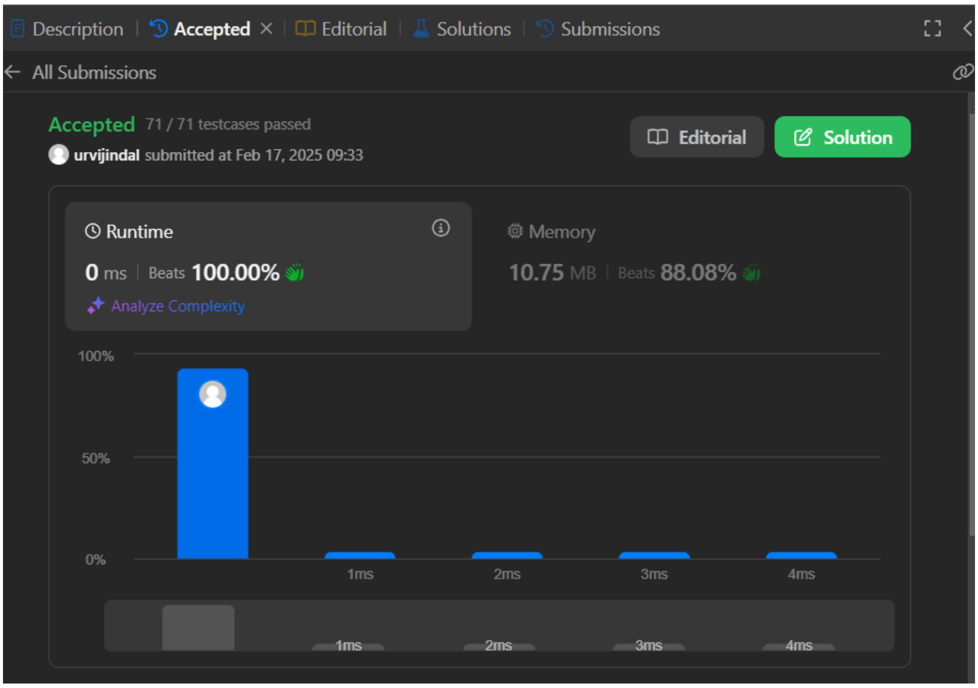
ASSIGNMENT-3

Name: Urvi Jindal **Section:** FL_IOT-603/A

UID: 22BCS14860

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;
    }
```



Accepted 71 / 71 testcases passed

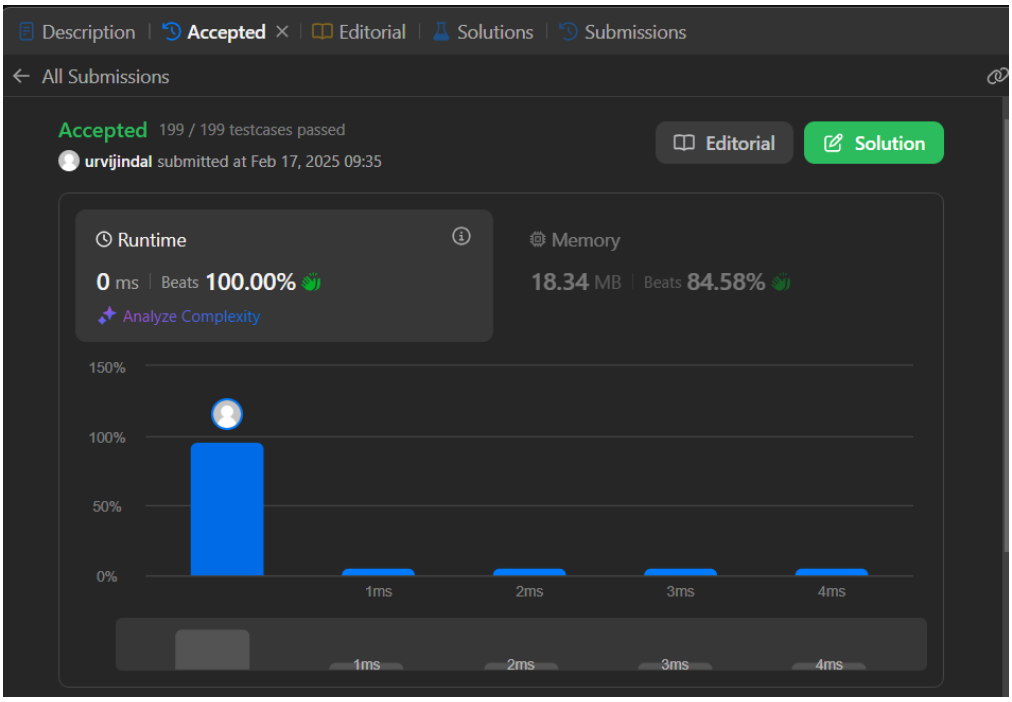
urvijindal submitted at Feb 17, 2025 09:33

0 ms | Beats 100.00%

10.75 MB | Beats 88.08%

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;
        }
    }
```



