Description | Accepted | Editorial | Solutions | Submissions

94. Binary Tree Inorder Traversal

Solved

Easy | Topics | Companies

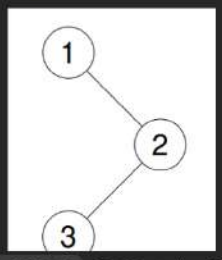
Given the `root` of a binary tree, return *the inorder traversal of its nodes' values.*

Example 1:

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

Output: `[1,3,2]`

Explanation:



13.9K | 189 | 125 Online

Code

C++ | Auto

```
 */
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int>ans;
        TreeNode*node=root;
        stack<TreeNode*>st;
        while(node!=NULL ||!st.empty() ){
if(node!=NULL){
    st.push(node);
    node=node->left;
}
else{
    node=st.top();
    st.pop();
    ans.push_back(node->val);
    node=node->right;
}
        }
        return ans;
    }
};
```

Saved | Ln 31, Col 6

Testcase | Test Result

Source

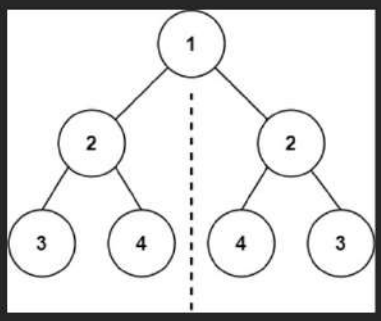
101. Symmetric Tree

Solved

Easy | Topics | Companies

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

Example 1:



Input: root = [1,2,2,3,4,4,3]
Output: true

15.8K | 191 | 98 Online

C++ | Auto

```cpp
 */
class Solution {
public:
bool check(TreeNode*l,TreeNode*r){
 if(l==NULL && r==NULL){
    return true;
 }
 if(l!=NULL && r!=NULL && l->val==r->val){
return check(l->left,r->right) && check(l->right,r->left);
 }
return false;
}
    bool isSymmetric(TreeNode* root) {
        if(root==NULL){
            return true;
        }
       bool ans= check(root->left,root->right);
       return ans;
    }
};
```

Saved | Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Description | Editorial | Solutions | Subm

104. Maximum Depth of Binary Tree

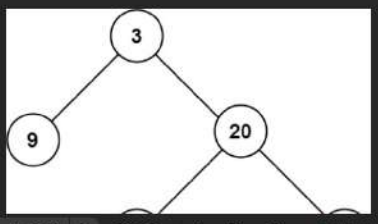
Solved

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:



13.3K | 155 | 171 Onli

Code

C++ | Auto

```cpp
class Solution {
public:

    int maxDepth(TreeNode* root) {
        if(root==NULL){
         return 0;
        }
        queue<TreeNode*>q;
        q.push(root);
        int count=0;
        while(!q.empty()){
count++;
int siz=q.size();

for(int i=0;i<siz;i++){
    TreeNode*temp=q.front();
q.pop();
if(temp->left!=NULL){
    q.push(temp->left);
}
if(temp->right!=NULL){
    q.push(temp->right);
}
}

        }
        return count;
    }
};
```

Saved | Ln 1, Co

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

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

Output

`3`

Expected

`3`

Contribute a testcase

Description | Editorial | Solutions | Submissions

98. Validate Binary Search Tree

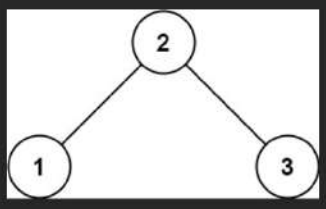
Medium | Topics | Companies

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:



17.4K | 220 | 175 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(right) {}
10  * };
11  */
12  class Solution {
13
14  bool isPossible(TreeNode* root, long long l, long long r){
15      if(root == nullptr)  return true;
16      if(root->val < r and root->val > l)
17          return isPossible(root->left, l, root->val) and
18                            isPossible(root->right, root->val, r);
19      else return false;
20  }
21
22  public:
23      bool isValidBST(TreeNode* root) {
24          long long int min = -1000000000000, max = 1000000000000;
25          return isPossible(root, min, max);
26      }
27  };
```

Saved | Ln 27, Col 3

Testcase | Test Result

Accepted Runtime: 0 ms

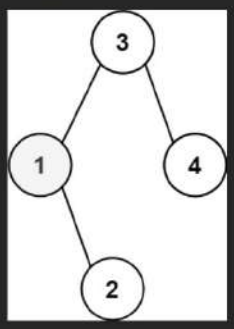
Description | Editorial | Solutions | Submissions

230. Kth Smallest Element in a BST

Medium | Topics | Companies | Hint

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

Example 1:



11.9K | 118 | 123 Online

Code

C++ | Auto

```cpp
class Solution {
public:
    int count = 0;

    int kthSmallest(TreeNode* root, int k) {
        TreeNode* result = helper(root, k);
        return result ? result->val : 0;
    }

    TreeNode* helper(TreeNode* root, int k) {
        if (root == nullptr) return nullptr;


        TreeNode* left = helper(root->left, k);
        if (left != nullptr) return left;

        count++;
        if (count == k) return root;


        return helper(root->right, k);
    }
};
```

Saved | Ln 12, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Description | Editorial | Solutions | Submissions

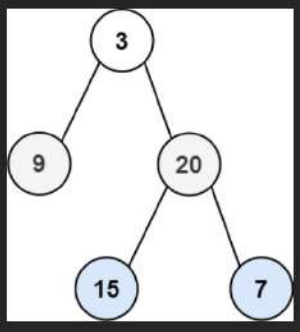
102. Binary Tree Level Order Traversal

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:



15.9K | 116 | 152 Online

Code

C++ | Auto

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

Saved | Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

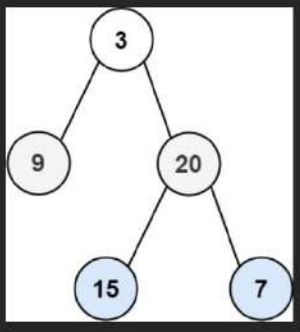
Description | Editorial | Solutions | Submissions

107. Binary Tree Level Order Traversal II

Medium | Topics | Companies

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

Example 1:



5K | 41 | 20 Online

Code

C++ | Auto

```cpp
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        if (!root) return {};
        vector<vector<int>> result;
        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);
            }
            result.push_back(level);
        }
        reverse(result.begin(), result.end());
        return result;
    }
};
```

Saved | Ln 35, Col 3

Testcase | Test Result

Accepted Runtime: 0 ms

Description | Editorial | Solutions | Submissions

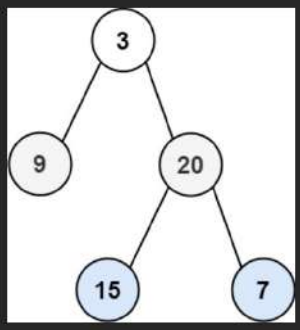
103. Binary Tree Zigzag Level Order Traversal

Solved

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:




```
Input: root = [3,9,20,null,null,15,7]
Output: [[3],[20,9],[15,7]]
```

Example 2:

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

Example 3:

11.2K | 135 | 72 Online

Code

C++ | Auto

