

ASSIGNMENT-3(AP)

Name: Saharsh Kumar

UID: 22BCS14059

Section: 22BCS_FL_IOT-604

Group: A

1) binary-tree-inorder-traversal

```
void inorder(TreeNode* root,vector<int>&ans){  
    if(root==NULL){  
        return;  
    }  
    inorder(root->left,ans);  
    ans.push_back(root->val);  
    inorder(root->right,ans);  
}  
vector<int> inorderTraversal(TreeNode* root) {  
    vector<int>ans;  
    inorder(root,ans);  
  
    return ans;  
}
```

94. Binary Tree Inorder Traversal Solved

Easy Topics Companies

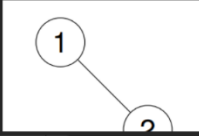
Given the `root` of a binary tree, return the *inorder traversal* of its nodes' values.

Example 1:

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

Output: `[1,3,2]`

Explanation:



```
class Solution {  
public:  
    void inorder(TreeNode* root, vector<int>&ans){  
        if(root==NULL){  
            return;  
        }  
        inorder(root->left,ans);  
        ans.push_back(root->val);  
        inorder(root->right,ans);  
    }  
    vector<int> inorderTraversal(TreeNode* root) {  
        vector<int>ans;  
        inorder(root,ans);  
  
        return ans;  
    }  
};
```

Testcase Test Result

2) symmetric-tree

```
bool mirrorcheck(TreeNode* p, TreeNode* q) {
```

```

    if (p == NULL && q == NULL) {
        return true;
    }
    if (p == NULL || q == NULL) {
        return false;
    }

    if (p->val != q->val) {
        return false;
    }

    return mirrorcheck(p->left, q->right) && mirrorcheck(p->right, q->left);
}

bool isSymmetric(TreeNode* root) {
    if (mirrorcheck(root->left, root->right) == true) {
        return true;
    } else {
        return false;
    }
}
}

```

Description

Editorial

Solutions

Submissions

101. Symmetric Tree

Solved

Easy

Given the *root* of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

Example 1:

15.8K 191 113 Online

Code

```

12  /*
13  class Solution {
14  public:
15      bool mirrorcheck(TreeNode* p, TreeNode* q) {
16          if (p == NULL && q == NULL) {
17              return true;
18          }
19          if (p == NULL || q == NULL) {
20              return false;
21          }
22
23          if (p->val != q->val) {
24              return false;
25          }

```

Run

Submit

Testcase

Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

3) maximum-depth-of-binary-tree

```

int maxDepth(TreeNode* root) {
    if (root == NULL) {
        return 0;
    }
}

```

```

}

int left=maxDepth(root->left);

int right=maxDepth(root->right);

int ans=max(left,right)+1;


return ans;

}

```

Description
Editorial
Solutions
Submissions

104. Maximum Depth of Binary Tree

Solved

Easy Topics Companies

Given the `root` of a binary tree, return its *maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:

```

graph TD
    3((3)) --> 9((9))
    3 --> 20((20))
    20 --> 21(( ))

```

13.3K 155 179 Online

Code

```

C++
1  *   TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(r
2  * };
3  */
4  class Solution {
5  public:
6      int maxDepth(TreeNode* root) {
7          if(root==NULL){
8              return 0;
9          }
10         int left=maxDepth(root->left);
11         int right=maxDepth(root->right);
12         int ans=max(left,right)+1;
13
14         return ans;
15     }
16 };

```

Ln 1, Col 1 Saved Run Submit

Testcase
Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

4) validate-binary-search-tree

```

bool solve(TreeNode* root,long long int lb,long long int ub){

    if(root==NULL){

        return true;

    }


    if(root->val>lb && root->val<ub){

        bool leftans=solve(root->left,lb,root->val);

        bool rightans=solve(root->right,root->val,ub);

        return leftans && rightans;

    }

    else{

```

```

        return false;
    }
}

bool isValidBST(TreeNode* root) {
    long long int lowerbound=-4294967296;
    long long int upperbound=4294967296;
    bool ans= solve(root,lowerbound,upperbound);
    return ans;
}

