Introduzione agli Algoritmi
19 Giugno 2014
Prof. Emanuela Fachini (canale 1) e Prof. Irene Finocchi (canale 2)

Parte I

Le risposte non motivate non saranno prese in considerazione.
Negli esercizi di progettazione, prima di passare allo pseudocodice descrivete l'idea algoritmica soggiacente. Per tutti gli algoritmi progettati è necessario analizzare tempo di esecuzione e correttezza.

Esercizio 1

Si consideri la seguente funzione:

```
test (intero n)
        if n ≤ 1 then return 1
        k = n*n
        while k ≥ 1 do k = k/2
        return k + 3*test(n/4)
```

Scrivere e risolvere la relazione di ricorrenza che descrive il tempo di esecuzione T(n) della funzione *test*.

Soluzione.

La relazione di ricorrenza è
$T(n) = T(n/4) + O(\lg n)$
visto che il ciclo while è eseguito $\lg n^2$ volte e che $\lg n^2 = 2\lg n$.
Per iterazione:
$T(n) = T(n/4) + c\lg n = T(n/4^2) + c\lg n/4 + c\lg n = T(n/4^2) + c\lg n - 2c + c\lg n = T(n/4^2) + 2c\lg n - 2c = T(n/4^3) + c\lg(n/4^2) + 2c\lg n - 2c = T(n/4^3) + c\lg n - c\lg(4^2) + 2c\lg n - 2c = T(n/4^3) + 3c\lg n - 4c - 2c$
Si vede che ci sarà un termine $c\lg n$ sommato a sé stesso $\lg n$ volte.
Questo ci consente di ipotizzare che la soluzione sia in$O(\lg^2 n)$. Verifichiamo l'ipotesi induttivamente:
$T(m) \leq d\lg^2 m$, per un certo $d > 0$ e per ogni $m < n$
$T(n) \leq d\lg^2 n/4 + c\lg n \leq d(\lg n - 2)^2 + c\lg n = d(\lg^2 n + 4 - 4\lg n) + c\lg n = d\lg^2 n + 4d - 4d\lg n + c\lg n$
Possiamo dimostrare che $d\lg^2 n + 4d - 4d\lg n + c\lg n \leq d\lg^2 n$?
Sì perché questo è vero se $c\lg n \leq -4d + 4d\lg n$ e questo è vero prendendo $d=c$ e $n \geq 4$, per esempio.

Esercizio 2

Abbiamo due max Heap H1 e H2 aventi n1 e n2 elementi rispettivamente. Ogni elemento di H1 è maggiore di ogni elemento di H2.Come si può unificare i due maxheap in un unico maxheap con n1+n2 chiavi in O(n2)?

Si assuma che entrambi gli heaps sono rappresentati in arrays di dimensione n1+n2.

Soluzione

Aggiungere le chiavi di H2 in H1, con indici n1+1,…n1+n2 sembra essere la soluzione corretta che rispetta i vincoli di complessità. Ma facendo attenzione vediamo che se gli elementi che aggiungiamo sono in numero maggiore di quelli presenti in H1, si rischia di non ottenere un maxHeap.
Esempio:

H1=

40	20	30							
1	2	3	4	5	6	7	8	9	10

H2=

16	8	13	5	6	11	10			
1	2	3	4	5	6	7	8	9	10

H1+H2=

40	20	30	16	8	13	5	6	11	10
1	2	3	4	5	6	7	8	9	10

ma il padre di (H1+H2)[10] = 10 è (H1+H2)[5] = 8, quindi si ha una violazione della proprietà del max-Heap. In questo caso bisogna quindi chiamare la Build_Max_Heap su H1+H2 con un costo O(n1+n2).
Se però prendiamo
H1=

40	20	30			
1	2	3	4	5	6

Introduzione agli Algoritmi
19 Giugno 2014
Prof. Emanuela Fachini (canale 1) e Prof. Irene Finocchi (canale 2)

