Gmail

rishav ghosh <rishavghosh605@gmail.com>

How to Formulaically Solve Tree Interview Questions

1 message

Daily Coding Problem <founders@dailycodingproblem.com> Mon, May 13, 2019 at 4:01 AM
To: rishavghosh605@gmail.com

Daily Coding Problem

Hey there,

Today I'd like to give you some tips on how to solve tree-based interview questions. Tree questions are very common at top tech company interviews. I had two tree questions in my Google onsite interviews and one during my Facebook onsite interviews. An awesome thing about them is that they can be formulaically solved every single time. It doesn't involve any genius insight. Let me show you how.

Instead of being too abstract, let's just dive right into an easy binary tree question. Then I'll walk through how to solve it and we can go into a harder problem after:

> Given the root to a binary tree, count the total number of nodes there are.

Before we move on further, feel free to take a moment to think about the answer!

Solving any binary tree question involves just two steps.

First is solving the base case. This usually means solving the leaf node case (a leaf node has no left or right children) or the null case. For the above problem, we can see that a null should represent 0 nodes while a leaf node should represent 1 node.

Second is the recursive step. Assuming you knew the solution to the left subtree and the right subtree, how could you combine the two results to give you the final solution? It's important to not get caught up

on how this works and just have faith that it works. If you start tracing the recursion, you're going to needlessly use up time and energy during the interview. Intuitively though, it works for similar reasons as why regular induction works. P(0) or the base case works which causes P(1) or the leaf node to work which causes P(2) to work and so on. For this problem, it's easy to combine the results of the left and right subtrees. Just add the two numbers and then another 1 for the root. Here's the code:

```
def count(node):
  return count(node.left) + count(node.right) + 1 if node else 0
```

You certainly won't get a question this easy but the process is the same for trickier problems. Here's another problem:

> Given the root to a binary tree, return the deepest node.

Base case for this question actually can't be null, because it's not a real result that can be combined (null is not a node). Here we should use the leaf node as the base case and return itself.

The recursive step for this problem is a little bit different because we can't actually use the results of the left and right subtrees directly. So we need to ask, what other information do we need to solve this question? It turns out if we tagged with each subresult node their depths, we could get the final solution by picking the higher depth leaf and then incrementing it:

```
def deepest(node):
    if node and not node.left and not node.right:
        return (node, 1) # Leaf and its depth

    if not node.left: # Then the deepest node is on the right
subtree
        return increment_depth(deepest(node.right))
    elif not node.right: # Then the deepest node is on the left
subtree
        return increment_depth(deepest(node.left))

    return increment_depth(
            max(deepest(node.left), deepest(node.right),
                    key=lambda x: x[1])) # Pick higher depth tuple
and then increment its depth

def increment_depth(node_depth_tuple):
```

```
    node, depth = node_depth_tuple
    return (node, depth + 1)
```

Best of luck!

Marc

If you liked this guide, feel free to forward it along! As always, shoot us an email if there's anything we can help with!
