104.[Maximum Depth of Binary Tree](https://leetcode.com/problems/maximum-depth-of-binary-tree/description/)

class Solution {

public:

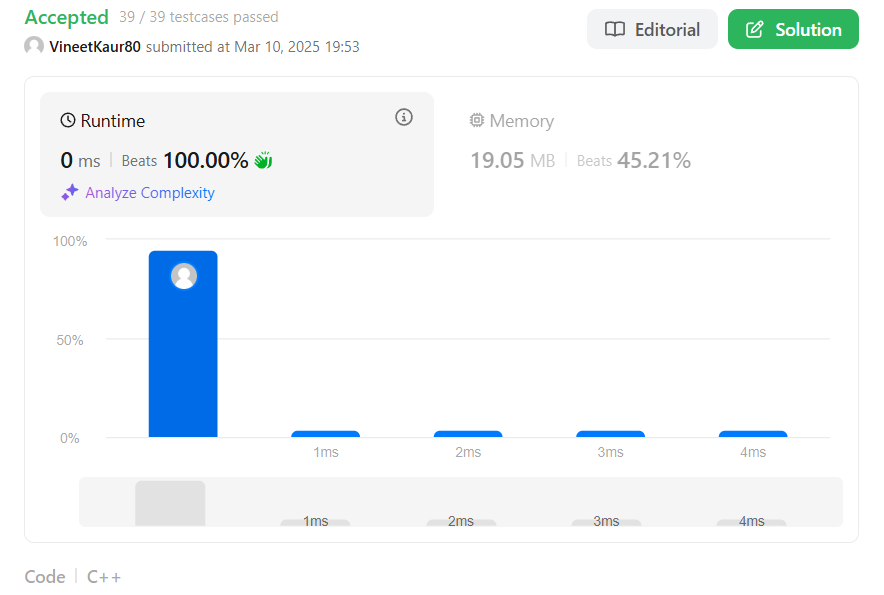
    int maxDepth(TreeNode\* root) {

        if (!root) return 0;

        return 1 + max(maxDepth(root->left), maxDepth(root->right));

    }

};



48.[Validate Binary Search Tree](https://leetcode.com/problems/validate-binary-search-tree/description/)

class Solution {

public:

    void rotate(vector<vector<int>>& matrix) {

        int layer=0;

        while(layer!=matrix.size()/2){

            for(int i=layer;i<matrix.size()-1-layer;i++){

                swap(matrix[matrix.size()-1-layer][i],matrix[matrix.size()-1-i][matrix.size()-1-layer]);//down<->right

                swap(matrix[matrix.size()-1-i][matrix.size()-1-layer],matrix[layer][matrix.size()-1-i]);

                //right<->up

  swap(matrix[layer][matrix.size()-1-i],matrix[i][layer]); //up<->left

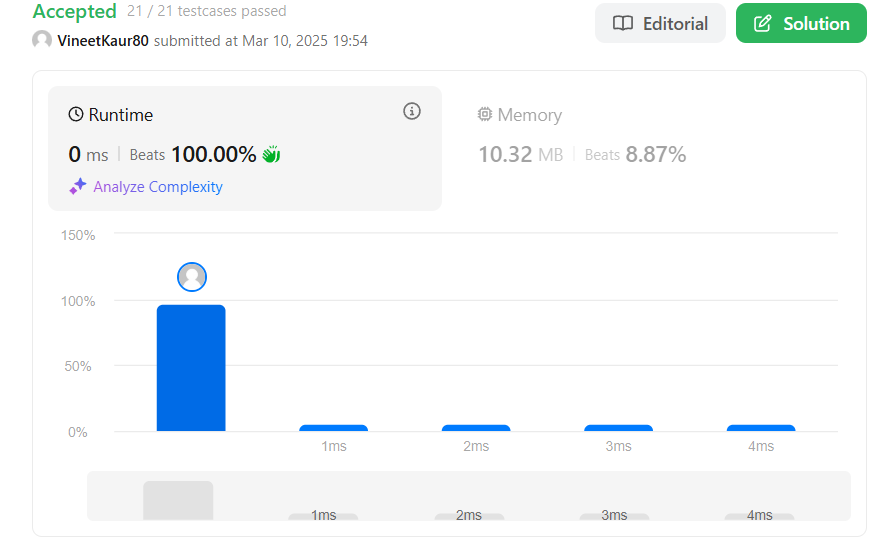
            }

            layer++;

        }

    }

};



101.[Symmetric Tree](https://leetcode.com/problems/symmetric-tree/description/)

class Solution {

public:

    bool isSymmetric(TreeNode\* root) {

        return isMirror(root->left,root->right);