```

The screenshot shows the LeetCode interface for problem 98, "Validate Binary Search Tree". The problem description states: "Given the root of a binary tree, determine if it is a valid binary search tree (BST). A valid BST is defined as follows: The left subtree of a node contains only nodes with keys less than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. Both the left and right subtrees must also be binary search trees." An example shows a tree with root 2 and no children. The solution is written in C++ and uses a recursive function 'solve' that checks if the root's value is within a given range [lb, ub]. The test results show the solution is "Accepted" with a runtime of 0 ms.

98. Validate Binary Search Tree Solved

Medium Topics Companies

Given the `root` of a binary tree, determine if it is a valid binary search tree (BST).

A **valid BST** is defined as follows:

- The left **subtree** of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:

Input: `2`

Output: `true`

17.4K 220 137 Online

Code

```

14 bool solve(TreeNode* root, long long int lb, long long int ub){
15     if(root==NULL){
16         return true;
17     }
18
19     if(root->val>lb && root->val<ub){
20         bool leftans=solve(root->left, lb, root->val);
21         bool rightans=solve(root->right, root->val, ub);
22         return leftans && rightans;
23     }
24     else{
25         return false;
26     }
27 }

```

Ln 1, Col 1 Saved Run Submit

Testcase **Test Result**

Accepted Runtime: 0 ms

Case 1 Case 2

5) kth-smallest-element-in-a-bst

```

int kthSmallest(TreeNode* root, int &k) {
    if(root==NULL){
        return -1;
    }

    int leftans=kthSmallest(root->left,k);
    if(leftans!=-1){
        return leftans;
    }
    k--;
}

```

```

if(k==0){
    return root->val;
}
int rightans=kthSmallest(root->right,k);
return rightans;
}

```

230. Kth Smallest Element in a BST Solved ✓

Medium Topics Companies Hint

Given the *root* of a binary search tree, and an integer *k*, return the *kth* smallest value (**1-indexed**) of all the values of the nodes in the tree.

Example 1:

```

class Solution {
public:
    int kthSmallest(TreeNode* root, int &k) {
        if(root==NULL){
            return -1;
        }

        int leftans=kthSmallest(root->left,k);
        if(leftans!=-1){
            return leftans;
        }
        k--;
        if(k==0){
            return root->val;
        }
        int rightans=kthSmallest(root->right,k);
        return rightans;
    }
};

```

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

6) binary-tree-level-order-traversal

```

vector<vector<int>> levelOrder(TreeNode* root) {
    vector<vector<int>>ans;
    if(root==NULL){
        return ans;
    }
    vector<int>demo;
    queue<TreeNode*>q;
    q.push(root);
    q.push(NULL);
    while(!q.empty()){
        TreeNode* temp=q.front();
        q.pop();
        if(temp==NULL){

```

```

        ans.push_back(demo);

        demo.clear();

        if(!q.empty()){
            q.push(NULL);
        }
    }

    else{

        demo.push_back(temp->val);

        if(temp->left){
            q.push(temp->left);
        }

        if(temp->right){
            q.push(temp->right);
        }

    }

}

return ans;

}

```

Problem List

Description
Editorial
Solutions
Submissions

102. Binary Tree Level Order Traversal Solved

Medium Topics Companies Hint

Given the `root` of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

Example 1:

```

graph TD
    3((3)) --- 9((9))
    3 --- 20((20))
    20 --- 21(( ))
    20 --- 22(( ))

```

15.9K 116 168 Online

Code

```

class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        if(root==NULL){
            return ans;
        }
        vector<int> demo;
        queue<TreeNode*> q;
        q.push(root);
        q.push(NULL);
        while(!q.empty()){
            TreeNode* temp=q.front();
            q.pop();

