

Algoritmi e Strutture Dati

Sessione invernale A.A. 2003/2004

Appello 26.01.04

Prima parte

1. Per un certo problema sono stati trovati due possibili algoritmi risolutivi. I loro tempi di esecuzione soddisfano alle due relazioni di ricorrenza riportate nei seguenti punti (a) e (b). Si dica, giustificando la risposta, quale dei due algoritmi è da preferire nel caso si debbano risolvere problemi di grandi dimensioni.

$$(a) \quad T_1(n) = 3T_1\left(\frac{n}{2}\right) + 3n^2 \lg^2 n$$

$$(b) \quad T_2(n) = 4T_2\left(\frac{n}{2}\right) + 2n^2 + n + 2 \lg^2 n$$

Utilizziamo il Master Theorem per valutare la complessità asintotica dei due algoritmi.

(a) Si ha: $a = 3, b = 2$, quindi $1 < \log_2 3 < 2$. Poiché $3n^2 \lg^2 n = \Omega(n^2) = \Omega(n^{\log_2 3 + \epsilon})$ se vale la condizione di regolarità ($\exists c < 1$ tale che $af(\frac{n}{b}) < cf(n)$) siamo nel caso 3 del Master Method. La condizione vale in quanto scelto $c = 3/4$ è facile vedere che $3(3\frac{n^2}{4} \lg^2(\frac{n}{2})) < \frac{3}{4}(3n^2 \lg^2 n)$. Allora $T_1(n) = \Theta(3n^2 \lg^2 n)$.

(b) Si ha: $a = 4, b = 2$, quindi $\log_2 4 = 2$. Poiché $2n^2 + n + 2 \lg^2 n = \Theta(n^2)$ siamo nel caso 2 del Master Method. Allora $T_2(n) = \Theta(n^2 \lg n)$.

Dobbiamo quindi confrontare $3n^2 \lg^2 n$ con $n^2 \lg n$. Poiché $n^2 \lg n = O(3n^2 \lg^2 n)$ ma non vale il viceversa, l'algoritmo da scegliere è il secondo.

2. Data la seguente procedura FUN se ne determini la complessità asintotica al crescere di $n \in N$

FUN(A, n)

```
1  if  $n < 1$  return 1
2   $s \leftarrow 0$ 
3  for  $j \leftarrow 1$  to  $n$ 
4      do  $s \leftarrow s + A[j]$ 
5  return  $s + 2 \text{ FUN}(A, n/2)$ 
```

$\text{FUN}(A, n)$

```

1  if  $n < 1$  return 1
2   $s \leftarrow 0$ 
3  for  $j \leftarrow 1$  to  $n$ 
4      do  $s \leftarrow s + A[j]$ 
5  return  $s + 2 \text{FUN}(A, n/2)$ 
```

$$\begin{array}{l} 1 \\ n + 1 \\ \sum_{j=1}^n 1 \\ T(n/2) \end{array}$$

Poiché $T_{\text{FUN}}(n)$ soddisfa la ricorrenza $T_{\text{FUN}}(n) = T_{\text{FUN}}(n/2) + kn$ la complessità asintotica della procedura FUN si trova risolvendo tale ricorrenza. Questo si ottiene facilmente utilizzando il Master Method. Si ha $a = 1, b = 2, \log_b a = 0, kn = \Omega(n^{0+\epsilon})$, e kn soddisfa la proprietà di regolarità. Quindi siamo nel caso 3 ed abbiamo $T_{\text{FUN}}(n) = \Theta(n)$.

3. Si consideri la struttura dati albero (posizionale e generale) con gli attributi $\text{key}[x]$, $\text{child}[x]$, $\text{sibling}[x]$ associati ad ogni nodo x . Si descriva un algoritmo che dato un intero k ed un albero T , calcola il numero dei nodi di T che hanno esattamente k figli.

Si può risolvere con una visita in ampiezza *breadth-first-search* contando le iterazioni del ciclo *while* che visita ciascun gruppo di fratelli. Usiamo una variabile globale tot posta uguale a 0 prima della chiamata esterna $\text{CHILDREN-COUNT}(\text{root}[T], k)$.

$\text{CHILDREN-COUNT}(x, k)$

```

1  if  $x = \text{NIL}$  return
2   $\text{ENQUEUE}(x, Q)$ 
3  while not  $\text{QUEUE-EMPTY}[Q]$ 
4      do  $y \leftarrow \text{HEAD}[Q]$ 
5           $\text{DEQUEUE}[Q]$ 
6           $s \leftarrow 1$ 
7          while  $y \neq \text{NIL}$ 
8              do  $s \leftarrow s + 1$ 
9                  if  $\text{CHILD}[y] \neq \text{NIL}$ 
10                      then  $\text{ENQUEUE}[\text{CHILD}[y], Q]$ 
11                       $y \leftarrow \text{SIBLING}[y]$ 
12          if  $s = k$ 
13              then  $\text{tot} \leftarrow \text{tot} + 1$ 
```

Seconda parte

4. Si consideri l'array $A[1..7]$ contenente gli elementi 3,25,10,50,2,5,7.

- (a) Dire se A soddisfa la proprietà di max-heap o di quasi-max-heap o nessuna delle due. Giustificare la risposta.

