Elementi di Algoritmi e Strutture Dati

Testo della prova scritta del 6 giugno 2007

docente: Luciano Gualà

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

Esercizio 1 [8 punti] Sia k una costante positiva, e sia $f(n) : \mathbb{N} \to \mathbb{R}^+$ una funzione tale che $f(n) = \omega(1)$. Dimostrare o confutare le seguenti affermazioni.

(a) $2^{n+2^k} = \omega(2^n)$

(b) $2^n = o(2^{n+f(n)})$

Soluzione esercizio 1

(a) Falsa, infatti, applicando la definizione di $\omega(\cdot)$ si ha:

$$\lim_{n\to\infty} \frac{2^{n+2^k}}{2^n} = \lim_{n\to\infty} \frac{2^n \cdot 2^{2^k}}{2^n} = 2^{2^k} \neq \infty$$

Si noti che 2^{2^k} è un valore costante che non dipende da n.

(b) Vera, infatti, applicando la definizione di $o(\cdot)$ si ha:

$$\lim_{n\to\infty} \frac{2^n}{2^{n+f(n)}} = \lim_{n\to\infty} \frac{2^n}{2^n \cdot 2^{f(n)}} = \lim_{n\to\infty} \frac{1}{2^{f(n)}} = 0$$

L'ultima uguaglianza segue perché $f(n)$ va ad infinito per n che tende ad infinito (poiché $f(n) = \omega(1)$) e perché 2^x è una funzione che va ad infinito quando il suo argomento x tende ad infinito.

Esercizio 2 [8 punti] Dato un heap binario rappresentato tramite vettore posizionale, realizzare un algoritmo che ristrutturi l'heap in modo tale che ogni figlio sinistro sia sempre maggiore o uguale del corrispondente fratello, e che restituisca inoltre i tre elementi più grandi contenuti nell'heap. *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 Sia H il vettore posizionale che rappresenta l'heap, e sia n il numero di elementi contenuti nell'heap. Una soluzione semplice al problema è quella di ordinare l'array H in modo non crescente. L'array risultante rappresenta un heap con gli stessi elementi ed è facile verificare che rispetta anche la proprietà aggiuntiva richiesta. Chiaramente, i tre elementi più grandi sono nelle prime tre

posizioni dell'array. Tale soluzione ha complessità $O(n \log n)$, se si usa un qualsiasi algoritmo ottimo per ordinare H (per esempio il Merge Sort).

Di seguito è riportata una soluzione più efficiente, che risolve il problema in tempo $O(n)$. L'idea è semplice: si considera la radice, si verifica se la proprietà richiesta è valida per i figli della radice. Se non è valida, si scambiano i figli e si chiama `fixHeap` sul figlio destro (l'albero sinistro della radice per costruzione è già un heap). Si è ottenuto così un heap che rispetta la condizione per i figli della radice. A questo punto si chiama ricorsivamente la stessa procedura sui due sottoalberi. Si noti, inoltre, che, dopo la ristrutturazione, i primi due massimi sono rispettivamente nella radice e nel figlio sinistro della radice, mentre il terzo massimo va cercato fra il figlio destro della radice e il figlio sinistro del nodo contenente il secondo massimo. Un possibile pseudocodice è mostrato di seguito. Per quanto riguarda l'analisi, è chiaro che la complessità dell'algoritmo `Ristruttura` è dominata da quella della chiamata `RistruttutaraRic`$(H, 1)$. Quest'ultimo algoritmo è del tutto simile all'algoritmo `Heapify` (una chiamata a `fixHeap` e due chiamate ricorsive sui due sottoalberi) ed è facile vedere che le equazioni di ricorrenza dei due algoritmi coincidono. Da cui segue che la complessità di `Ristruttura` è lineare nel numero di elementi presenti nell'heap (si può fare la stessa analisi fatta per `Heapify`).

```
Ristruttura(H)
Sia Max un vettore di 3 elementi;
RistrutturaRic(H, 1);
Max[1] = H[1];
Max[2] = H[2];
if H[3] ≥ H[4] then
    Max[3] = H[3];
else
    Max[3] = H[4];
end
return Max;
```

dove

```
RistrutturaRic(H, i)
if 2i + 1 ≤ HeapSize[H] then
    s = 2i;
    d = 2i + 1;
    if H[s]<H[d] then
        scambia H[s] e H[d];
        fixHeap(H, d);
    end
    RistrutturaRic(H, s);
    RistrutturaRic(H, d);
end
```

Esercizio 3 [8 punti] A partire da un albero AVL vuoto, si mostrino le modifiche apportate all'albero in seguito agli inserimenti elencati in (a) e poi alle cancellazioni elencate in (b). L'albero risultante è un albero di Fibonacci? Si motivi la risposta.

(a) *Inserimenti*: 10, 16, 20, 13, 14, 12, 11;

(b) *Cancellazioni*: 20, 11, 16.

