. Si considerino le seguenti due funzioni: $g(n) = n^{135} 2^{2^n}$ e $f(n) = n^{132} 2^{2^n}$. Per tali funzioni vale $g(n) = \omega(n^{134} \cdot 2^{2^n})$. Inoltre, è anche vero che $n^4 \cdot (f(n))^2 = \omega(g(n) \cdot f(n))$ (e quindi $n^4 \cdot (f(n))^2 \neq O(g(n) \cdot f(n))$), infatti:

$$\lim_{n\to\infty} \frac{n^4 f(n) f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^4 \cdot n^{132} 2^{2^n}}{n^{135} \cdot 2^{2^n}} = \lim_{n\to\infty} n = \infty.$$

Esercizio 2 [8 punti] Dato un albero AVL con n chiavi e un intero $k \leq n$, realizzare un algoritmo che restituisca l'elemento che occupa la k-esima posizione nella sequenza ordinata delle chiavi. *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 Di seguito discutiamo brevemente quattro differenti soluzioni al problema. Dei quattro algoritmi proposti, forniremo solo una descrizione di alto livello dell'idea di base. I relativi pseudocodici sono lasciati come esercizio allo studente.

Una soluzione semplice è quella di visitare l'abero AVL con un qualsiasi algoritmo di visita e, ogni volta che un nodo viene visitato, si inserisce il corrispondente elemento all'interno di un vettore (di dimensione n). Quando la visita è terminata, si ordina il vettore con un qualsiasi algoritmo ottimo per l'ordinamente (per esempio

il Merge Sort), e si restituisce l'elemento che occupa la k-esima posizione dell'array. Questa soluzione ha complessità $O(n \log n)$, poiché visitare l'albero costa $O(n)$, mentre oridinare l'array prende tempo $O(n \log n)$. Si noti, inoltre, che tale soluzione non usa l'ipotesi che l'albero di partenza sia un albero AVL.

Una soluzione più efficiente consiste nel modificare il primo algoritmo fornito usando, come algoritmo di visita, quello in profondità simmetrica. Poiché l'albero è un AVL, e quindi un albero binario di ricerca, la visita restituirà i nodi in ordine crescente di chiave, il che implica che il vettore risultante sarà già ordinato (se si usa l'accortezza di inserire i nodi nell'array a partire dalla prima posizione, e poi nella seconda e via via fino all'n-esima). Tale soluzione ha complessità $O(n)$. Si noti che questa soluzione usa il fatto che l'albero è una albero binario di ricerca, ma non usa l'ipotesi che l'abero è anche AVL (e quindi bilanciato).

La terza soluzione che proponiamo fa uso anche dell'ipotesi che l'albero di partenza è bilanciato. L'algoritmo è il seguente. Per $k-1$ volte, si trova il minimo e lo si cancella. Quindi, la k-esima ricerca del minimo restituirà l'elemento voluto. Tale soluzione ha complessità $O(k \log n)$, in quanto ogni ricerca e ogni cancellazione ha costo $O(\log n)$, e ci sono in tutto $2k-1$ operazioni (k ricerche e $k-1$ cancellazioni). Si osservi, infine, che questa soluzione è non peggiore della precedente per valori "non troppo grandi" di k, più formalmente quando $k = O(\frac{n}{\log n})$, ed è strettamente migliore quando $k = o(\frac{n}{\log n})$.

L'ultima soluzione che proponiamo domina le altre tre. L'algoritmo è semplice: si visita l'albero AVL in ordine simmetrico, si tiene traccia del numero di nodi correntemente visitati, e quando si arriva al k-esimo nodo, si stoppa la visita e si restituisce il corrispondente elemento. Si noti che il numero di nodi visitati dall'algoritmo è esattamente k (e non n come nella prima e nella seconda soluzione); si osservi, inoltre, che, poiché la visita in ordine simmetrico inizia dal nodo di chiave minima che è una foglia dell'albero, l'algoritmo deve in ogni caso attraversare tutto il cammino che dalla radice porta a tale nodo. Poiché questo cammino ha lunghezza $O(\log n)$, la complessità dell'algoritmo è $O(\max\{k, \log n\})$.

Esercizio 3 [8 punti] Si mostri l'albero binario corrispondente al vettore posizionale T_1, e si dia l'ordine di visita dei nodi mediante una visita in profondità simmetrica. Poi si consideri il vettore dei padri T_2. Si mostri l'albero corrispondente, si dica qual è il grado e la profondità del nodo A e si dia l'ordine di visita dei nodi mediante un visita in ampiezza. Si assuma che entrambi i vettori abbiano indici compresi fra 1 e 10.

$T_1 = [G, V, T, O, R, A, A, T, R, E]$.

$T_2 = [(G, 8), (H, 6), (C, 8), (B, \texttt{null}), (I, 2), (E, 8), (L, 2), (A, 4), (D, 8), (F, 8)]$

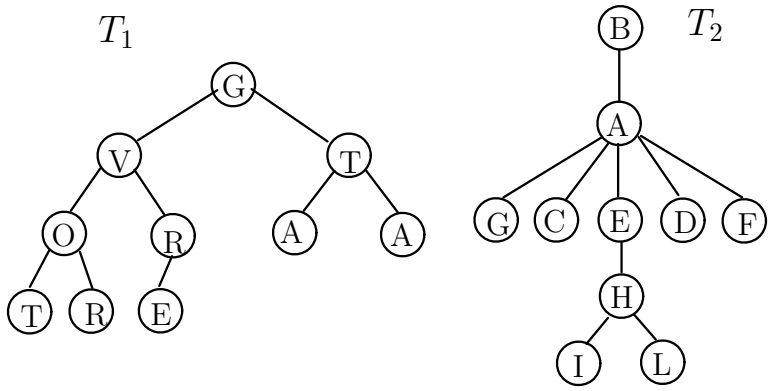

Figura 1: Gli alberi T_1 e T_2.

Soluzione esercizio 3 I due alberi sono mostrati in Figura 1. L'ordine di visita dei nodi di T_1 è: T O R V E R G A T A. Per quanto riguarda l'albero T_2, il grado di A è 5, la sua profondità 1, e l'ordine di visita dei suoi nodi è: B A G C E D F H I L.

Esercizio 4 [8 punti] Si definisca formalmente un albero AVL e si dimostri che un albero AVL con n nodi ha altezza $O(\log n)$.

Soluzione esercizio 4 Si consulti il libro di testo.