CTCI 4.6: Successor

January 24, 2018

Question

Write an algorithm to find the "next" node (i.e., in-order successor) of a given node in a binary search tree. You may assume that each node has a link to its parent.

Explanation and Algorithm

Recall in-order traversal: traverse left subtree, current node, and right subtree. Based on this, if we apply in-order traversal to a hypothetical node, the order will be left subtree, current side, and right subtree. This means that the next node is on the right side. Here's how to find out which node on the right side. It should be the leftmost node on the right subtree. //

We have to deal with the case where the entire tree has no right subtree. If a node n doesn't have a right subtree, that means we're done traversing the subtree of that node. We then go to n's parent we'll call q. If n was left of q, the next node we traverse should be q.If n is to the left of q, that means we need to traverse upwards from q until we find a node x that has not been fully traversed. We know a node hasn't been fully traversed if the left node has been fully traversed, but its parent has not.

Hints

1. Recall the basic algorithm and code for in order traversal. This is key for implementing this program.
2. Based on this information, what does it mean for a node to not be fully traversed?
3.

Code

```
/*Pseudocode*/

Node inorderSuccessor(Node n){
   if(n has right subtree){
      return leftmost child of right subtree
    }else{
      while(n is right child of n.parent){
         n = n.parent; //Go up
        }
        return n.parent //parent has not been traversed
    }
}
/*Answer 2 properly handles null case while implementing algo above */

TreeNode inorderSuccessor(Node n)
   if(n.right != null){
      return null;
    }

    if(n != null){ //found right children; return leftmost node of right
         subtree
      return leftMostChild(n.right);
    }else{
      TreeNode q = n;
        TreeNode x = q.parent;
        //go up until arrived at left instead of right
        while(x != null && x.left != q){
         q = x;
            x = x.parent;
        }
        return x.;
    }

TreeNode leftMostchild(TreeNode n){
   if(n == null){
      return null;
    }
   while(n.left != null){
      n = n.left;
    }
    return n;
}
```

Run time analysis

Runs in $O(h)$ where h is the height of the tree.