Accepted 199 / 199 testcases passed

urvijindal submitted at Feb 17, 2025 09:35

Runtime

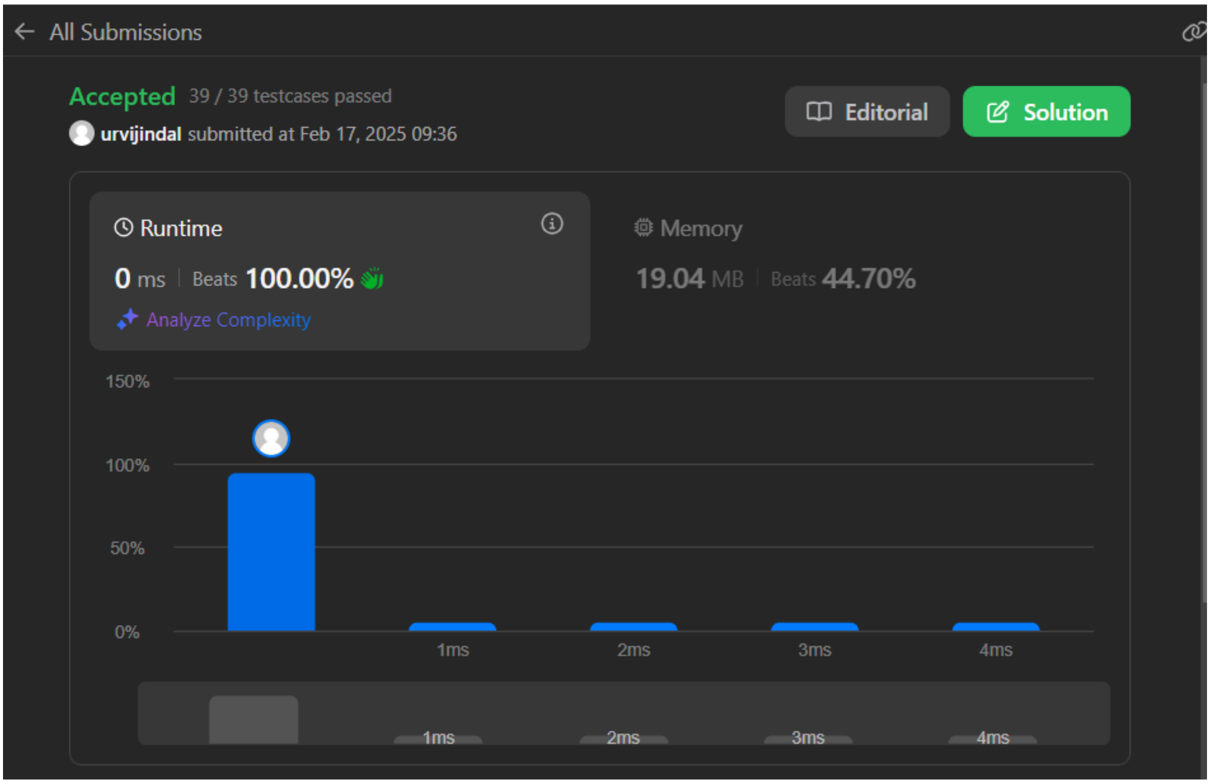
0 ms | Beats 100.00%

Memory

18.34 MB | Beats 84.58%

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;
    }
```



Accepted 39 / 39 testcases passed

urvijindal submitted at Feb 17, 2025 09:36

Runtime

0 ms | Beats 100.00%

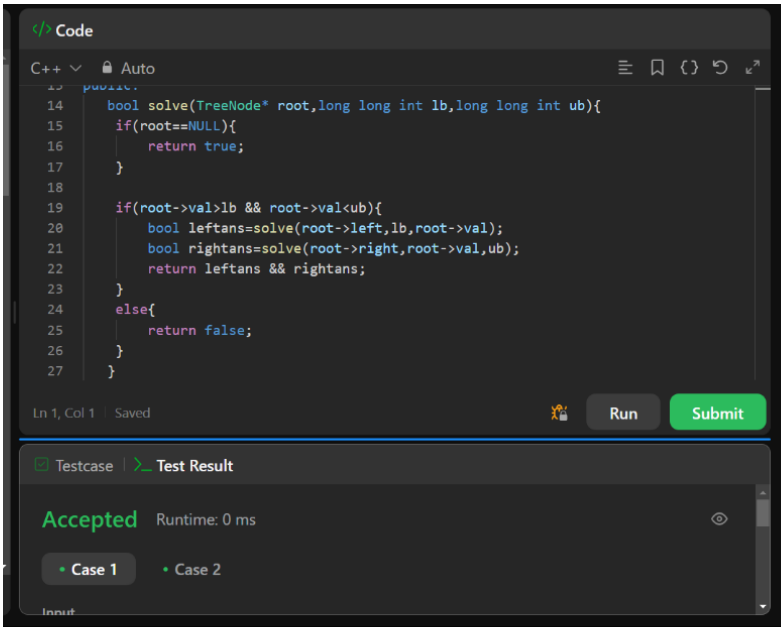
Memory

19.04 MB | Beats 44.70%