```

Testcase
Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

7) binary-tree-zigzag-level-order-traversal

```
vector<vector<int>> zigzagLevelOrder(TreeNode* root) {  
    vector<vector<int>>ans;  
    if(root==NULL){  
        return ans;  
    }  
    queue<TreeNode*>q;  
    bool LtoR=true;  
    q.push(root);  
    while(!q.empty()){  
        int width=q.size();  
        vector<int>level(width);  
        for(int i=0;i<width;i++){  
            TreeNode* frontnode=q.front();  
            q.pop();  
            int index= LtoR? i: width-i-1;  
            level[index]=frontnode->val;  
            if(frontnode->left){  
                q.push(frontnode->left);  
            }  
            if(frontnode->right){  
                q.push(frontnode->right);  
            }  
        }  
        LtoR=!LtoR;  
        ans.push_back(level);  
    }  
    return ans;  
}
```

Problem List

Description
Editorial
Solutions
Submissions

103. Binary Tree Zigzag Level Order Traversal

Medium Topics Companies

Given the `root` of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).

Example 1:

```

graph TD
    3((3)) --> 9((9))
    3 --> 20((20))

```

C++
Auto

```

12 class Solution {
13 public:
14     vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
15         vector<vector<int>> ans;
16         if(root==NULL){
17             return ans;
18         }
19         queue<TreeNode*> q;
20         bool LtoR=true;
21         q.push(root);
22         while(!q.empty()){
23             int width=q.size();
24             vector<int> level(width);
25             for(int i=0;i<width;i++){

```

Ln 1, Col 1 Saved
Run
Submit

Testcase
Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

8) binary-tree-right-side-view

```

void RV(TreeNode* root,int level,vector<int>&ans){

    if(root==NULL){

        return;

    }

    if(level==ans.size()){

        ans.push_back(root->val);

    }

    RV(root->right,level+1,ans);

    RV(root->left,level+1,ans);

}

vector<int> rightSideView(TreeNode* root) {

    vector<int> ans;

    int level=0;

    RV(root,level,ans);

    return ans;

}

```


199. Binary Tree Right Side View Solved

Medium Topics Companies

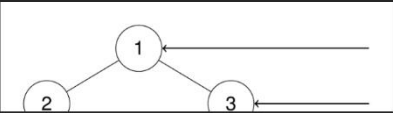
Given the `root` of a binary tree, imagine yourself standing on the **right side** of it, return the values of the nodes you can see ordered from top to bottom.

Example 1:

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

Output: `[1,3,4]`

Explanation:



12.5K 212 125 Online

```

C++
TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}

class Solution {
public:
    void RV(TreeNode* root, int level, vector<int>&ans){
        if(root==NULL){
            return;
        }
        if(level==ans.size()){
            ans.push_back(root->val);
        }
        RV(root->right, level+1, ans);
    }
};

```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3 Case 4

9) construct-binary-tree-from-inorder-and-postorder-traversal

```

int findposition(int element,int size,vector<int>&inorder){
    for(int i=0;i<size;i++){
        if(element==inorder[i]){
            return i;
        }
    }
    return -1;
}

```

```

TreeNode* BT(vector<int>&inorder,vector<int>&postorder,int size,int &postindex,int
startinorder,int endinorder){

```

```

    if(postindex<0 || startinorder>endinorder){
        return NULL;
    }

```

```

    int element=postorder[postindex--];

```

```

    TreeNode* root= new TreeNode(element);

```

```

    int position =findposition(element,size,inorder);

```

```

    root->right=BT(inorder,postorder,size,postindex,position+1,endinorder);

```

```
root->left=BT(inorder,postorder,size,postindex,startinorder,position-1);
```

```
return root;
```

```
}
```

```
TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
```

```
    int size=inorder.size();
```

```
    int postindex=size-1;
```

```
    int startinorder=0;
```

```
    int endinorder=size-1;
```

```
    TreeNode* root= BT(inorder,postorder,size,postindex,startinorder,endinorder);
```

```
    return root;
```

```
}
```

The screenshot shows a coding platform interface. On the left, the problem description for "106. Construct Binary Tree from Inorder and Postorder Traversal" is visible. It states: "Given two integer arrays `inorder` and `postorder` where `inorder` is the inorder traversal of a binary tree and `postorder` is the postorder traversal of the same tree, construct and return the binary tree." An example is provided with a diagram of a binary tree with root 3, left child 9, and right child 20.

