B

Introduzione agli algoritmi
Prova di esame del 30/6/2016
Prof.sse E. Fachini - R. Petreschi

Si consideri l'operazione di inserimento in un AVL. Si supponga di trovarsi nel caso in cui una rotazione a destra è sufficiente a ribilanciare l'albero. Si illustri il processo di aggiornamento dei fattori di bilanciamento. Si descriva, anche aiutandosi con dei disegni, la situazione del sotto albero di radice x, T(x), su cui viene eseguita la rotazione, prima dell'inserimento, dopo l'inserimento e dopo la rotazione. Si mostri come vengono modificati i fattori di bilanciamento dei nodi implicati nella rotazione. Si spieghi perché non è necessario aggiornare i fattori di bilanciamento di tutti gli altri nodi nel cammino da x alla radice, così come dei nodi che non appartengono al cammino nuovo nodo inserito-radice. Si concluda con l'analisi del tempo asintotico dell'operazione di inserimento in questo caso.

Sol. Si consultino libro di testo e lucidi.

2. Si determini il tempo di esecuzione asintotico T(n) della seguente funzione:

```
analizzami(A,i,j)
n = j - i+1
c = 1
m = (i + j +1)/2
d = m
while d >1 do
      for j=1 to n do c++
      d = d - 2
if n>1 then
         return analizzami(A,i,m-1) + analizzami(A,m+1,j)
```

Sol. La relazione di ricorrenza da risolvere è
$T(n)=2T(n/2) + cn^2$ se $n>1$ $T(n) = d$ altrimenti, dove c e d sono costanti positive.
Si dimostra facilmente che ha soluzione in $O(n^2)$.

3. Sia dato un albero binario di ricerca, T, implementato con strutture a puntatore in modo che per ogni nodo x siano presenti oltre al campo

chiave e ai campi puntatori ai figli e al padre, un campo, succ, puntatore al nodo la cui chiave è la successiva in T di quella di x. Si mostri come deve essere modificato l' algoritmo di cancellazione di una chiave, in modo che tutti i campi del nuovo nodo siano opportunamente valorizzati. Si valuti il tempo asintotico di esecuzione dell'algoritmo presentato.
Questa operazione si avvantaggia del nuovo campo aggiunto in ogni nodo? Si argomenti senza necessariamente fornire lo pseudocodice dell'operazione nel caso in cui non sia presente il campo succ.

Sol.
Ricordando l'algoritmo per la cancellazione di una chiave k in un ABR, sappiamo che nel caso in cui la chiave da cancellare è in un nodo con due figli, il nodo che verrà cancellato è quello di chiave successiva a k, e quindi si potrà utilizzare il campo succ per accedere in tempo costante a quel nodo. Bisogna anche aggiornare il campo succ del nodo di chiave precedente a k. Poiché il nodo di chiave k va cercato, si può nella discesa verso il nodo, aggiungere un parametro, prec, per tenere conto dell'ultimo nodo attraversato scendendo a destra. Se il nodo di chiave k ha un figlio sinistro il precedente è quello di chiave massima nel sotto albero sinistro del nodo, altrimenti è il nodo prec salvato nella discesa verso k.
Per cancellare un nodo, avendo a disposizione il puntatore al nodo da cancellare, x e il puntatore al nodo di chiave precedente a k, prec, procediamo come nell'algoritmo della cancellazione in programma, apportando le necessarie modifiche.
In generale è solo il nodo prec che deve aggiornare il suo campo succ con x.succ.

Se il nodo x ha al più un figlio allora semplicemente il campo succ del nodo precedente deve prendere il valore del campo succ di x e il padre di x diventerà padre del figlio di x, se c'è.
Se il nodo ha due figli, copiamo la chiave del nodo y=x.succ nel nodo x, il campo succ del nodo precedente non deve essere modificato, perché la chiave di y è proprio il successivo di k, mentre il campo succ di x, con la nuova chiave, deve prendere il valore del campo succ di y. Poi il padre del nodo y prende il figlio destro di y come proprio figlio sinistro.

Il tempo di esecuzione della cancellazione è in O(h), dove h è l'altezza dell'albero, perché le modifiche effettuate localmente sono di costo costante ma il numero di passi per la ricerca della chiave da cancellare più quello per trovarne il precedente è al più pari all'altezza.

Pseudocodice:

Delete(T,k)
input: T è un puntatore alla radice di un albero binario, k è la chiave da cancellare
prec: T è un ABR non nullo e la chiave k è presente in T
output: l'albero T privato della chiave k
DeleteSucc(T,k,NIL)

La funzione chiamata è così specificata:

DeleteSucc(T,k,prec)
input: T è un puntatore alla radice di un albero binario, k è la chiave da cancellare e prec è un puntatore alla radice di un albero binario
prec: T è un ABR non nullo e la chiave k è presente in T
output: l'albero T viene modificato con la cancellazione di k e l'opportuno aggiornamento dei campi succ dei vari nodi coinvolti.

pseudocodice di DeleteSucc:

```
if T.key = k then
  { if T.right == NIL  then
     {if T.p ≠ NIL
        if T.p.right == T then T.p.right = T.left else
             T.p.left = T.left
      else T = T.left;
```

```
      /Se T è figlio destro/sinistro senza figlio destro il suo
eventuale figlio sinistro diventa figlio destro/sinistro di suo
padre.
       if T.left ≠ NIL then
                {prec = Maximum(T.left); T.left.p = T.p}
       if prec ≠ NIL then prec.succ = T.succ
      }
    else /T ha figlio il figlio destro
       if T.left == NIL /T non ha figlio il figlio sinistro  then
         {if T.p ≠ NIL
             if T.p.left == T then T.p.left = T.right else
                T.p.right = T.right
          else T = T.right;
       /Se T è figlio sinistro/destro il suo eventuale figlio de-
stro diventa figlio sinistro/destro di suo padre.
       if T.right ≠ NIL then T.right.p = T.p
       if prec ≠ NIL then prec.succ = x.succ
        }
if T.left ≠ NIL and T.right ≠ NIL then
    {T.key = T.succ.key
     T.succ = T.succ.succ
/il campo succ del precedente di T non deve essere modifi-
cato, visto che sostituiamo la chiave di T con quella del suo
successivo. Se si volesse invece sostituire tutto il nodo, bi-
sognerebbe aggiornare anche questo puntatore.
       y = T.succ  /si deve cancellare il nodo y che è un figlio
sinistro che non ha il figlio sinistro perché  è il minimo nel
sotto albero destro di T
     if y.right ≠ NIL then y.right.p = y.p;
     y.p.left = y.right
else
   if T.key < k then DeleteSucc(T.right,k,T) else
                        DeleteSucc(T.left,k,prec)
```

L’operazione che in modo più evidente si avvantaggia della presenza del campo al successivo è la visita inorder, che pur essendo sempre in Θ(n), dove n è il numero dei nodi, si avvallerebbe dell’individuazione in tempo costante del successivo di ogni nodo.
Anche la cancellazione migliora, pur rimanendo in O(h), tenendo conto del tempo necessario a individuare il nodo da cancellare e il precedente.