Daily Coding Problem #223

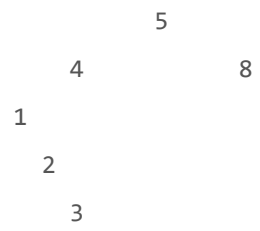
Problem

This problem was asked by Palantir.

Typically, an implementation of in-order traversal of a binary tree has `O(h)` space complexity, where `h` is the height of the tree. Write a program to compute the in-order traversal of a binary tree using `O(1)` space.

Solution

The basic idea is to restructure the tree while we traverse it, so that going to the right will always go to the "correct" next node.



Let's take the tree above. When printing out the nodes, the logical node after 4 should be 5, since 4 has no right child. Similarly, after 3 would have to come 4. Speaking more programmatically, we want to make it so that the rightmost descendant of any node's left child points to itself.

For the example above, we can do this starting with the root. We first find that the left child of the root is 4. Then, since 4 has no right descendants besides itself, we make 5 the right child of 4. Following this, we descend left and follow the same process. The left child of 4 is 1, and the rightmost descendant of 1 is 3, so we should make 4 the right child of 3.

What happens when we can no longer go left? In such cases, we are ready to print out the value of the current node and go right. For this tree, the algorithm described so far will successfully print out 1, 2, and 3, and move us back to 4.

There is a problem, though. Once we return to 4, we will proceed to loop back over 1, 2, and 3 indefinitely. To avoid this, we must first check to see whether the descendant points to itself. If so, we print the node's value and continue right.

Putting it all together, the code should look as follows:

```
def in_order_traversal(root):
    curr = root

    while curr:
        if not curr.left:
            print curr.data
            curr = curr.right

        else:
            # Find the rightmost descendant of the left child.
            desc = curr.left
            while desc.right and curr != desc.right:
                desc = desc.right

            # Make it point to the current node.
            if not desc.right:
                desc.right = curr
```

```
        curr = curr.left

    # If the rightmost descendant already points to the current node,
    # revert the changes we made to the tree and print the node's value.
    else:
        desc.right = None
        print curr.data
        curr = curr.right
```