On the right, the solution code is displayed in C++:

```
int findposition(int element,int size,vector<int>&inorder){
    for(int i=0;i<size;i++){
        if(element==inorder[i]){
            return i;
        }
    }
    return -1;
}

TreeNode* BT(vector<int>&inorder,vector<int>&postorder,int size,int &postindex,int
&startinorder,int &endinorder){
    if(postindex<0 || startinorder>endinorder){
        return NULL;
    }
    int element=postorder[postindex--];
    TreeNode* root= new TreeNode(element);
```

The code is shown in a dark-themed editor with line numbers. Below the code, there are buttons for "Run" and "Submit". At the bottom, the test result is shown as "Accepted" with a runtime of 0 ms.

10) vertical-order-traversal-of-a-binary-tree

```
vector<vector<int>>> verticalTraversal(TreeNode* root) {
```

```
    vector<vector<int>>>ans;
```

```
    queue<pair<TreeNode*,pair<int,int>>>>q;
```

```
    q.push({root,{0,0}});
```

```

map<int,map<int,multiset<int>>>mp;
while(!q.empty()){
    auto temp=q.front();
    q.pop();
    TreeNode* node=temp.first;
    auto coordinate=temp.second;
    int row=coordinate.first;
    int col=coordinate.second;
    mp[col][row].insert(node->val);
    if(node->left){
        q.push({node->left,{row+1,col-1}});
    }
    if(node->right){
        q.push({node->right,{row+1,col+1}});
    }
}

for(auto i:mp){
    auto &map=i.second;
    vector<int>vline;
    for(auto j:map){
        auto &multiset=j.second;
        vline.insert(vline.end(),multiset.begin(),multiset.end());
    }
    ans.push_back(vline);
}
return ans;
}

```

Problem List

DescriptionEditorialSolutionsSubmissions

987. Vertical Order Traversal of a Binary Tree

Solved

HardTopicsCompanies

Given the `root` of a binary tree, calculate the **vertical order traversal** of the binary tree.

For each node at position `(row, col)`, its left and right children will be at positions `(row + 1, col - 1)` and `(row + 1, col + 1)` respectively. The root of the tree is at `(0, 0)`.

The **vertical order traversal** of a binary tree is a list of top-to-bottom orderings for each column index starting from the leftmost column and ending on the rightmost column. There may be multiple nodes in the same row and same column. In such a case, sort these nodes by their values.

Return the **vertical order traversal** of the binary tree.

Example 1:

8K146

88 Online

Code

C++Auto

```
6 *   TreeNode *right;
7 *   TreeNode() : val(0), left(nullptr), right(nullptr) {}
8 *   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9 *   TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(r
10 * };
11 //
12 class Solution {
13 public:
14     vector<vector<int>>> verticalTraversal(TreeNode* root) {
15         vector<vector<int>>>ans;
16         queue<pair<TreeNode*,pair<int,int>>>>q;
17         q.push({root,{0,0}});
18         map<int,map<int,multiset<int>>>>mp;
19         while(!q.empty()){
20             auto temp=q.front();
21             q.pop();
22             if(temp.first->left){
23                 q.push({temp.first->left,{temp.second.first-1,temp.second.second+1}});
24             }
25             if(temp.first->right){
26                 q.push({temp.first->right,{temp.second.first+1,temp.second.second+1}});
27             }
28             for(auto &it:mp[temp.second.first]){
29                 ans[temp.second.first].push_back(*it);
30             }
31             mp.clear();
32         }
33         return ans;
34     }
35 }
```

Ln 1, Col 1 Saved

RunSubmit

TestcaseTest Result

AcceptedRuntime: 0 ms

Case 1Case 2Case 3