Master algorithms together on Binary Search! Create a room, invite your friends, and race to finish the problems.

Daily Coding Problem #48

Problem

This problem was asked by Google.

Given pre-order and in-order traversals of a binary tree, write a function to reconstruct the tree.

For example, given the following preorder traversal:

[a, b, d, e, c, f, g]

And the following inorder traversal:

[d, b, e, a, f, c, g]

You should return the following tree:

a

```
   / \
  b   c
 / \ / \
d  e f  g
```

Solution

Recall the definitions of preorder and inorder traversals:

For preorder:

```
- Evaluate root node
- Evaluate left node recursively
- Evaluate right node recursively
```

For inorder:

```
- Evaluate left node recursively
- Evaluate root node
- Evaluate right node recursively
```

It's helpful to go over an example. Consider the following tree:

```
    a
   / \
  b   c
 / \ / \
d  e f  g
```

The preorder traversal for this tree would be [a, b, d, e, c, f, g].

The inorder traversal for this tree would be [d, b, e, a, f, c, g].

Notice that because we always evaluate the root node first in a preorder traversal, the first element in the preorder traversal will always be the root. The second element is then either the root of the left node if there is one, or the root of the right node. But how do we know?

We can look at the inorder traversal.

Because we look at the left node first in an inorder traversal, all the elements up until the root will be part of the left subtree. All elements after the root will be the right subtree.

```
Preorder:
[a, b, d, e, c, f, g]
| r | left  | right |


Inorder:
[d, b, e, a, f, c, g]
| left  | r | right |


(r = root)
```

This gives us an idea for how to solve the problem:

- Find the root by looking at the first element in the preorder traversal
- Find out how many elements are in the left subtree and right subtree by searching for the index of the root in the inorder traversal
- Recursively reconstruct the left subtree and right subtree

The code for this problem would look like this:

```
def reconstruct(preorder, inorder):
    if not preorder and not inorder:
        return None
    if len(preorder) == len(inorder) == 1:
        return preorder[0]

    root = preorder[0]
    root_i = inorder.index(root)
    root.left = reconstruct(preorder[1:1 + root_i], inorder[0:root_i])
    root.right = reconstruct(preorder[1 + root_i:], inorder[root_i + 1:])
    return root
```

© Daily Coding Problem 2019 Privacy Policy Terms of Service Press