```cpp
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        if(root==NULL){
            return{};
        }
        vector<vector<int>>ans;
        queue<TreeNode*>q;
        q.push(root);
        bool flag=true;
        //  TreeNode*node=root;
        while(!q.empty()){
int n=q.size();
vector<int>v;
for(int i=0;i<n;i++){
TreeNode*temp=q.front();
v.push_back(temp->val);
q.pop();
if(temp->left!=NULL){
    q.push(temp->left);
}
if(temp->right!=NULL){
    q.push(temp->right);
}
}
if(flag==true){
    ans.push_back(v);
    flag=false;
}
else{
reverse(v.begin(),v.end());
ans.push_back(v);
flag=true;
}
        }
        return ans;
```

Saved | Ln 1, Col 1

Testcase | Test Result

Accepted | Runtime: 0 ms

Description | Editorial | Solutions | Submissions

To exit full screen, press Esc

199. Binary Tree Right Side View

Solved

Medium | Topics | Companies

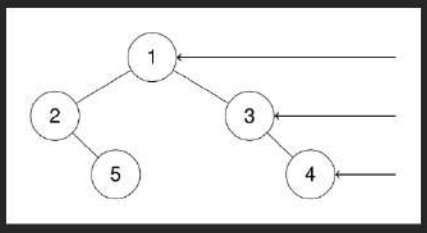
Given the `root` of a binary tree, imagine yourself standing on the **right side** of it, return *the values of the nodes you can see ordered from top to bottom.*

Example 1:

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

Output: `[1,3,4]`

Explanation:



Example 2:

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

Output: `[1,3,4,5]`

Explanation:

12.5K | 213 | 123 Online

Code

C++ | Auto

```cpp
public:
    vector<int> rightSideView(TreeNode* root) {
        if(root==NULL){
            return{};
        }
        vector<int>ans;
        queue<pair<int,TreeNode*>>q;
        map<int,int>mp;
        q.push({0,root});
        while(!q.empty()){
         auto it=q.front();
         q.pop();
         int level=it.first;
         TreeNode*temp=it.second;
         if(mp.find(level)==mp.end()){
            mp[level]=temp->val;
         }
         if(temp->right!=NULL){
            q.push({level+1,temp->right});
         }
         if(temp->left!=NULL){
            q.push({level+1,temp->left});
         }
        }
        for(auto it:mp){
         ans.push_back(it.second);
        }
        return ans;
    }
};
```

Saved | Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3 | Case 4

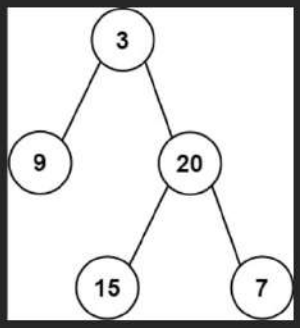
Description | Editorial | Solutions | Submissions

106. Construct Binary Tree from Inorder and Postorder Traversal

Medium | Topics | Companies

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*.

Example 1:




```
Input: inorder = [9,3,15,20,7], postorder = [9,15,7,20,3]
Output: [3,9,20,null,null,15,7]
```

Example 2:

```
Input: inorder = [-1], postorder = [-1]
Output: [-1]
```

8.3K | 78 | 55 Online

Code

C++ | Auto

```cpp
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        unordered_map<int, int> rec;
        for (int i = 0; i < inorder.size(); i++) {
            rec[inorder[i]] = i;
        }
        return helper(inorder, postorder, 0, inorder.size() - 1, 0, postorder.size() - 1, rec);
    }

    TreeNode* helper(vector<int>& inorder, vector<int>& postorder,
                     int inStart, int inEnd,
                     int postStart, int postEnd,
                     unordered_map<int, int>& rec) {
        if (inStart > inEnd || postStart > postEnd) return nullptr;

        int val = postorder[postEnd];
        TreeNode* root = new TreeNode(val);
        int idx = rec[val];
        int leftSubtreeSize = idx - inStart;

        root->left = helper(inorder, postorder,
                            inStart, idx - 1,
                            postStart, postStart + leftSubtreeSize - 1,
                            rec);

        root->right = helper(inorder, postorder,
                             idx + 1, inEnd,
                             postStart + leftSubtreeSize, postEnd - 1,
                             rec);

        return root;
    }
};
```

Saved | Ln 12, Col 1

Testcase | Test Result

Accepted Runtime: 3 ms

Problem List | Run | Submit | Premium

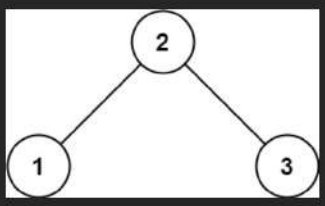
Description | Editorial | Solutions | Submissions

513. Find Bottom Left Tree Value

Medium | Topics | Companies

Given the `root` of a binary tree, return the leftmost value in the last row of the tree.

Example 1:




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

Example 2:

3.9K | 129 | 7 Online

Code

C++ | Auto