    }

    private:

    bool isMirror(TreeNode\*p,TreeNode\*q){

        if(!p && !q) return true;

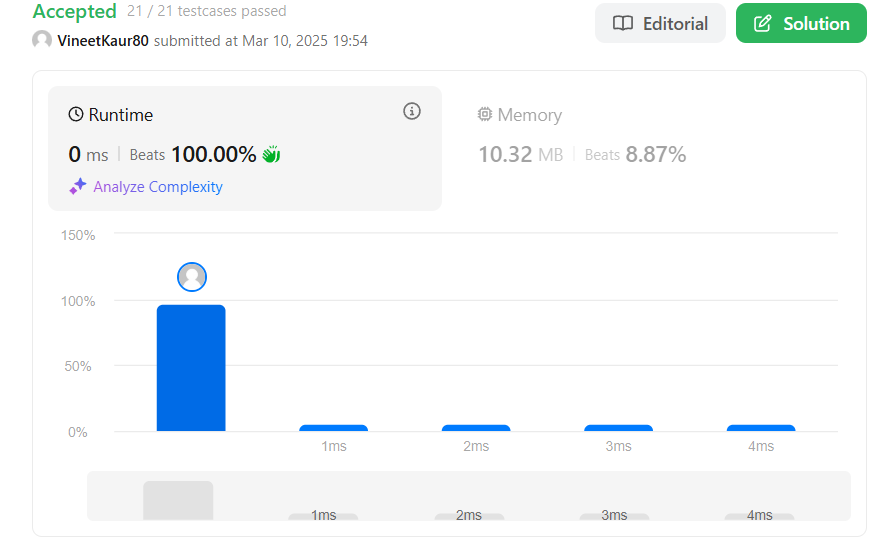
        if(!p || !q) return false;

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

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

    }

};



103.[Binary Tree Zigzag Level Order Traversal](https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/description/)

class Solution {

private:

    void zigzagLevelOrderHelper(TreeNode\* root, vector<int>& temp,

                                vector<vector<int>>& ans) {

        queue<TreeNode\*> q;

        q.push(root);

        while (!q.empty()) {

            int n = q.size();

            while (n--) {

                TreeNode\* node = q.front();

                q.pop();

                temp.push\_back(node->val);

                if (node->left) {

                    q.push(node->left);

                }

                if (node->right) {

                    q.push(node->right);

                }

            }

            ans.push\_back(temp);

            temp.clear();

        }

    }

public:

    vector<vector<int>> zigzagLevelOrder(TreeNode\* root) {

        vector<int> temp;

        vector<vector<int>> ans;

        if (!root)

            return ans;

        zigzagLevelOrderHelper(root, temp, ans);

        int counter = 0;

        for (auto& it : ans) {

            if (counter % 2 == 1) {

                reverse(it.begin(), it.end());

            }

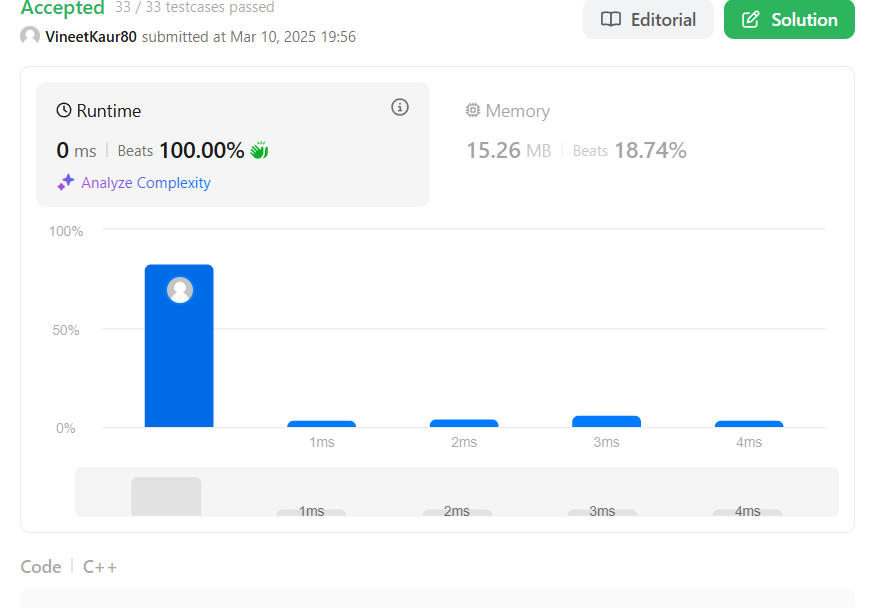
            counter++;

