Binary Tree Postorder Traversal

Given a binary tree, return the *postorder* traversal of its nodes' values.

For example:
Given binary tree `{1,#,2,3}`,

```
   1
    \
     2
    /
   3
```

return `[3,2,1]`.

Note: Recursive solution is trivial, could you do it iteratively?

Solution 1

pre-order traversal is **root-left-right**, and post order is **left-right-root**. modify the code for pre-order to make it root-right-left, and then **reverse** the output so that we can get left-right-root .

1. Create an empty stack, Push root node to the stack.
2. Do following while stack is not empty.

   2.1. pop an item from the stack and print it.

   2.2. push the left child of popped item to stack.

   2.3. push the right child of popped item to stack.

3. reverse the ouput.

```
class Solution {
public:
    vector<int> postorderTraversal(TreeNode *root) {
        stack<TreeNode*> nodeStack;
        vector<int> result;
        //base case
        if(root==NULL)
        return result;
        nodeStack.push(root);
    while(!nodeStack.empty())
    {
        TreeNode* node= nodeStack.top();
        result.push_back(node->val);
        nodeStack.pop();
        if(node->left)
        nodeStack.push(node->left);
        if(node->right)
        nodeStack.push(node->right);
    }
     reverse(result.begin(),result.end());
     return result;

}
```

};

written by Deepalaxmi original link here

Solution 2

i have saw lots of post in this discussion, but most of them are not concise, just share mine for your reference, writing a concise code is very important

```
vector<int> postorderTraversal(TreeNode *root) {
    vector<int> v;
    if (!root) return v;

    stack<TreeNode *> s;
    s.push(root);

    TreeNode *p = NULL;
    while(!s.empty()) {
        p = s.top();
        s.pop();
        v.insert(v.begin(), p->val);
        if (p->left) s.push(p->left);
        if (p->right) s.push(p->right);
    }

    return v;
}
```

written by shichaotan original link here

Solution 3

Here I summarize the iterative implementation for preorder, inorder, and postorder traverse.

Pre Order Traverse

```java
public List<Integer> preorderTraversal(TreeNode root) {
    List<Integer> result = new ArrayList<>();
    Deque<TreeNode> stack = new ArrayDeque<>();
    TreeNode p = root;
    while(!stack.isEmpty() || p != null) {
        if(p != null) {
            stack.push(p);
            result.add(p.val);  // Add before going to children
            p = p.left;
        } else {
            TreeNode node = stack.pop();
            p = node.right;
        }
    }
    return result;
}
```

In Order Traverse

```java
public List<Integer> inorderTraversal(TreeNode root) {
    List<Integer> result = new ArrayList<>();
    Deque<TreeNode> stack = new ArrayDeque<>();
    TreeNode p = root;
    while(!stack.isEmpty() || p != null) {
        if(p != null) {
            stack.push(p);
            p = p.left;
        } else {
            TreeNode node = stack.pop();
            result.add(node.val);  // Add after all left children
            p = node.right;
        }
    }
    return result;
}
```

Post Order Traverse

```
public List<Integer> postorderTraversal(TreeNode root) {
    LinkedList<Integer> result = new LinkedList<>();
    Deque<TreeNode> stack = new ArrayDeque<>();
    TreeNode p = root;
    while(!stack.isEmpty() || p != null) {
        if(p != null) {
            stack.push(p);
            result.addFirst(p.val);  // Reverse the process of preorder
            p = p.right;             // Reverse the process of preorder
        } else {
            TreeNode node = stack.pop();
            p = node.left;           // Reverse the process of preorder
        }
    }
    return result;
}
```

written by yavinci original link here

From Leetcoder.