CSCI-GA.1170-001/002 Fundamental Algorithms October 29, 2015

Solutions to Problem 2 of Homework 5 (10 (+5) points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, October 13*

Assume you are given a binary search tree T of height h and with n elements in it. For simplicity, assume all the elements are distinct.

(a) (5 pts) Use a slight modification of the PostOrder-Tree-Walk procedure to argue that in time $\Theta(n)$ you can compute, for every node v, the number of even nodes (call it $even(v)$) in v's sub-tree.
(**Hint**: In addition to $even(v)$, also compute the total number of nodes in v's subtree.)

Solution:

Let T be the given tree and x be $T.root$, then number of even nodes in x, $even(x)$ will be

- If $x.value$ is even, then $even(x.left) + even(x.right) + 1$
- If $x.value$ is odd, then $even(x.left) + even(x.right)$

Algorithm: NumEvenNodes(T)
$x \leftarrow T.root$
if *x is NULL* **then**
  Return 0
$even_l \leftarrow$ NumEvenNodes($x.left$)
$even_r \leftarrow$ NumEvenNodes($x.right$)
if *$x.value$ is divisible by 2* **then**
  $even(x) \leftarrow even_l + even_r + 1$
else
  $even(x) \leftarrow even_l + even_r$
Return $even(x)$

Algorithm 2: Algorithm to calculate number of even nodes in a Tree T

□

(b) (5 pts) Now that each node v contains the value $even(v)$, show how to keep maintaining this value for each successive $Insert$ operation. Namely, show how to perform an $Insert$ operation in time $O(h)$, while correctly maintaining all the $even(v)$ values.

Solution:

Let z be the node being inserted

- If $z.key$ is odd then just insert z at the appropriate position. This step takes $O(h)$ time
- If $z.key$ is even, then for every node v, z visits while being inserted add 1 to $even(v)$. This step takes $O(h)$ time

Therefore, the updated Insert algorithm takes $O(h)$ time

```
Algorithm: INSERT(T, z)
y ← NULL
x ← T.root
if z.key is divisible by 2 then
    z.even = 1
else
    z.even = 0
while x is not NULL do
    if z.key is divisible by 2 then
        x.even = x.even + 1
    y = x
    if z.key < x.key then
        x = x.left
    else
        x = x.right
end
z.p = y
if y is NULL then
    T.root ← z
else if z.key < y.key then
    y.left ← z
else
    y.right ← z
```

Algorithm 3: Insert procedure in $O(h)$ while maintaining all the $even(v)$ values

$\square$

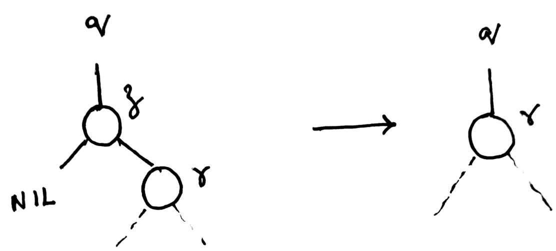
(c)* (5 pts) (**Extra Credit:**) Similar to part (b), but do it for the *Delete* operation. Namely, show how to perform a *Delete* operation in time $O(h)$, while correctly maintaining all the $even(v)$ values.

Solution:

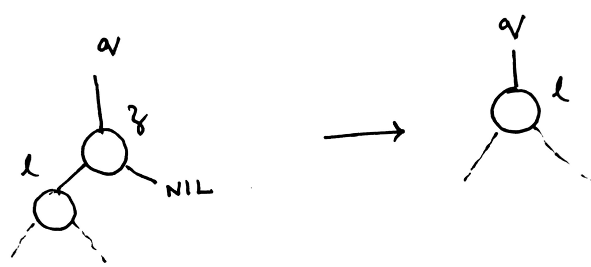
Let z be the node to be deleted. Then we have the following 4 cases and in all the cases first we do the following operation

If $z.key$ is even then starting from z go up until the root and for each node v in the path, decrease $even(v)$ by 1. Therefore, this takes $O(h)$ time

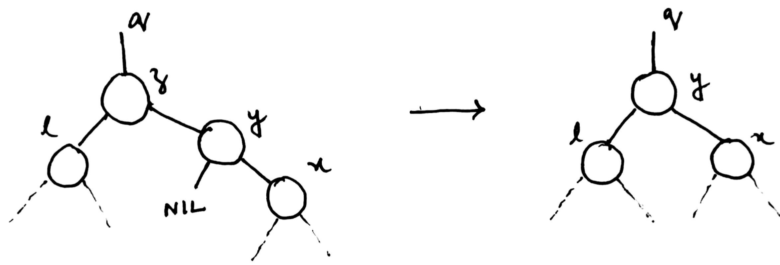
- Node z has no left child. We replace z by its right child r which may or may not be NIL. This takes only $O(1)$ time



- Node z has a left child l but no right child. We replace z by l. This takes only $O(1)$ time

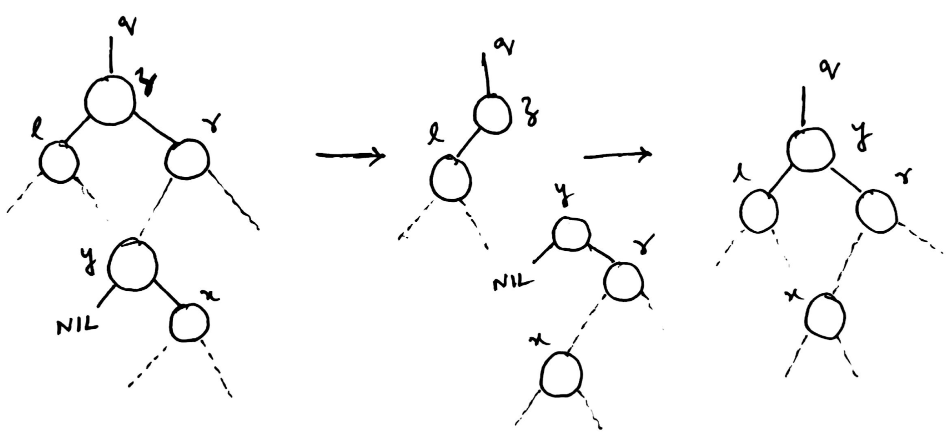


- Node z has two children, its left child is node l, its right child is its successor y and y's right child is node x. We replace z by y, updating y's left child to become l, but leaving x as y's right child and update $even(y) = even(y) + even(l)$. This takes only $O(1)$ time



update $even(y) = even(y) + even(l)$. Therefore, this takes $O(h)$ time

- Node z has two children (left child l and right child r), and its successor $y \neq r$ lies within the subtree rooted at r. If y is even then for each node u starting from $y.p$ to r decrease $even(u)$ by 1 and replace y by it's own right child x, set y to be r's parent and update $even(y) = 1 + even(r)$.



Then set y to be q's child and again update $even(y) = even(y) + even(l)$. This, takes $O(h)$ time

Therefore by adding some extra computations which take $O(h)$ time we can still perform the delete operation while correctly maintaining the $even(v)$ values in $O(h)$ time $\square$