        }

        return ans;

    }

};



236.[Lowest Common Ancestor of a Binary Tree](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/description/)

class Solution {

public:

    TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

        if (root == nullptr || root == p || root == q) {

            return root;

        }

        TreeNode\* left = lowestCommonAncestor(root->left, p, q);

        TreeNode\* right = lowestCommonAncestor(root->right, p, q);

        if (left != nullptr && right != nullptr) {

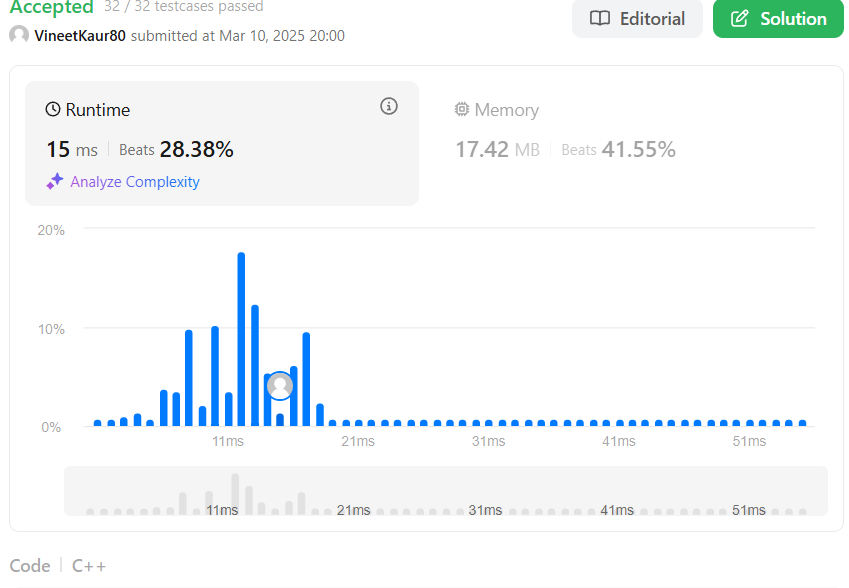
            return root;

        }

        return left != nullptr ? left : right;

    }

};



102.[Binary Tree Level Order Traversal](https://leetcode.com/problems/binary-tree-level-order-traversal/description/)

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);

            }

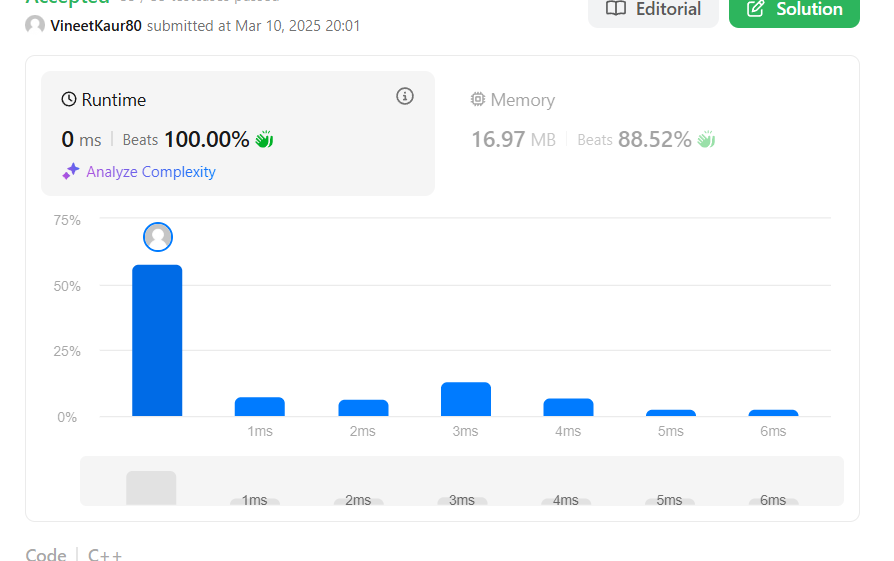
            ans.push\_back(v);

    }

        return ans;

    }

};



230.[Kth Smallest Element in a BST](https://leetcode.com/problems/kth-smallest-element-in-a-bst/description/)

class Solution {

public:

void preOrderTraversal(TreeNode\* root, vector<int> &v){

if(root == NULL) return;

//root, left, right

v.push\_back(root->val);

preOrderTraversal(root->left, v);

preOrderTraversal(root->right, v);

}

int kthSmallest(TreeNode\* root, int k) {

vector<int> v;