H2=

16	8	13			
1	2	3	4	5	6

H1+H2=

40	20	30	16	8	13
1	2	3	4	5	6

L'array ottenuto è un maxheap! La differenza nei due casi dipende dal numero degli elementi.
Se n1≥ n2 allora i nodi aggiunti diventano figli delle foglie. Infatti:
caso 1 se n1= 2n+1, cioè è dispari, le foglie sono i nodi da n+1 fino a 2n+1; ci sono dunque n+1 foglie e n nodi interni; si possono aggiungere fino a 2n+2 figli alle n+1 foglie e 2n+2>n2 visto che 2n+1 = n1 ≥ n2 per ipotesi;
caso 2 se n1 = 2n, cioè è pari, le foglie sono i nodi da n+1 a 2n, quindi ci sono n foglie e n nodi interni, si possono avere fino a 2n figli delle foglie e 2n=n1≥n2;
Se invece n1< n2 è necessario chiamare la Build_Max_Heap per eliminare tutti i possibili casi di violazione della proprietà del Max_Heap.
La complessità è O(n2) nel primo caso perché si tratta solo di copiare gli elementi di H2 nel max Heap H1, nel secondo perchè, essendo n1<n2 possiamo dire che O(n1+n2)=O(n2)

.

Introduzione agli Algoritmi
19 giugno 2014
Prof. Emanuela Fachini (canale 1) e Prof. Irene Finocchi (canale 2)

Parte II

Esercizio 1

1. Sia dato un ABR T, la chiave di una sua foglia, x e quella del padre di x, y.
   E' sempre vero che o
   - y è la più piccola chiave in T più grande di x
     oppure
   - y è la più grande chiave in T più piccola di x?

2. Rispondere alla stessa domanda del punto (a), assumendo però che la chiave x non sia contenuta in una foglia, ma in un nodo avente un unico figlio.

3. Sia ancora x un nodo di un ABR T. Si descriva sinteticamente la procedura per individuare il successivo di x in T.

Soluzione

Le prime due domande riguardano il calcolo del successivo e del precedente in casi particolari.
L'affermazione 1 è vera, infatti:

- se x è figlio destro di y, vuol dire che la sua chiave è maggiore di quella del padre, d'altro canto il padre è il primo nodo risalendo verso la radice nel cui sottoalbero destro si trova x, quindi è il maggiore tra i più piccoli, cioè il precedente.
- se x è figlio sinistro di y, vuol dire che la sua chiave è minore di quella del padre, d'altro canto il padre è il primo nodo risalendo verso la radice nel cui sottoalbero sinistro si trova x, quindi è il minore tra i più grandi, cioè il successivo.

L'affermazione 2 è falsa, infatti:

- se x ha un figlio destro z il suo successivo è il minimo nel sottoalbero radicato in z, quindi il padre di x non può esserlo, d'altro canto se x è figlio sinistro e suo padre figlio destro il precedente è il padre di y.

- se x ha un figlio sinistro z, simmetricamente il suo precedente è il massimo nel sottoalbero radicato in z e se x è figlio destro e suo padre figlio sinistro il successivo è il padre di y.

Per la tre si veda il libro di testo o gli appunti in rete.

Esercizio 2

Scrivete un algoritmo che, preso in input un ABR T e il puntatore a un suo nodo x, restituisce il numero degli elementi di chiave minore di quella di x e il numero degli elementi di chiave maggiore.
Si motivi l'algoritmo dimostrandone la correttezza e se analizzi la complessità di tempo.

Si tratta di modificare la visita inorder (o anche un'altra visita) in modo da confrontare in ogni passo il nodo visitato con la chiave di x, incrementando un contatore dei minori nel caso la chiave del nodo visitato risulti minore di quella di x e incrementando un contatore dei maggiori altrimenti.
Se si sceglie la visita inorder si può usarne l'implementazione iterativa che chiama la funzione successor a partire dal minimo fino al massimo.
Se si preferisce la versione ricorsiva, basta dotare la funzione di visita di un output (min,mag) che viene opportunamente aggiornato ad ogni chiamata.
La complessità è banalmente O(n), dove n è il numero dei nodi dell'ABR.