Missakshi/Sakshi_CSIT | (2) Tree - LeetCode | (2) N-ary Tree Postord | (2) N-ary Tree Preorde | (2) Binary Tree Inorder | (2) Binary Tree Postor | Loading...

leetcode.com/problems/n-ary-tree-postorder-traversal/

LeetCode Explore Problems Interview (New) Contest Discuss Store ☆ Premium

Description | Solution | Discuss | Submissions

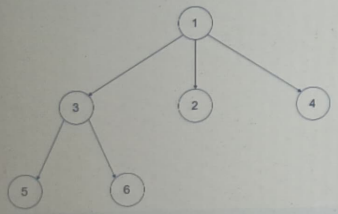
590. N-ary Tree Postorder Traversal

Easy 1727 86 Add to List Share

Given the `root` of an n-ary tree, return *the postorder traversal of its nodes' values.*

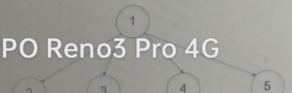
Nary-Tree input serialization is represented in their level order traversal. Each group of children is separated by the null value (See examples)

Example 1:




```
Input: root = [1,null,3,2,4,null,5,6]
Output: [5,6,3,2,4,1]
```

Example 2:



C++ | Autocomplete

```cpp
/*
// Definition for a Node.
class Node {
public:
    int val;
    vector<Node*> children;

    Node() {}

    Node(int _val) {
        val = _val;
    }

    Node(int _val, vector<Node*> _children) {
        val = _val;
        children = _children;
    }
};
*/

class Solution {
public:
    vector<int> postorder(Node* root) {
        vector<int> ans;
        traverse(root,ans);
        return ans;
    }
    void traverse(Node* node, vector<int>& ans) {
         if(node==NULL)  return;
        for(auto x:node->children){
            traverse(x,ans);
        }
        ans.push_back(node->val);
    }
};
```

Your previous code was restored from your local storage. Reset to default

Run Code | Submit

OPPO Reno3 Pro 4G

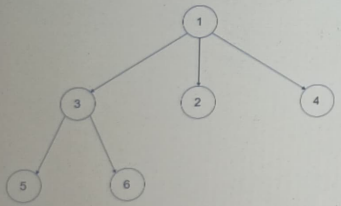
589. N-ary Tree Preorder Traversal

Easy 2076 100 Add to List Share

Given the `root` of an n-ary tree, return *the preorder traversal of its nodes' values.*

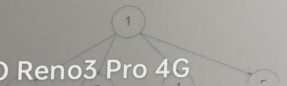
Nary-Tree input serialization is represented in their level order traversal. Each group of children is separated by the null value (See examples)

Example 1:




```
Input: root = [1,null,3,2,4,null,5,6]
Output: [1,3,5,6,2,4]
```

Example 2:



C++ Autocomplete

```cpp
/*
// Definition for a Node.
class Node {
public:
    int val;
    vector<Node*> children;

    Node() {}

    Node(int _val) {
        val = _val;
    }

    Node(int _val, vector<Node*> _children) {
        val = _val;
        children = _children;
    }
};
*/

class Solution {
public:
    vector<int> preorder(Node* root) {
        vector<int> ans;
        if (root) pre(root, &ans);
        return ans;
    }
    void pre(Node* node, vector<int>* ans) {
        ans->push_back(node->val);
        for (Node* child : node->children)
            pre(child, ans);
    }
};
```

Your previous code was restored from your local storage. Reset to default

Run Code Submit

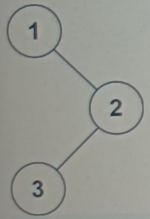
Description | Solution | Discuss (999+) | Submissions

94. Binary Tree Inorder Traversal

Easy | 8606 | 402 | Add to List | Share

Given the `root` of a binary tree, return *the inorder traversal of its nodes' values.*

Example 1:




```
Input: root = [1,null,2,3]
Output: [1,3,2]
```

Example 2:

```
Input: root = []
Output: []
```

Example 3:

```
Input: root = [1]
```

C++ | Autocomplete

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> nodes;
        inorder(root, nodes);
        return nodes;
    }
    private:
    void inorder(TreeNode* root, vector<int>& nodes) {
        if (!root) return;
        inorder(root -> left, nodes);
        nodes.push_back(root -> val);
        inorder(root -> right, nodes);
    }
};
```

Your previous code was restored from your local storage. Reset to default

Console | Contribute

Run Code | Submit

Description | Solution | Discuss (999+) | Submissions

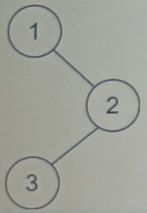
C++ | Autocomplete

145. Binary Tree Postorder Traversal

Easy 4676 148 Add to List Share

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

Example 1:




```
Input: root = [1,null,2,3]
Output: [3,2,1]
```

Example 2:

```
Input: root = []
Output: []
```

Example 3:

```
Input: root = [1]
```

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    vector<int> postorderTraversal(TreeNode* root) {
        vector<int> nodes;
      postorder(root, nodes);
      return nodes;
    }
    private:
    void postorder(TreeNode* root, vector<int>& nodes) {
        if (!root) return;
        postorder(root -> left, nodes);
        postorder(root -> right, nodes);
         nodes.push_back(root -> val);
    }
};
```

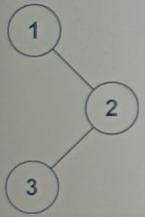
Your previous code was restored from your local storage. Reset to default

144. Binary Tree Preorder Traversal

Easy 4635 137 Add to List Share

Given the `root` of a binary tree, return *the preorder traversal of its nodes' values.*

Example 1:




```
Input: root = [1,null,2,3]
Output: [1,2,3]
```

Example 2:

```
Input: root = []
Output: []
```

Example 3:

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> nodes;
      preorder(root, nodes);
      return nodes;
    }
    private:
    void preorder(TreeNode* root, vector<int>& nodes) {
        if (!root) return;
        nodes.push_back(root -> val);
        preorder(root -> left, nodes);
        preorder(root -> right, nodes);
    }
};
```