Non è né un max-heap né un quasi-max-heap. Infatti $A[1] < A[\text{LEFT}[1]]$ e quindi non è un max-heap, inoltre $A[\text{LEFT}[1]] < A[\text{LEFT}[\text{LEFT}[1]]]$ e quindi non è neppure un quasi-max-heap.

- (b) Nel caso in cui A non sia uno heap descrivere il risultato dell'applicazione della procedura $\text{BUILD-MAX-HEAP}(A)$.

L'array che si ottiene dopo l'applicazione di $\text{BUILD-MAX-HEAP}(A)$ è $A = [50, 25, 10, 3, 2, 5, 7]$.

- (c) Descrivere infine quale è il risultato dell'applicazione della procedura $\text{HEAP-EXTRACT-MAX}(A)$ al max-heap risultante dai punti precedenti.

Dopo l'estrazione del massimo l'array diviene $A = [25, 7, 10, 3, 2, 5]$.

5. Scrivere un algoritmo che dato un albero binario di ricerca T ed una chiave k restituisce il numero di chiavi di T il cui valore è minore di k . Valutare la complessità dell'algoritmo proposto.

Proseguiamo come per una ricerca della chiave k contando tutti i nodi che vengono lasciati alla sinistra del cammino percorso durante la ricerca. Si noti che la stessa procedura può essere utilizzata anche per *contare* i nodi che vengono lasciati alla sinistra dato che alla sinistra di una chiave minore di k ci sono solo nodi minori di k . La chiamata esterna sarà $\text{LESS}(\text{ROOT}[T], k)$.

$\text{LESS}(x, k)$

```
1  if  $x = \text{NIL}$ 
2      then return 0
3  if  $\text{KEY}[x] \geq k$ 
4      then return  $\text{LESS}(\text{LEFT}[x])$ 
5  else return  $1 + \text{LESS}(\text{LEFT}[x]) + \text{LESS}(\text{RIGHT}[x])$ .
```

6. Descrivere le proprietà della procedura di partizione di una array che viene utilizzata dall'algoritmo $\text{QUICKSORT}(A, p, q)$.

L'algoritmo di quicksort si basa su di una procedura di partizione che deve soddisfare la seguente proprietà.

PARTIZIONA[A, p, q] riorganizza la porzione $p \cdots q$ dell'array A e restituisce un indice r compreso tra p e q in modo che nell'array riorganizzata tutti gli elementi in $A[p \cdots r]$ siano minori o uguali a tutti gli elementi in $A[r+1 \cdots q]$. Formalmente

precondizione: $A[1..n]$ è una array di lunghezza n , $1 \leq p \leq q \leq n$.

$r \leftarrow \text{PARTIZIONA}[A, p, q]$

postcondizione: $p \leq r \leq q$ e $A[p \cdots r] \leq A[r+1 \cdots q]$.

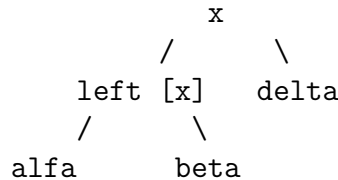
Si osservi che l'uso del pivot nelle realizzazioni studiate *serve* a garantire questa proprietà.

7. Sia x un nodo in un albero Rosso/Nero T tale che $\text{colore}[x] = \text{nero}$, $\text{colore}[\text{left}[x]] = \text{rosso}$, $\text{colore}[\text{right}[x]] = \text{nero}$. Dire, giustificando formalmente la risposta, se l'applicazione di una rotazione a destra al nodo x distrugge la proprietà di albero Rosso/Nero o no.

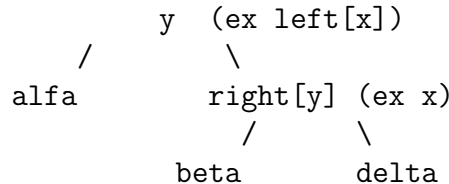
La rotazione non mantiene la proprietà (N.B. in nessun caso).

Chiamiamo altezza nera estesa di T l'altezza nera di T incrementata di 1 se la radice di T è nera e la denotiamo con $bh^*(T)$.

Siano α e β i figli di $\text{left}[x]$ e chiamiamo δ il nodo $\text{right}[x]$. Poiché T è un albero Rosso/Nero, e $\text{colore}[\text{left}[x]] = \text{rosso}$, deve essere $bh^*(\alpha) = bh^*(\beta) = bh^*(\delta)$.



Chiamiamo y la nuova radice del sottoalbero dopo la rotazione.



Il figlio sinistro di y è α e quindi la sua altezza nera estesa è proprio $bh^*(\alpha)$. I figli di $\text{right}[y]$ sono β e δ che hanno la stessa altezza nera estesa, cioè $bh^*(\beta)$. Pertanto $\text{right}[y]$, che è nero ha altezza nera estesa $bh^*(\beta) + 1$.

Poiché $bh^*(\beta) + 1 = bh^*(\alpha) + 1 \neq bh^*(\alpha)$ l'albero dopo la rotazione non gode più della proprietà sulle altezze nere.