Soluzione esercizio 3 In figura 1 è mostrato l'albero AVL a seguito degli inserimenti (sono mostrate anche le situazioni intermedie prima delle eventuali rotazioni) mentre in figura 2 è mostrato l'albero a seguito delle cancellazioni.

L'albero risultante è un albero di Fibonacci perchè è un albero AVL (di altezza 2) con il minimo numero di nodi ($F_{2+3} - 1 = F_5 - 1 = 4$ nodi), ovvero la cancellazione di un qualsiasi altro nodo sbilancia l'albero o ne cambia l'altezza.

Esercizio 4 [8 punti] Illustrare in modo sintetico e conciso la delimitazione inferiore al numero di confronti per il problema dell'ordinamento.

Soluzione esercizio 4 Consultare il libro di testo.

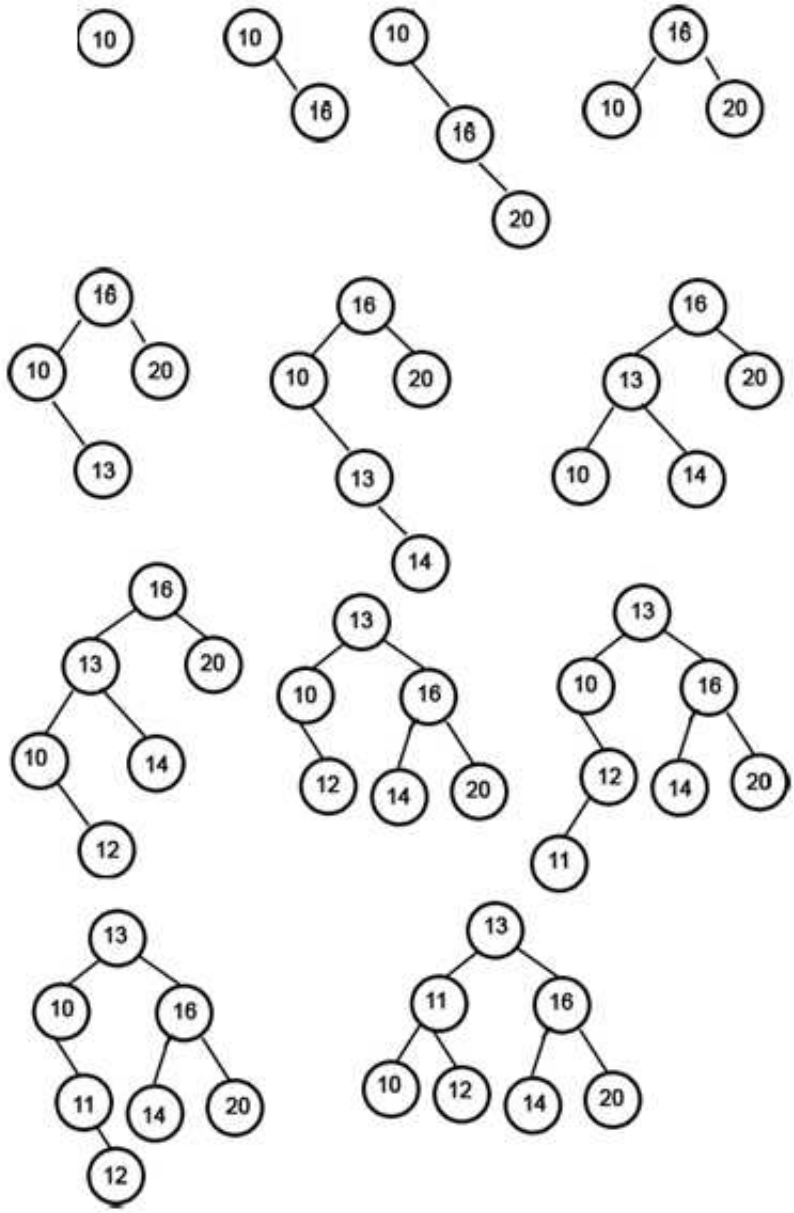

Figura 1: *Inserimenti*: 10, 16, 20, 13, 14, 12, 11

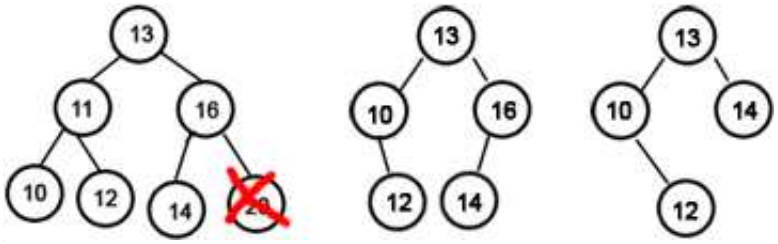

Figura 2: *Cancellazioni*: 20, 11, 16