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;
 }
```

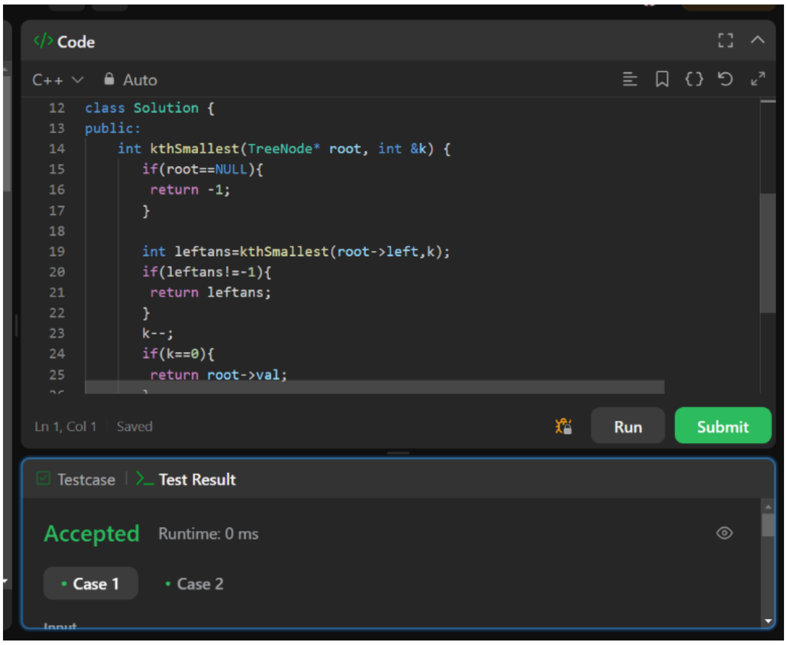


```
public:
    bool solve(TreeNode* root,long long int lb,long long int ub){
     if(root==NULL){
         return true;
     }

     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;
     }
    }
```

Accepted Runtime: 0 ms

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;
  }
```

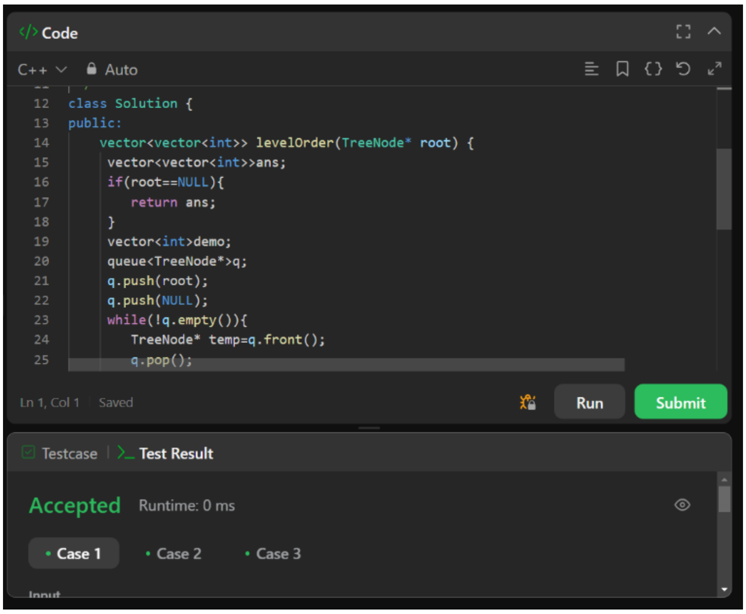


