Introduzione agli Algoritmi / Algoritmi I (A.A. 2009-2010)
Docenti: Proff. Finocchi, Carlucci
Esame del 24 Giugno 2010

Gli studenti esonerati devono svolgere solo gli esercizi 3 e 4.

Esercizio 1 (8 punti). Supponiamo di modificare l'algoritmo `insertionSort` come segue: al passo generico, la ricerca della posizione in cui inserire l'elemento $A[k+1]$ tra gli elementi $A[1], ..., A[k]$ (righe 3 e 4 in Figura 4.4 del libro di testo) viene sostituita da una ricerca binaria.

- Dare una stima del *numero di confronti tra elementi* nel caso peggiore. Rispetto al caso in cui la ricerca della posizione avviene con una scansione lineare, la ricerca binaria permette di ottenere un numero di confronti asintoticamente inferiore?
- Dare una stima del *numero di spostamenti di elementi* nel caso peggiore (istruzione "$A[t+1] \leftarrow A[t]$" nella riga 6). Rispetto al caso in cui la ricerca della posizione avviene con una scansione lineare, la ricerca binaria permette di ottenere un numero di spostamenti asintoticamente inferiore?
- Stimare il *tempo di esecuzione* della nuova implementazione dell'`insertionSort` nel caso peggiore. Rispetto al caso in cui la ricerca della posizione avviene con una scansione lineare, il tempo di esecuzione è asintoticamente migliore?

Esercizio 2 (8 punti). La *torre di Hanoi* è un rompicapo matematico composto da tre paletti A, B e C e da n dischi di grandezza decrescente. Il gioco inizia con tutti i dischi incolonnati sul paletto A in ordine decrescente, in modo da formare un cono. Lo scopo del gioco è di portare tutti dischi sul paletto C, potendo spostare solo un disco alla volta e potendo mettere un disco solo su un disco più grande, mai su uno più piccolo.

1. La classica soluzione ricorsiva del gioco della torre di Hanoi lavora come segue:
   (a) Sposta i primi $n-1$ dischi da A a B. (Questo lascia il disco più grande da solo sul paletto A.)
   (b) Sposta il disco più grande da A a C.
   (c) Sposta gli altri $n-1$ dischi da B a C.

   Analizzare il numero di spostamenti di dischi (e quindi il tempo di esecuzione) del suddetto algoritmo.
2. Il professor De Rompicapis, che ama molto i giochi matematici, sostiene di aver trovato un algoritmo *polinomiale* per risolvere il gioco della torre di Hanoi usando *quattro* paletti A, B, C e D. L'algoritmo opera come segue:
   (a) Sposta i primi $n-2$ dischi da A a B. (Questo lascia i due dischi più grandi, $n-1$ e n, da soli sul paletto A.)
   (b) Sposta il disco $n-1$ da A a C.
   (c) Sposta il disco n da A a D.
   (d) Sposta il disco $n-1$ da C a D.
   (e) Sposta gli altri $n-2$ dischi da B a D.

   Il professore merita il premio Nobel o, piuttosto, il premio Ignobel? Motivare la risposta.

Esercizio 3 (7 punti). Sia H un min-heap (ovvero un heap in cui il minimo è nella radice) contenente n chiavi, di cui $\lfloor n/2 \rfloor$ hanno valore 1 e $\lceil n/2 \rceil$ hanno valore 2. Si assuma che $n = 2^k - 1$, per $k \geq 2$. Rispondere alle seguenti domande motivando dettagliatamente le risposte.

1. E' possibile disporre le n chiavi nell'heap in modo che tutte le chiavi di valore 1 occupino livelli minori delle chiavi di valore 2?
2. E' possibile disporre le n chiavi nell'heap in modo che esistano un nodo u con chiave 1 ed un nodo v con chiave 2 tali che $livello(u) > livello(v)$?
3. Esiste una disposizione delle chiavi per cui l'operazione `extractMin` di estrazione del minimo richieda tempo $O(1)$?
4. E' vero che l'operazione `extractMin` di estrazione del minimo richiede sempre tempo $O(1)$, indipendentemente dalla disposizione delle chiavi?
5. E' corretto dire che inserire in H una nuova chiave di valore 0 costa $\Omega(\log n)$?

I livelli si intendono numerati da 0, a partire dalla radice.

Esercizio 4 (9 punti). Sia T un albero AVL contenente n chiavi intere, positive, tutte distinte tra loro.

1. Assumere che ogni nodo u di T contenga, oltre alla chiave, ai puntatori e al fattore di bilanciamento, anche un campo `sum` in cui è mantenuta la somma delle chiavi nel sottoalbero radicato in u.

   Esempio: se T è l'albero di ricerca binario completo contenente le chiavi 1, 2, 3, 4, 5, 6, e 7, allora:

   - $\texttt{sum}(4) = 28$ poiché la chiave 4 è nella radice dell'albero e $1 + 2 + 3 + 4 + 5 + 6 + 7 = 28$;
   - $\texttt{sum}(6) = 18$ poiché la chiave 6 è nel figlio destro della radice e $5 + 6 + 7 = 18$;
   - $\texttt{sum}(3) = 3$ poiché la chiave 3 è in una foglia.

   Mostrare che il campo `sum` può essere mantenuto efficientemente a fronte di inserimenti e cancellazioni di nodi (incluse le rotazioni).

2. Implementare una nuova operazione `newOp` che, dati un albero AVL T e un valore intero s, restituisca la più piccola chiave k di T tale che la somma delle chiavi minori di k è strettamente maggiore di s. Se una tale chiave non esiste, `newOp` deve restituire $+\infty$.

   Esempio: sia T l'albero dell'esempio precedente. Allora:

   - $\texttt{newOp}(T, 9) = 5$ poiché $1 + 2 + 3 + 4 = 10 > 9$ ma $1 + 2 + 3 = 6 \leq 9$;
   - $\texttt{newOp}(T, 15) = 7$ poiché $1 + 2 + 3 + 4 + 5 + 6 = 21 > 15$ ma $1 + 2 + 3 + 4 + 5 = 15 \leq 15$;
   - $\texttt{newOp}(T, 100) = +\infty$ poiché $1 + 2 + 3 + 4 + 5 + 6 + 7 = 28 \leq 100$.

   Proporre un algoritmo per implementare `newOp`, analizzarne la correttezza e il tempo di esecuzione, che dovrebbe essere $O(\log n)$. Vi consiglio di usare opportunamente il campo `sum` introdotto al punto 1.

In bocca al lupo!