Daily Coding Problem #36

Problem

This problem was asked by Dropbox.

Given the root to a binary search tree, find the second largest node in the tree.

Solution

An in-order traversal of the binary search tree would give us all the nodes of the tree in sorted order. So the naive solution here might be do an in-order traversal of the tree, store it in an array, and return the second-to-last element in the array.

This takes O(N) time and space since we have to go through and store every node in the tree.

We can do better. Notice that the in-order traversal explores always the left node first before the current node. We could do something similar to that by exploring the right node first.

Let's do a reverse in-order traversal, where we first call ourselves recursively on the right node. Because it's reversed, that should give us the binary tree in reverse sorted order.

So we can keep a counter, and once we start processing the current node we can increment the counter. Once it hits 2, that means the current node we're looking at is the second largest, so we can stuff it in a variable and eventually return that.

```
def second_largest(root):
    def inorder(node):
        if not node or count[0] == 2:
```

```
            return

        if node.right:
            inorder(node.right)

        count[0] += 1
        if count[0] == 2:
            val.append(node.val)
            return

        if node.left:
            inorder(node.left)

    count = [0]
    val = []
    inorder(root)
    return val[0]
```

Unfortunately because of Python's demented scoping rules, we have to wrap `count` and `val` in a list. Ugly!