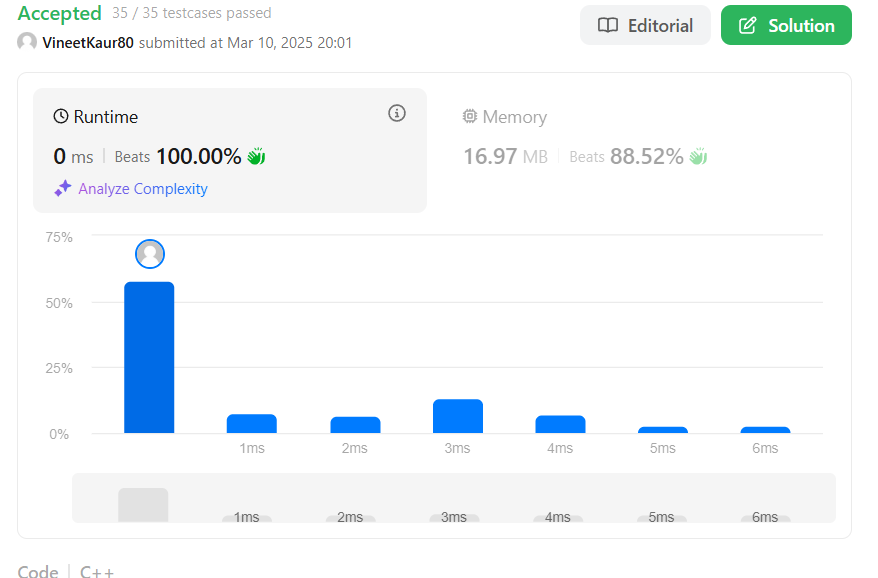
preOrderTraversal(root, v);

sort(v.begin(), v.end());

return v[k-1];

}

};



116.[Populating Next Right Pointers in Each Node](https://leetcode.com/problems/populating-next-right-pointers-in-each-node/description/)

class Solution {

public:

    void preOrderTraversal(TreeNode\* root, vector<int> &v){

        if(root == NULL)    return;

        //root, left, right

        v.push\_back(root->val);

        preOrderTraversal(root->left, v);

        preOrderTraversal(root->right, v);

    }

    int kthSmallest(TreeNode\* root, int k) {

        vector<int> v;

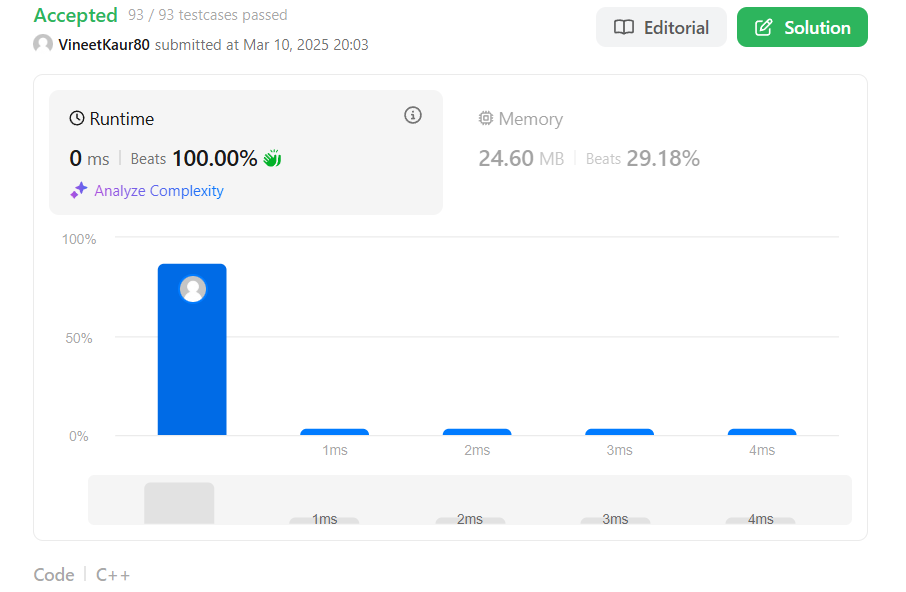
        preOrderTraversal(root, v);

        sort(v.begin(), v.end());

        return v[k-1];

    }

};



404.[Sum of Left Leaves](https://leetcode.com/problems/sum-of-left-leaves/description/)

class Solution {

public:

Node\* connect(Node\* root) {

if(root==nullptr) return {};

queue<Node\*> q;

q.push(root);

while(!q.empty()){

int n = q.size();

for(int i=0;i<n;i++){

Node\* t = q.front();

q.pop();

if(i!=n-1){

t->next=q.front();

}

if(t->left) q.push(t->left);

if(t->right) q.push(t->right);

}

}

return root;

}

};