```cpp
int findBottomLeftValue(TreeNode* root) {
    queue<TreeNode*> q;
    q.push(root);
    int leftmost_value;

    while (!q.empty()) {
        TreeNode* node = q.front();
        q.pop();

        leftmost_value = node->val;

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

    return leftmost_value;
}
};
```

Saved | Ln 35, Col 3

Testcase | Test Result

Accepted Runtime: 0 ms

Problem List

Run Submit Premium

Description | Editorial | Solutions | Submissions

124. Binary Tree Maximum Path Sum

Solved

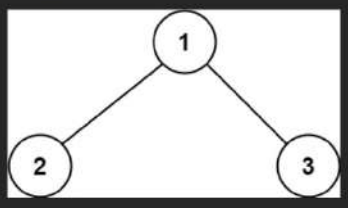
Hard Topics Companies

A **path** in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence **at most once**. Note that the path does not need to pass through the root.

The **path sum** of a path is the sum of the node's values in the path.

Given the `root` of a binary tree, return *the maximum* ***path sum*** *of any* ***non-empty*** *path*.

Example 1:



Input: root = [1,2,3]
Output: 6

17.3K 221 167 Online

Code

C++ Auto

```
*/
class Solution {
public:
int helper(TreeNode*root,int &maxi){
if(root==NULL){
    return 0;
}
int leftsum=max(0,helper(root->left,maxi));
int rightsum=max(0,helper(root->right,maxi));
maxi=max(maxi,leftsum+rightsum+root->val);
return root->val+max(leftsum,rightsum);
}
    int maxPathSum(TreeNode* root) {
        if(root==NULL){
            return 0 ;
        }
        int maxi=INT_MIN;
        helper(root,maxi);
return maxi;
    }
};
```

Saved Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Problem List

Run Submit

Premium

Description | Editorial | Solutions | Submissions

987. Vertical Order Traversal of a Binary Tree

Solved

Hard | Topics | Companies

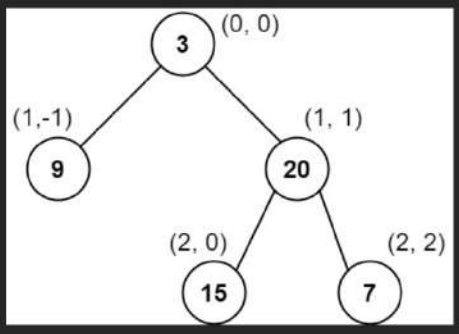
Given the `root` of a binary tree, calculate the **vertical order traversal** of the binary tree.

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:



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

8K 146

100 Online

Code

C++ Auto

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

    map<int, vector<pair<int, int>>> nodes;
    queue<pair<TreeNode*, pair<int, int>>> q;
    q.push({root, {0, 0}});
    while (!q.empty()) {
        auto p = q.front();
        q.pop();
        TreeNode* node = p.first;
        int x = p.second.first, y = p.second.second;
        nodes[x].emplace_back(y, node->val);

        if (node->left) {
            q.push({node->left, {x - 1, y + 1}});
        }
        if (node->right) {
            q.push({node->right, {x + 1, y + 1}});
        }
    }
    vector<vector<int>> result;
    for (auto& p : nodes) {
        sort(p.second.begin(), p.second.end());
        vector<int> col;
        for (auto& q : p.second) {
            col.push_back(q.second);
        }
        result.push_back(col);
    }
    return result;
}
};
```

Saved

Ln 9, Col 30

Testcase | Test Result

Accepted Runtime: 2 ms

Case 1 | Case 2 | Case 3