CTCI 4.3: List of Depths

December 7, 2017

Question

Given a binary tree, design an algorithm which creates a linked list of all the nodes at each depth (e.g., if you have a tree with depth D, you'll have D linked lists).

Explanation and Algorithm

At first it might seem like a level-by-level traversal is necessary, but it actually isn't. We could implement a modification of pre-order traversal. We will pass level+1 to the next recursive call. Then you could implement depth-first search or breadth first search.

Hints

1.

Code

```
/*Answer 1: Depth First Search */

void createLevelLinkedList(TreeNode root,
    ArrayList<LinkedList<TreeNode>> lists, int level){
   //base case
    if(root == null){
      return;
    }
    LinkedList<TreeNode> list = null;
    if(lists.size() == level){ //level not contained in list
      list = new LinkedList<TreeNode>();
    }else{
      list = list.get(level);
    }
    list.add(root);
    createLevelLinkedList(root.left, lists, level+1);
    createLevelLinkedList(root.right, lists, level+1);
}

ArrayList<LinkedList<TreeNode>>createLevelLinkedList(TreeNode root) {
   ArrayList<LinkedList<TreeNode>> lists = new
       ArrayList<LinkedList<TreeNode>>();
   createLevelLinkedList(root, lists, 0);
   return lists;
}


/*Answer 2: Breadth First Search */
void createLevelLinkedList(TreeNode root,
    ArrayList<LinkedList<TreeNode>> lists, int level){
   ArrayList<LinkedList<TreeNode>> result = new
       ArrayList<LinkedList<TreeNode>>();
    LinkedList<TreeNode> curr = new LinkedList<TreeNode>();
    if(root != null){
      curr.add(root);
    }

    while(curr.size() > 0){
      result.add(curr); //add previous level
      LinkedList<TreeNode> parents = current; //go to next level
       curr = new LinkedList<TreeNode>();
       for(TreeNode parent : parents){
        if(parent.left != null){
              curr.add(parent.left);
           }
           if(parent.right != null){
              curr.add(parent.right);
           }
       }
    }
   return result;
}
```

Run time analysis

Both solutions run on O(n) time. They only differ in space efficiency. The first solution uses O($log(n)$) recursive calls to add a new level. The second solution doesn't require this extra space.