Algoritmi e Strutture Dati

Sessione estiva A.A. 2003/2004

Appello 10.06.04

1. *Data la ricorrenza*

$$T(n) = 2 \cdot T(\frac{2n}{3}) + n^2$$

utilizzando il metodo di sostituzione dimostrare che $T(n) = \Omega(n^2)$.

Soluzione: Dobbiamo dimostrare che esistono due costanti positive c e n_0 tale che $T(n) \geq cn^2$, per ogni $n \geq n_0$. Poiché $T(n) = 2 \cdot T(\frac{2n}{3}) + n^2$, assumendo $T(\frac{2n}{3}) \geq c \cdot (\frac{2n}{3})^2$ si ottiene: $T(n) = 2 \cdot T(\frac{2n}{3}) + n^2 \geq 2c \cdot (\frac{2n}{3})^2 + n^2$ $= (\frac{8}{9} \cdot c + 1)n^2$
La diseguaglianza $(\frac{8}{9} \cdot c + 1)n^2 \geq cn^2$ è soddisfatta per ogni $c \leq 9$ e per ogni $n_0 > 0$. Quindi esistono c ed n_0 tali che $T(n) \geq cn^2$ per ogni $n \geq n_0$.

2. *Date le seguenti procedure* A *e* B, *si determini la complessità asintotica della procedura* A(n) *su input* $n \in N$

A(n)
$s \leftarrow 0$
for $i \leftarrow 1$ **to** n
    do $s \leftarrow s + \mathrm{B}(n)$
return s

B(m)
if $m = 1$
    then return 0
    else return $B(m/2) + m$

Soluzione: La complessità di B può essere espressa tramite la ricorrenza $T_B(n) = T_B(n/2) + \Theta(1)$ che si risolve facilmente con il master method ottenendo $T_B(n) = \Theta(logn)$. Per la complessità di A abbiamo

A(n)

$s \leftarrow 0$	1
for $i \leftarrow 1$ **to** n	$n + 1$
do $s \leftarrow s + \mathrm{B}(n)$	$\sum_{i=1}^{n} T_B(n) = nT_B(n)$
return s	1

e quindi:

$$T_A(n) = \sum_{i=1}^{n} T_B(n) = \sum_{i=1}^{n} \Theta(logn) = \Theta(nlogn)$$

3. *Si consideri la struttura dati albero (posizionale e generale) e si assuma che ad ogni nodo x, oltre agli usuali attributi key$[x]$, child$[x]$, sibling$[x]$, sia associato un attributo color$[x]$ che può assumere i valori* bianco *o* nero. *Si descriva un algoritmo che dato un albero T calcola il numero dei nodi di T che hanno tutti i figli bianchi.*

Soluzione: Si può risolvere con una visita in ampiezza *breadth-first-search* verificando la condizione richiesta durante il ciclo while di visita di ciascun gruppo di fratelli. La chiamata esterna è COUNT(*root*$[T]$).

```
COUNT(x)
tot ← 0
if x = NIL
   then return tot
ENQUEUE(x,Q)
while not QUEUE-EMPTY[Q]
      do y ← HEAD[Q]
         DEQUEUE[Q]
         equal ← TRUE
         while y ≠ NIL
               do if COLOR[Y] = NERO
                     then equal ← FALSE
                  if CHILD[y] ≠ NIL
                     then ENQUEUE[CHILD[y],Q]
                  y ← SIBLING[y]
         if equal = TRUE
            then tot ← tot + 1
return tot
```

4. *Considerare le seguenti procedure e per ciascuna dire se può essere utilizzata per verificare una struttura dati nota. (Giustificare bene la risposta)*

```
VERIFICA-1(A, k)
test ← TRUE
for i ← k downto 2
      do if (A[i] < A[i/2])
            then test ← FALSE
return test
```

```
VERIFICA-2(T)
if T = NIL
   then return TRUE
   else  test ← VERIFICA-2(left[T])
         test ← test & VERIFICA-2(right[T])
         test ← test & KEY[left[T]] ≤ KEY[T]
         test ← test & KEY[right[T]] ≥ KEY[T]
return test
```

Soluzione: La procedura Verifica-1 può essere utilizzata per verificare la proprietà di min-heap.
Per quanto riguarda la procedura Verifica-2, anche sostituendo le istruzioni

```
test ← test & KEY[left[T]] ≤ KEY[T]
test ← test & KEY[right[T]] ≥ KEY[T]
```

con

```
if (left[T] ≠ NIL)
   then test ← test & KEY[left[T]] ≤ KEY[T]
if (right[T] ≠ NIL)
   then test ← test & KEY[right[T]] ≥ KEY[T]
```

la procedura NON può essere utilizzata per verificare la proprietà BST perchè non confronta la chiave di un nodo con TUTTE le chiavi nei sottoalberi sinistro e destro. Si consideri ad esempio il seguente albero (che NON è un BST dato che la chiave 12 è alla sinistra della chiave 10):

```
          10
         /   \
       8       15
     /   \
   4       12
```

5. *(a) Scrivere un algoritmo che trovi il massimo elemento di un array utilizzando un approccio divide-et-impera.*
(b) Determinare la complessità dell' algoritmo sviluppato.
(c) Dimostrare la correttezza dell' algoritmo sviluppato.

Soluzione: Vedi lezione di tutorato del 7 Novembre 2003

6. *Disegnare, se posibile, un albero Rosso/Nero con altezza uguale a 3 ed altezza nera uguale a 2.*

Soluzione: Ci sono molti alberi Rosso/Neri che possono essere presentati, un esempio può essere l'albero completo di altezza 3 che ha tutti i nodi neri a parte quelli sul primo livello, figli della radice.

7. *Dire in cosa consiste il problema delle collisioni in una tabella hash e spiegare le tecniche usate per affrontarlo.*

Soluzione: Vedi testo.