csgator Follow
Dec 27, 2017 · 2 min read

Leetcode Pattern 0 | Iterative traversals on Trees

The key to solve algorithm problems posed in technical interviews or elsewhere is to quickly identify the underlying patterns. This is my attempt to decompose leetcode problems and group them into minimal sets. I figured out this would be a useful insight for future people in tech (long live whiteboards !) to actually write a series of short blog posts.

Here I present one such powerful pattern related to trees and present 3 different problems that can be solved using this pattern. Whenever you solve a tree problem using recursion, always follow it up writing an iterative solution and vice-versa.

Iterative solutions help us really understand what is going on while recursion is concise and beautiful, we need to be able to write both in order to become good programmers.

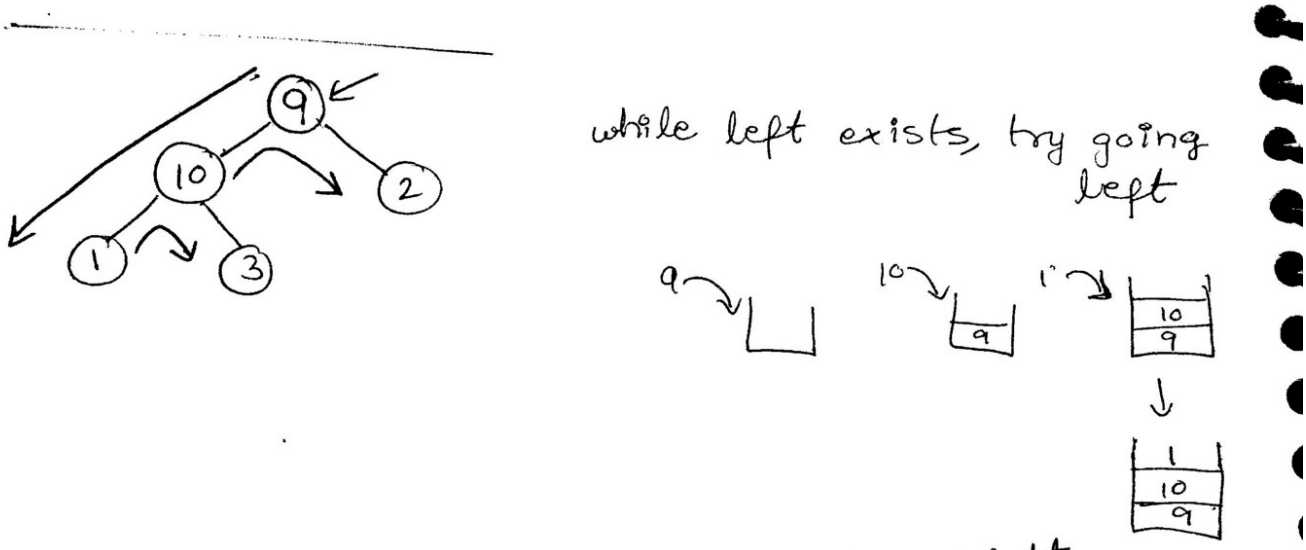
An iterative in-order traversal using stack to solve multiple tree problems:

Inorder Traversal of Binary Tree

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> inorder;
        stack<TreeNode*> nodes;

        if(root == NULL) return inorder;

        while(root != NULL || !nodes.empty()){
            //push left children if available
            while(root != NULL){
```

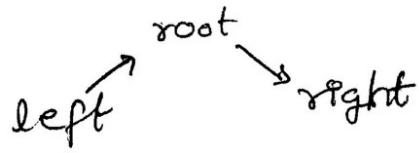
```
                nodes.push(root);
                root = root->left;
            }

            //retrieve top node and store its right child if exists
            root = nodes.top();
            nodes.pop();

            inorder.push_back(root->val);
            root = root->right;
        }

        return inorder;
    }
};
```

inorderTraversal.cpp hosted with ❤ by GitHub view raw



If no more left available, try going right by picking top most in stack

Remember Inorder



Now visualize how the while loop does this!! (It's amazing :))

Validate if Tree is a BST

For a binary tree to be a BST, the inorder has to be in sorted (ascending) order.

```
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        if(root == NULL) return true;

        stack<TreeNode*> s;
        TreeNode* prev = NULL;

        while(root != NULL || !s.empty()){
            while(root != NULL){
                s.push(root);
                root = root->left;
            }

            root = s.top(); s.pop();
            if(prev != NULL && prev->val >= root->val) return false;
            prev = root;
            root = root->right;
        }

        return true;
    }
};
```

isValidBST.cpp hosted with ❤ by GitHub view raw

Kth Smallest element in BST

Inorder traversal of a BST gives us a sorted order of the items in it. So a simple inorder breaking off at the kth item would give us our answer !

```
class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        stack<TreeNode*> s;

        while(root != NULL || !s.empty()){
            while(root != NULL){
```

```
                s.push(root);
                root = root->left;
            }

            root = s.top(); s.pop();

            if(--k == 0) return root->val;

            root = root->right;
        }

        return -1;
    }
};
```

KthSmallest.cpp hosted with ❤ by GitHub view raw

Try doing this for preorder and postorder too and see how many new problems you would have unlocked ! (Share your solutions in the comments)

I am also going to translate the solutions into Java and Python as I progress, feel free to contribute back by sharing your own patterns or translating code into your favorite language of choice.

Also feel free to connect with me on LI : https://www.linkedin.com/in/sourabh-reddy/

Programming Leetcode Algorithms Datastructures Whiteboarding