Algoritmi e Strutture Dati

Sessione estiva A.A. 2003/2004

Appello 09.07.04

1. *Per un certo problema sono stati trovati due possibili algoritmi risolutivi. Il tempo di esecuzione del primo è rappresentato dalla funzione T_1 riportata nel seguito al punto* (a) *mentre per il secondo è soddisfatta la relazione di ricorrenza riportate al punto* (b). *Si dica, giustificando la risposta, quale dei due algoritmi è da preferire nel caso si debbano risolvere problemi di grandi dimensioni.*

(a) $T_1(n) = 2n^2 + nlgn$

(b) $T_2(n) = 4T_2(\frac{n}{2}) + 5n^2 + 2log\ n^2$

Utilizziamo il Master Theorem per valutare la complessità asintotica del secondo algoritmo.

- *a* Si ha: $a = 4, b = 2$, quindi $\log_2 4 = 2$. Poiché $5n^2 + 2log\ n^2 = \Theta(n^2)$ siamo nel caso 2 del Master Method. Allora $T_2(n) = \Theta(n^2 \text{lg } n)$.

Dobbiamo quindi confrontare $2n^2 + n\text{lg}n$ con $n^2\text{lg } n$. Poiché $2n^2 + n\text{lg}n = O(n^2\text{lg } n)$ ma non vale il viceversa (vedi lezione di tutorato del 14.10.04), l'algoritmo da scegliere è il primo.

2. *Considerare le seguenti procedure A e B e determinare la complessità asintotica della procedura B(n) su input $n \in N$.*

```
A(n)
s ← 0
for i ← 1 to n
        do s ← s + i
return s
```

```
B(n)
m ← A(n)
s ← 1
for i ← 1 to m
        do s ← s * i
return s
```

Soluzione: La procedura $A(n)$ è lineare rispetto all'input n e calcola la somma dei primi n numeri interi. La chiamata di A(n) all'interno di B ha quindi costo di ordine $\Theta(n)$ ed assegna ad m un valore in $\Theta(n^2)$ (la somma dei primi n numeri interi). Pertanto il ciclo for della procedura $B(n)$ ha costo $\Theta(m) = \Theta(n^2)$ e la complessità asintotica globale della procedura B(n) è $\Theta(n^2)$.

3. *Sia T un albero binario di ricerca bilanciato con chiavi intere. Descrivere un algoritmo efficiente per verificare se tutti i nodi di T hanno chiave strettamente compresa tra due valori dati $k1$ e $k2$.*
Discutere la complessità dell'algoritmo in funzione del numero n di nodi dell'albero.

Soluzione: E' sufficiente verificare che il minimo elemento di T sia maggiore di $k1$ ed il massimo minore di $k2$. Essendo l'albero bilanciato questo può essere fatto in tempo $O(lgn)$, dove n è il numero di nodi.

4. *Definiamo una operazione* concatenate *il cui input è costituito da due insiemi di chiavi $S1$ ed $S2$ tali che le chiavi in $S1$ sono tutte minori o uguali delle chiavi in $S2$ e il cui output è la fusione dei due insiemi. Supponendo che gli insiemi $S1$ ed $S2$ siano rappresentati con alberi binari di ricerca, progettare un algoritmo per realizzare l'operazione* concatenate*. L'algoritmo deve avere complessità $O(h)$ nel caso peggiore, dove h è l'altezza massima dei due alberi,.*

Soluzione: Poichè non si considerano alberi bilanciati è sufficiente porre $S1$ come figlio sinistro del nodo $BSTmin(S2)$ (o $S2$ come figlio destro di $BSTmax(S1)$). L'esercizio non specifica se gli insiemi sono rappresentati con o senza duplicazioni; nel secondo caso è sufficiente controllare se la chiave del massimo di $S1$ è uguale alla chiave del minimo di $S2$ e in caso affermativo chiamare la $BSTdelete$ prima della fusione. Tutte queste operazioni hanno complessità $O(h)$.

5. *Definire la struttura dati* heap *e descriverne almeno una applicazione.*
Soluzione: Vedi testo.

6. *Sia L una lista concatenata il cui campo chiave contiene valori interi. Progettare un algoritmo che modifica L eliminando tutti gli elementi con chiave pari.*
Dimostrare la correttezza dell'algoritmo tramite l'uso di invarianti.

Soluzione: Possiamo scorrere la lista tenendo un puntatore *predy* all'ultimo elemento con chiave dispari considerato; scorrendo la lista aggiorneremo via via il campo next di *predy* fino a che questo non punterà correttamente al prossimo elemento con chiave dispari. All'inizio *predy* viene posto uguale alla costante NIL e il suo aggiornamento inizierà solo dopo che verrà incontato un elemento con chiave dispari; la testa della lista assumerà lo stesso valore di *predy* la prima volta che verrà trovato un elemento con chiave dispari. L'invariante è:
Gli elementi concatenati compresi tra HEAD[L] e *predy* sono tutti e soli gli elementi che nella lista originale sono compresi tra HEAD[L] e y e hanno chiave dispari.

```
EliminaPari(L)
y ← HEAD[L]
predy ← NIL
while y ≠ NIL
        do if (mod(KEY[y], 2) ≠ 0)
               then if (predy = NIL)
                          then HEAD[L] ← y
                     predy ← y
               else  if (predy ≠ NIL)
                          then NEXT[predy] ← NEXT[y]
           y ← NEXT[y]
```

7. *Definire l'operazione di rotazione a sinistra di un sottoalbero di un albero binario. Mostrare un esempio in cui è richiesta una tale operazione dopo l'inserimento o la cancellazione di una chiave in un albero rosso/nero.*

Soluzione: Si veda il testo. Un semplice esempio si trova nel caso 1 di RB-delete.