```
class Solution {
public:
    int kthSmallest(TreeNode* root, int &k) {
       if(root==NULL){
        return -1;
       }

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

Accepted Runtime: 0 ms

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;
}
```

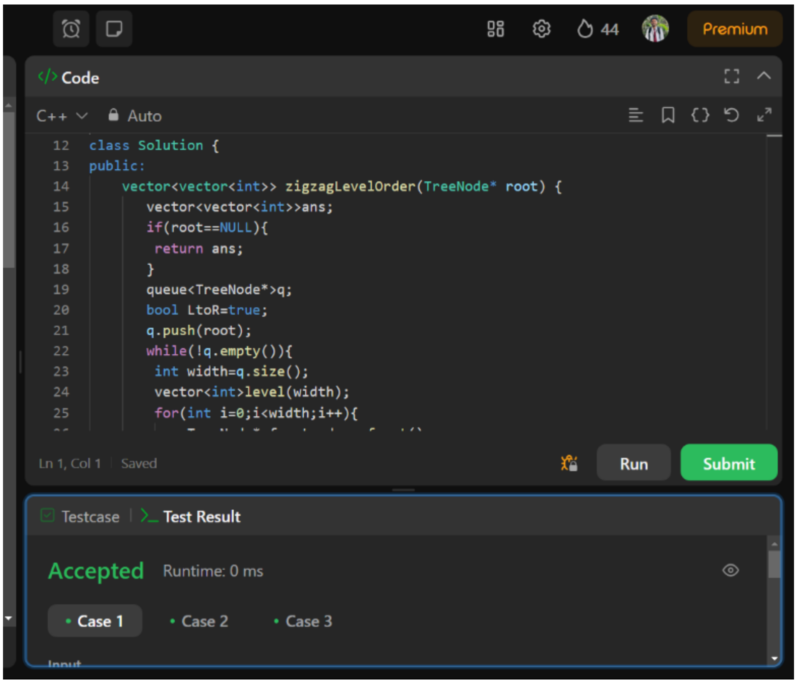


```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
     vector<vector<int>>ans;
     if(root==NULL){
        return ans;
     }
     vector<int>demo;
     queue<TreeNode*>q;
     q.push(root);
     q.push(NULL);
     while(!q.empty()){
        TreeNode* temp=q.front();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

Accepted Runtime: 0 ms

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;
   }
```

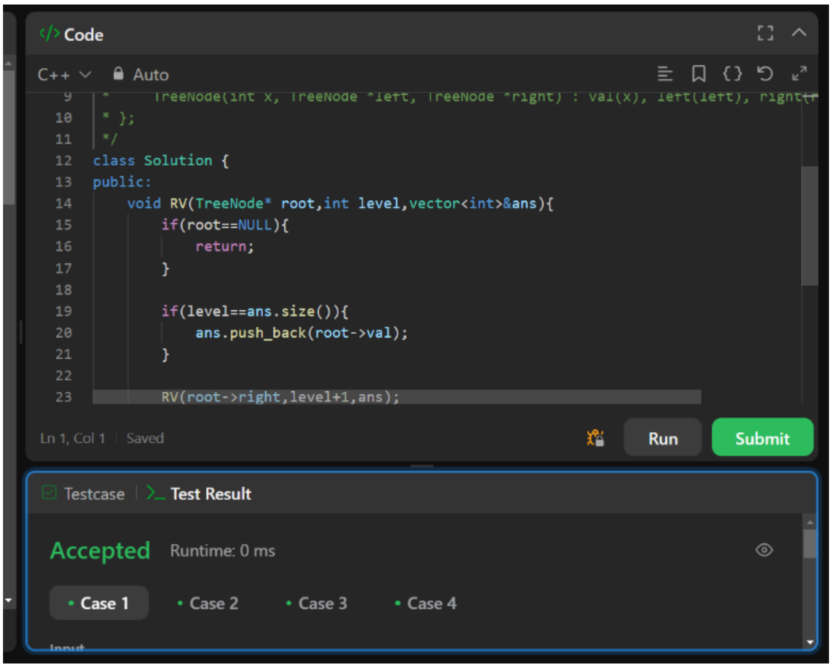


