**Name:** Utkarsh Prakhar

**UID:** 22BCS16208

**Section:** 22BCS\_IOT-612-B

**Subject:** Advanced Programming Lab-2

**Assignment**

1. **Code: (Maximum Depth of Binary Tree)**

class Solution {

public:

int maxDepth(TreeNode\* root) {

if (root == nullptr) {

return 0;

}

int leftDepth = maxDepth(root->left);

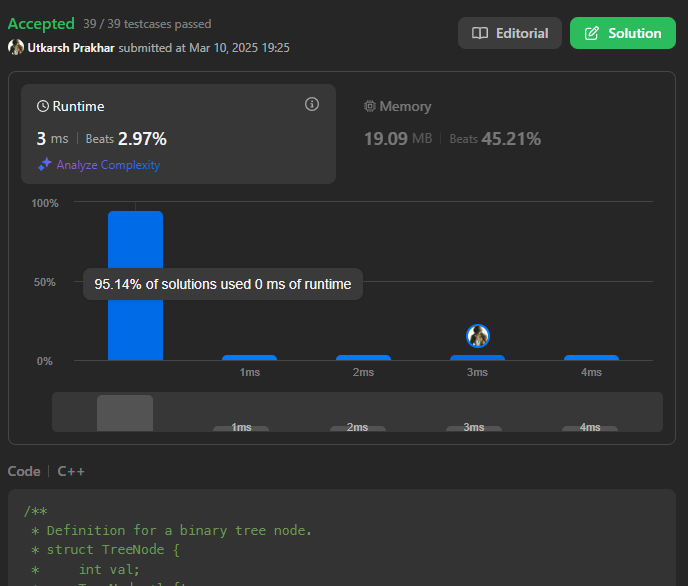
int rightDepth = maxDepth(root->right);

return 1 + max(leftDepth, rightDepth);

}

};

**Output:**

****

1. **Code: (Validate Binary Search Tree)**

class Solution {

public:

bool isValidBST(TreeNode\* root, long minVal, long maxVal) {

if (root == nullptr) return true;

if (root->val <= minVal || root->val >= maxVal) return false;

return isValidBST(root->left, minVal, root->val) &&

isValidBST(root->right, root->val, maxVal);

}

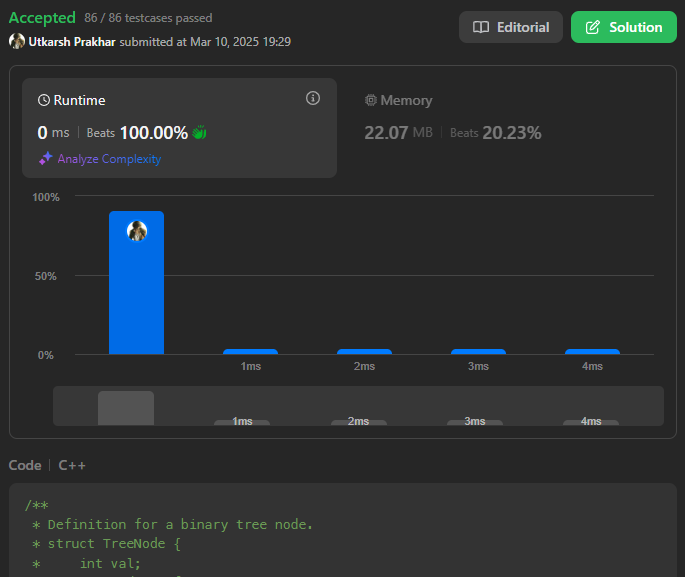
bool isValidBST(TreeNode\* root) {

return isValidBST(root, LONG\_MIN, LONG\_MAX);

}

};

**OUTPUT:**

****

1. **Code: (Symmetric Tree)**

class Solution {

public:

bool isSymmetric(TreeNode\* root) {

if (!root) return true;

queue<TreeNode\*> q;

q.push(root->left);

q.push(root->right);

while (!q.empty()) {

TreeNode\* t1 = q.front(); q.pop();

TreeNode\* t2 = q.front(); q.pop();

if (!t1 && !t2) continue;

if (!t1 || !t2) return false;

if (t1->val != t2->val) return false;

q.push(t1->left);

q.push(t2->right);

q.push(t1->right);

q.push(t2->left);

