

Assignment-3(AP lab)

Name:Akul

UID:22BCS10330

Section:605-B

94. Binary tree inorder traversal

Code:

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> res;
        inorder(root, res);
        return res;
    }

private:
    void inorder(TreeNode* node, vector<int>& res) {
        if (!node) {
            return;
        }
        inorder(node->left, res);
        res.push_back(node->val);
        inorder(node->right, res);
    }
};
```

Output:

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• Case 1

• Case 2

• Case 3

• Case 4

Input

root =
[1,null,2,3]

Output

[1,3,2]

Expected

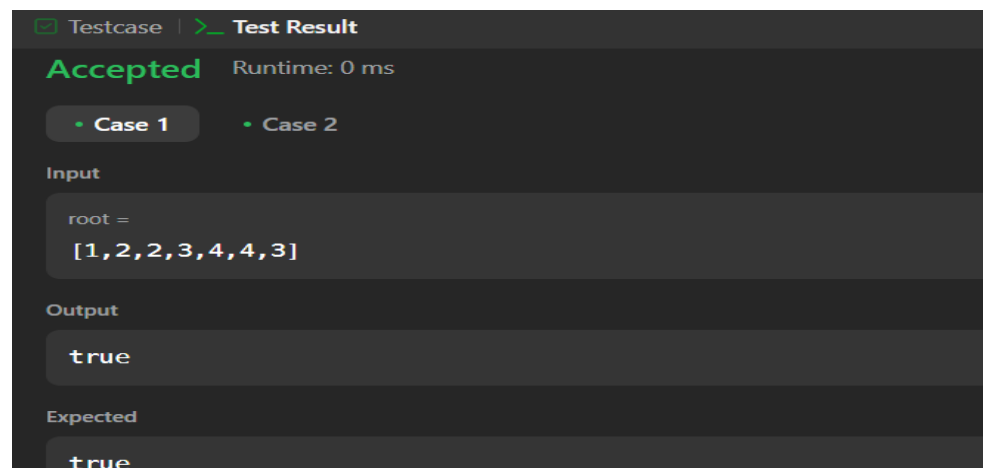
101.Symmetric tree

Code:

```
class Solution {
public:
    bool isMirror(TreeNode* left, TreeNode* right) {
        if (!left && !right) return true;
        if (!left || !right) return false;
        return (left->val == right->val) && isMirror(left->left, right->right) &&
isMirror(left->right, right->left);
    }

    bool isSymmetric(TreeNode* root) {
        if (!root) return true;
        return isMirror(root->left, root->right);
    }
};
```

Output:



The screenshot shows a test result interface with a dark theme. At the top, there are tabs for 'Testcase' and 'Test Result', with 'Test Result' being the active tab. Below the tabs, the status 'Accepted' is displayed in green, followed by 'Runtime: 0 ms'. There are two test cases listed: 'Case 1' and 'Case 2'. Under 'Case 1', the 'Input' is shown as 'root = [1,2,2,3,4,4,3]' and the 'Output' is 'true'. The 'Expected' result is also 'true'. The interface is clean and modern, with a focus on the test results.

104.Maximum depth of binary tree

Code:

```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(!root) return 0;
        int maxLeft = maxDepth(root->left);
        int maxRight = maxDepth(root->right);
        return max(maxLeft, maxRight)+1;
    }
};
```

Output:

The screenshot shows a test result interface with a dark theme. At the top, there are tabs for 'Testcase' and 'Test Result', with 'Test Result' being the active tab. Below the tabs, the status 'Accepted' is displayed in green, followed by 'Runtime: 0 ms'. There are two buttons labeled 'Case 1' and 'Case 2', with 'Case 1' being selected. The 'Input' section shows 'root =' followed by the array '[3,9,20,null,null,15,7]'. The 'Output' section shows the value '3'. The 'Expected' section also shows the value '3'.

98. Validate binary search tree

Code:

```
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return valid(root, LONG_MIN, LONG_MAX);
    }
private:
    bool valid(TreeNode* node, long minimum, long maximum) {
        if (!node) return true;

        if (!(node->val > minimum && node->val < maximum)) return false;

        return valid(node->left, minimum, node->val) && valid(node->right, node->val, maximum);
    }
};
```

Output:

The screenshot shows a test result interface with a dark theme. At the top, there are tabs for 'Testcase' and 'Test Result', with 'Test Result' being the active tab. Below the tabs, the status 'Accepted' is displayed in green, followed by 'Runtime: 0 ms'. There are two buttons labeled 'Case 1' and 'Case 2', with 'Case 1' being selected. The 'Input' section shows 'root =' followed by the array '[2,1,3]'. The 'Output' section shows the value 'true'. The 'Expected' section also shows the value 'true'.

230.k th smallest element in a BST

Code:

```
class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        const int leftCount = countNodes(root->left);
        if (leftCount == k - 1)
            return root->val;
        if (leftCount >= k)
            return kthSmallest(root->left, k);
        return kthSmallest(root->right, k - 1 - leftCount);
    }
private:
    int countNodes(TreeNode* root) {
        if (root == nullptr)
            return 0;
        return 1 + countNodes(root->left) + countNodes(root->right);
    }
};
```

Output:

☒ Testcase | >_ Test Result

Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

root =
[3,1,4,null,2]

k =
1

Output

1

102.Binary tree level order traversal

Code:

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>>ans;
        if(root==NULL)return ans;
        queue<TreeNode*>q;
        q.push(root);
        while(!q.empty()){
            int s=q.size();
            vector<int>v;
            for(int i=0;i<s;i++){
                TreeNode *node=q.front();
                q.pop();
                if(node->left!=NULL)q.push(node->left);
                if(node->right!=NULL)q.push(node->right);
                v.push_back(node->val);
            }
            ans.push_back(v);
        }
        return ans;
    }
};
```

Output:

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• Case 1

• Case 2

• Case 3

Input

root =
[3,9,20,null,null,15,7]

Output

[[3],[9,20],[15,7]]

Expected

[[3],[9,20],[15,7]]

107.Binary tree level order traversal

Code:

```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int>> res;
        if(!root) return res;
        queue<TreeNode*> q;
        q.push(root);
        while(!q.empty()){
            int size = q.size();
            vector<int> current;
            for(int i = 0; i < size; i++){
                TreeNode* node = q.front();
                q.pop();
                current.push_back(node->val);
                if(node->left) q.push(node->left);
                if(node->right) q.push(node->right);
            }
            res.push_back(current);
        }
        reverse(res.begin(), res.end());
        return res;
    }
};
```

Output:

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• Case 1

• Case 2

• Case 3

Input

root =
[3,9,20,null,null,15,7]

Output

[[15,7],[9,20],[3]]

Expected

[[15,7],[9,20],[3]]

103.Binary tree zigzag level order traversal

Code:

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> res;
        if(root==NULL){
            return res;
        }
        queue<TreeNode*>q;
        q.push(root);
        bool L2R=true;
        while(!q.empty()){
            int size=q.size();
            vector<int> arr(size);
            for(int i=0;i<size;i++){
                TreeNode* node=q.front();
                q.pop();
                int ind=(L2R)?i:(size-i-1);
                arr[ind]=node->val;
                if(node->left){
                    q.push(node->left);
                }
                if(node->right){
                    q.push(node->right);
                }
            }
            L2R=!L2R;
            res.push_back(arr);
        }
        return res;
    }
};
```

Output:

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input

root =
[3,9,20,null,null,15,7]

Output

[[3], [20,9], [15,7]]

Expected

[[3], [20,9], [15,7]]

199.Binary tree right side view

Code:

```
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int>ans;
        if(root==NULL) return ans;
        queue<TreeNode*>q;
        q.push(root);

        while(!q.empty()){
            int size=q.size();
            vector<int>level;
            for(int i=0;i<size;i++){
                TreeNode* node= q.front();
                q.pop();
                level.push_back(node->val);
                if(node->left) q.push(node->left);
                if(node->right) q.push(node->right);
            }
            ans.push_back(level[size-1]);
        }
        return ans;
    }
};
```

Output:

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• Case 1

• Case 2

• Case 3

• Case 4

Input

root =
[1,2,3,null,5,null,4]

Output

[1,3,4]

Expected

[1,3,4]

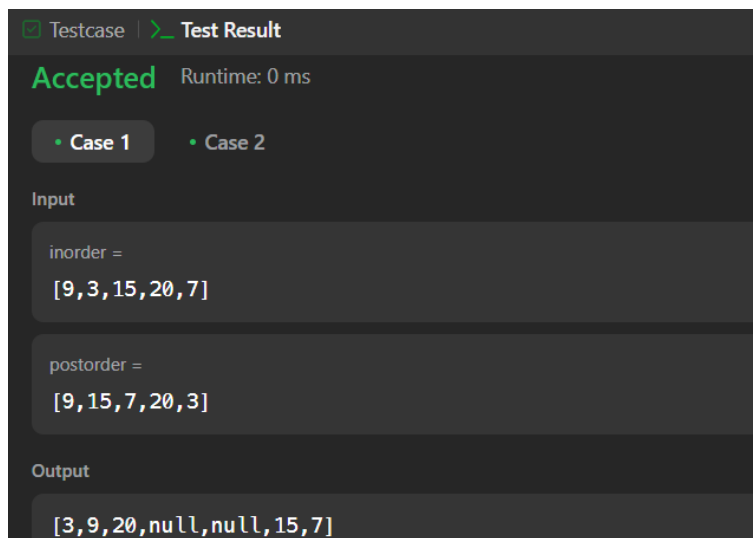
106. Construct binary tree from inorder and postorder traversal

Code:

```
class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        unordered_map<int, int> index;
        for (int i = 0; i < inorder.size(); i++) {
            index[inorder[i]] = i;
        }
        return buildTreeHelper(inorder, postorder, 0, inorder.size() - 1, 0,
            postorder.size() - 1, index);
    }

    TreeNode* buildTreeHelper(vector<int>& inorder, vector<int>& postorder, int
inorderStart, int inorderEnd, int postorderStart, int postorderEnd,
unordered_map<int, int>& index) {
        if (inorderStart > inorderEnd || postorderStart > postorderEnd) {
            return nullptr;
        }
        int rootVal = postorder[postorderEnd];
        TreeNode* root = new TreeNode(rootVal);
        int inorderRootIndex = index[rootVal];
        int leftSubtreeSize = inorderRootIndex - inorderStart;
        root->left = buildTreeHelper(inorder, postorder, inorderStart,
inorderRootIndex - 1, postorderStart, postorderStart + leftSubtreeSize - 1,
index);
        root->right = buildTreeHelper(inorder, postorder, inorderRootIndex + 1,
inorderEnd, postorderStart + leftSubtreeSize, postorderEnd - 1, index);
        return root;
    }
};
```

Output:



The screenshot shows a test result interface with a dark theme. At the top, there are tabs for 'Testcase' and 'Test Result', with 'Test Result' being the active tab. Below the tabs, the word 'Accepted' is displayed in green, followed by 'Runtime: 0 ms'. There are two buttons labeled 'Case 1' and 'Case 2', with 'Case 1' being the selected one. Under the 'Input' section, there are two text boxes: the first is labeled 'inorder =' and contains the array '[9,3,15,20,7]'; the second is labeled 'postorder =' and contains the array '[9,15,7,20,3]'. Under the 'Output' section, there is a text box containing the array '[3,9,20,null,null,15,7]'. The interface uses a dark gray background with light gray text for labels and green text for the 'Accepted' status.

513.Find bottom left tree value

Code:

```
class Solution {
public:
    int findBottomLeftValue(TreeNode* root) {
        queue <TreeNode*> q;
        q.push(root);
        TreeNode* cur = NULL;

        while(!q.empty()){
            cur = q.front();
            q.pop();

            if(cur->right) q.push(cur->right);
            if(cur->left) q.push(cur->left);
        }
        return cur->val;
    }
};
```

Output:

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

- Case 1
- Case 2

Input

root =
[2,1,3]

Output

1

Expected

1

124. Binary tree maximum path sum

Code:

```
class Solution {
public:
    int maxPathSum(TreeNode* root) {
        int ans = INT_MIN;
        maxPathSumDownFrom(root, ans);
        return ans;
    }

private:
    int maxPathSumDownFrom(TreeNode* root, int& ans) {
        if (root == nullptr)
            return 0;
        const int l = max(0, maxPathSumDownFrom(root->left, ans));
        const int r = max(0, maxPathSumDownFrom(root->right, ans));
        ans = max(ans, root->val + l + r);
        return root->val + max(l, r);
    }
};
```

Output:

☒ Testcase | **Test Result**

Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

root =
[1,2,3]

Output

6

Expected

6

987.Vertical order traversal of a binary tree

Code:

```
class Solution {
public:
    vector<vector<int>> verticalTraversal(TreeNode* root)
    {
        map<int, multiset<pair<int, int>>> mp; // [x][y, val]
        traverse(root, 0, 0, mp);
        vector<vector<int>> res;
        for(auto& [x, st] : mp)
        {
            res.push_back({});
            for(auto& [y, val] : st) res.back().push_back(val);
        }
        return res;
    }

protected:
    void traverse(TreeNode* node, int x, int y, map<int, multiset<pair<int,
int>>>& mp)
    {
        if(!node) return;
        mp[x].insert({y, node->val});
        traverse(node->left, x-1, y+1, mp);
        traverse(node->right, x+1, y+1, mp);
    }
};
```

Output:

☒ Testcase | [Test Result](#)

Accepted Runtime: 2 ms

• Case 1

• Case 2

• Case 3

Input

root =
[3,9,20,null,null,15,7]

Output

[[9], [3,15], [20], [7]]

Expected

[[9], [3,15], [20], [7]]