```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>>ans;
        if(root==NULL){
         return ans;
        }
        queue<TreeNode*>q;
        bool LtoR=true;
        q.push(root);
        while(!q.empty()){
         int width=q.size();
         vector<int>level(width);
         for(int i=0;i<width;i++){
```

Accepted Runtime: 0 ms

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;
    }
```



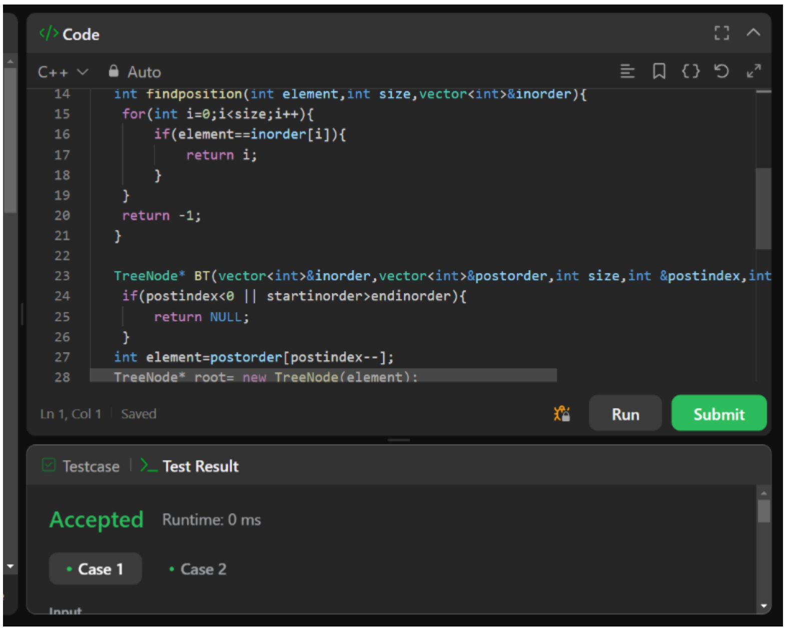
```
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(r
 * };
 */
class Solution {
public:
    void RV(TreeNode* root,int level,vector<int>&ans){
        if(root==NULL){
            return;
        }

        if(level==ans.size()){
            ans.push_back(root->val);
        }

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

Accepted Runtime: 0 ms

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;
```

}

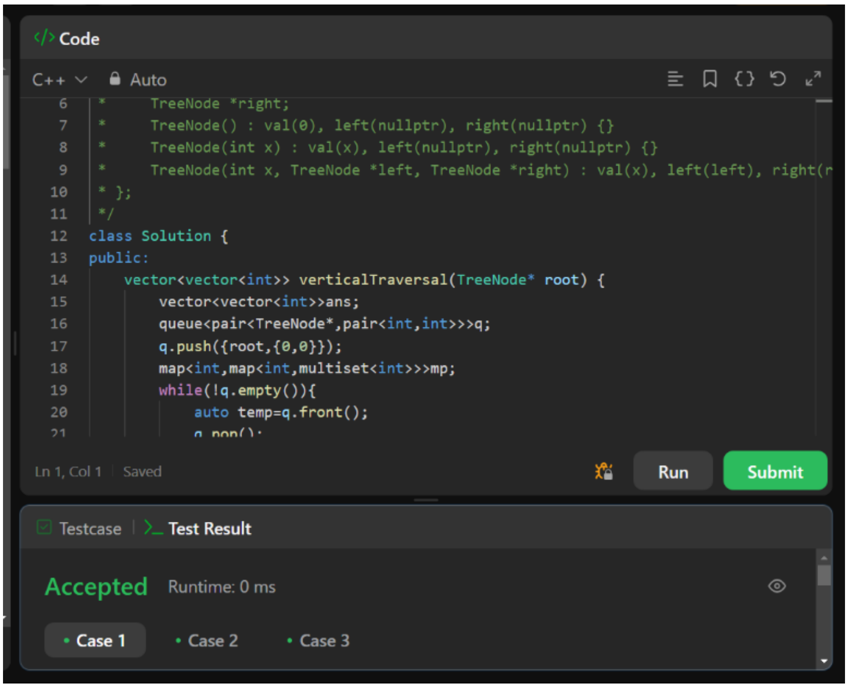


```
int findposition(int element,int size,vector<int>&inorder){
 for(int i=0;i<size;i++){
     if(element==inorder[i]){
         return i;
     }
 }
 return -1;
}

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

Accepted Runtime: 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;
}
```



```
 *     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(r
 * };
 */
class Solution {
public:
    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();
```

Accepted Runtime: 0 ms