CS 120: Intro to Computer Programming II

In-Class Activity - 13 Stacks and Queues - Day 2

Today, we're going to explore a problem where a Queue would be very useful. But first, we're going to try to solve it using a recursive function, and see what is difficult about that strategy.

Save the code from today's ICA - you will find it useful in our next day.

Activity 1 - Turn in this one

Today, you will be writing a recursive function `_shallow_match(tree, letter)`, which will search the tree for a node that matches a special condition. If it finds such a node, it returns a tuple, $(depth, val)$ where `depth` is the depth of the node (measured as the distance from the root) and `val` is the value at that node.

You may assume that the tree is a binary tree, but do **not** assume that it's a BST. Also, it might be empty.

If multiple nodes match the search condition, then return the depth and value of the shallowest one. If there is a tie, you may choose either alternative.

If no nodes in the tree match the condition, then you must return a tuple $(depth, None)$ - where the `depth` here can be **any** integer value. (While it must be an integer, it will be otherwise ignored.)

So, what sort of nodes are you looking for? Nodes where `letter` is the first character of the value.

For this first Activity, we are going to discuss the problem before we write it. Write down the group's answers to each question.

- What will your function return in the case of an empty tree?
- Suppose that you find a node which matches the search condition. Is it still necessary to recurse into the children of that node, or not?
- After you recurse into the children, what test will you use to determine whether or not they found something?
- Suppose that you recurse into the left child of a node, and it found a matching node. Is it still required to recurse into the right child, as well?
- If the current node doesn't match the search condition, and neither of your child trees find one, either, then what should this function return?
- If the current node doesn't match the search condition, and only one of the child trees finds a matching node, then what should you return?
- If the current node doesn't match the search condition, and both of the child trees find a matching nodes, then how do you know which to return?
- If your child finds a node with depth 3 (meaning that there are 3 steps from your child node to the node they found), how far away is that node from you?

(activity continues on the next page)

Activity 2 - Turn in this one

Now, write the function `_shallow_word(tree, letter)`. **Make sure to write some test code,** to confirm it works properly.

Follow this up by writing `shallow_word(tree, letter)`, a non-recursive function which uses the first one as a helper. The key difference is that this function returns **only** the value.

Solution:

```
def _shallow_tree(tree, letter):
    if tree is None:
        return (0,None)
    if tree.val[0] == letter:
        return (0, tree.val)

    # get depth & value from both sides.  Note that some values might
    # be None.
    ld,lv = _shallow_tree(tree.left,  letter)
    rd,rv = _shallow_tree(tree.right, letter)

    # if the left side returned a value of None, then we can just return
    # whatever the right side returned (which might be None, or a value).
    # Remember to increment the depth - the depth doesn't matter if the
    # value is None, but it's critical if the value is not.
    if lv is None:
        return (rd+1, rv)

    # similar for None on the right side
    if rv is None:
        return (ld+1, lv)

    # if we get here, then both sides returned non-None, and thus we must
    # compare depths.  But since the depth is first in the tuple,, min()
    # makes this easy!
    return min( (ld+1,lv), (rd+1,rv) )



def shallow_tree(tree, letter):
    # we don't care about the depth of the answer that is returned to
    # us - we *only* care about the value.
    depth,val = _shallow_tree(tree, letter)
    return val
```

Activity 3 - Optional

OPTIONAL. Complete this if you have time, and turn it in. If you don't have time, you may report to your TA that you ran out of time.

Consider a **very large** tree - hundreds or thousands of nodes. But, as it so happens, the node `root.right.right`

matches the criteria - you are going to return it from the function, eventually.

However, your function - which is recursive - don't notice this very early on. Instead, it will do quite a lot of work before it finds the node. (In some student implementations, it may actually search the **entire** tree; in most student implementations, it will search most of the tree.)

Discuss with your group why humans can **quickly** recognize that the node `root.right.right` - which is relatively close to the root - should be returned, but for computers it's a slow, laborious process.

Then, discuss with your group if you could find a strategy to **make the computer just as fast.**