**INTRODUZIONE AGLI ALGORITMI**
**Seconda prova di Esonero**
**30 Maggio 2019**
**Prof.ssa Calamoneri - Prof.ssa Fachini - Prof.ssa Petreschi**

## Esercizio 1.

Dato un albero binario di ricerca *T*, sia *a* la chiave di un suo nodo *x*. Si risponda alle seguenti domande dimostrando le risposte:

1. E' vero che se *x* ha due figli, allora il nodo contenente il successivo di *a* non ha figlio sinistro?
2. E' vero che se *x* è un nodo foglia e il padre di *x* ha chiave di valore *b*, allora o *b* o è il successivo di *a* o è il precedente di *a*?
3. Si considerino due operazioni successive su *T*. La prima inserisce un nodo *y* di chiave *k* e la seconda elimina lo stesso nodo *y*. L'albero risultante è uguale a *T*?
4. Si supponga che *T* sia un albero AVL e su questo si eseguano le operazioni descritte al punto 3. Che tipo di risposta avremo?

**Sol.**

Risposta 1. E' vero perchè il suo successivo è il minimo nel sottoalbero destro e in quanto tale non può avere un figlio sinistro, la cui chiave dovrebbe essere minore.

Risposta 2. E' vero perchè se x è una foglia non ha figli e quindi il suo successivo e il suo precedente va ricercato tra i suoi antenati. Se x è un figlio sinistro, suo padre è il primo nodo risalendo da x che lo ha nel sottoalbero sinistro quindi è il suo successivo. Se z di chiave c fosse il padre di b, e y fosse figlio sinistro di z si avrebbe che a < b < c, se invece y fosse figlio destro di z allora c < a < b

Se invece x è figlio destro allora suo padre è il primo nodo incontrato risalendo da x che lo ha nel sottoalbero destro e quindi il padre è il suo precedente.

Risposta 3. E' vero, infatti un nuovo nodo in un ABR è sempre inserito come foglia, quindi la sua cancellazione ripristina l'albero che si aveva prima dell'inserimento.

Risposta 4. In generale non è detto che l'albero sia lo stesso, se infatti si esegue una rotazione allora la cancellazione del nodo può lasciare un albero diverso da quello di partenza. Per esempio:

T= 100
50 150
30

Inseriamo 20
100
50 150
30
20

L'albero è sbilanciato e il nodo con FB = 2 è il nodo di chiave 50, con una rotazione a destra su 50 si ribilancia l'albero, siamo infatti in un caso left-left, visto che il nodo 30 ha FB = 1. Dopo la rotazione si ha

100
30 150
20 50

Se ora si cancella 20 si ottiene

100
30 150
50

e questo albero è diverso da T.

**Esercizio 2.**

Dato un vettore *A* di *n* elementi (con *n* potenza di 4), considerare la variante dell'algoritmo di Merge Sort che ordina il vettore *A* dividendolo ad ogni passo in 4 sottovettori invece che in due.

Supponendo di avere una funzione *Fondi(A, $ind_1$, $ind_2$, $ind_3$, $ind_4$, $ind_5$)* che fonde in un unico sottovettore ordinato dall'indice $ind_1$ all'indice $ind_5$ i 4 sottovettori ordinati che vanno dall'indice $ind_i$ all'indice $ind_{i+1}$, *i=1, …, 4*, si scriva (giustificandola) l'equazione di ricorrenza che ne esprime il costo computazionale e la si risolva utilizzando il metodo di sostituzione.

**Sol.**

La relazione di ricorrenza è $T(n) = 4T(n/4) + \Theta(n)$, visto che si hanno 4 chiamate della funzione mergeSort su n/4 elementi e la fondi lavora in tempo lineare.

Per risolverla esplicitiamo le costanti e fissiamo il costo del caso base:

$T(n) = 4T(n/4) + cn$ se $n \geq 2$

$T(n) = d$ altrimenti

Sia $n = 4^h$, l'albero della ritorsione è

cn …….. cn

cn/4 cn/4 cn/4 cn/4 ……4cn/4= cn

$cn/4^{h-1}$ $cn/4^{h-1}$ …….. cn

d d …….$4^h$d

Quindi il costo su ogni livello è cn, possiamo quindi concludere che $T(n) = 4^h d + \sum_{i=0,\ldots,h-1} cn = O(n \lg n)$

Prova induttiva che $T(n) = O(n \lg n)$

Dimostriamo che esistono k ed $n_0$ tali che $T(n) \leq k \, n\log_4 n$.

Supponiamo che la tesi sia vera per ogni $m < n$ e cosideriamo

$T(n) = 4T(n/4) + cn \leq 4 \, kn/4\log_4 n/4 + cn = nk\log_4 n - kn + cn$

Controlliamo se $kn\log_4 n - kn + cn \leq kn\log_4 n$ cioè se esiste un valore di k per cui $- kn + cn \leq 0$.Tale valore esiste ed è per esempio k= c.

Andando a vedere il caso base

$T(1) = d$ che non è maggiorato da $\log_4 1$ che è 0.

Prendiamo allora $T(4) = 4T(1) + 4c = 4(d+c)$ e vediamo per

quale valore di k si ha $4(d+c) \leq 4k\log_4 4 = 4k$. Basta prendere $k= d+c$ e $n_0 = 4$. Si può quindi concludere che $T(n) = O(n\lg n)$.
Il limite inferiore che si ricava dalla definizione è lineare, ma si può dimostrare in modo analogo che vale anche $T(n) = \Omega(n \lg n)$.

**Esercizio 3.**

Si consideri un albero binario qualunque, memorizzato tramite strutture e puntatori e dato tramite il puntatore *T* alla sua radice.

Le strutture che rappresentano i nodi dell'albero sono dotate di un campo aggiuntivo *bil*, che è inizializzato a 0, e che dovrebbe contenere per ogni nodo la differenza tra l'altezza del sottoalbero sinistro e l'altezza del sottoalbero destro.

Si progetti un algoritmo che scriva il corretto valore di *bil* in ogni nodo dell'albero, se ne scriva lo pseudocodice e se ne calcoli il tempo di esecuzione asintotico.

Sol.

E' una modifica dell'algoritmo per calcolare l'altezza di un albero binario. Si tratta di calcolare l'altezza del sotto albero radicato in ogni nodo e poi aggiornare di conseguenza il campo *bil.* Se infatti il sottoalbero sinistro ha altezza h1 e quello destro h2 allora il nodo padre avrà un campo *bil* pari a h1-h2*.*

Si tratta di una visita in post order quindi il tempo di esecuzione asintotico è O(n).

Pseudocodice:

FattoriBil(T)
Input: T è il puntatore alla radice di un albero binario
Output l'altezza dell'albero e la modifica del campo bel di ogni nodo con la differenza tra l'altezza del sottoalbero sinistro e l'altezza del sottoalbero destro

if T = NULL the return -1

```
h1 = FattoriBil(T.left)
h2 = FattoriBil(T.right)
T.bil = h1- h2
return max(h1.h2) +1
```