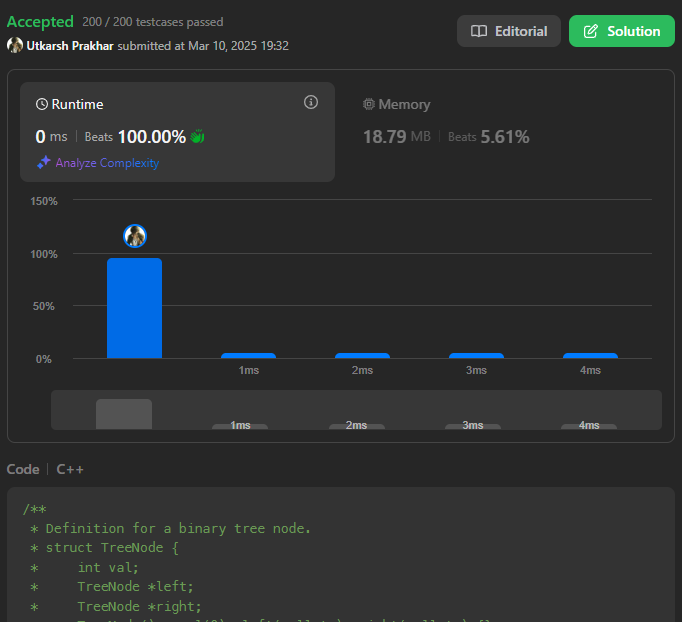
}

return true;

}

};

**Output:**

****

1. **Code: (Binary Tree Zigzag Level Order Trversal)**

class Solution {

public:

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

vector<vector<int>> result;

if (!root) return result;

queue<TreeNode\*> q;

q.push(root);

bool leftToRight = true;

while (!q.empty()) {

int levelSize = q.size();

deque<int> levelNodes;

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

TreeNode\* node = q.front();

q.pop();

if (leftToRight)

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

else

levelNodes.push\_front(node->val);

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

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

}

result.push\_back(vector<int>(levelNodes.begin(), levelNodes.end()));

leftToRight = !leftToRight;

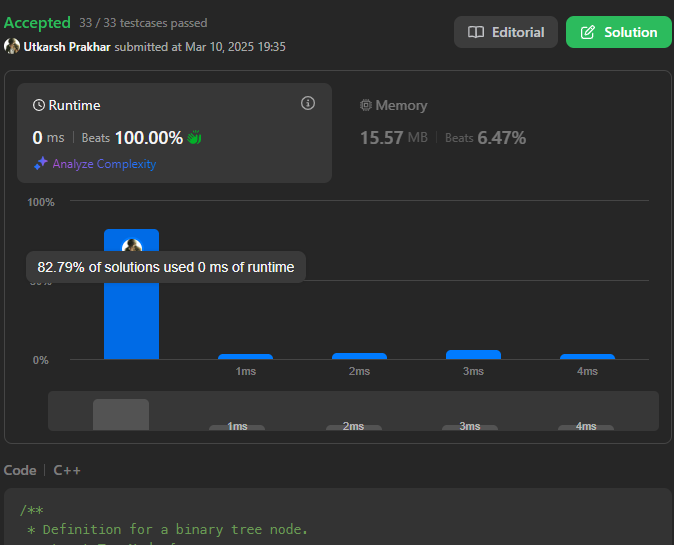
}

return result;

}

};

**Output:**

****

1. **Code: (Lowest common ancestor of a binary tree)**

class Solution {

public:

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

if (!root || root == p || root == q) return root;

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

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

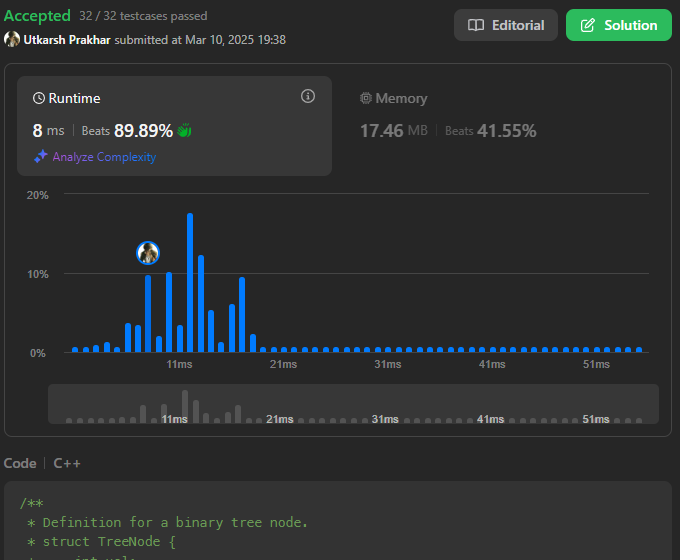
if (left && right) return root;

return left ? left : right;

}

};

**Output:**

****

1. **Code: (Binary Tree Inorder Traversal)**

class Solution {

public:

void inorder(TreeNode\* root, vector<int>& result) {

if (!root) return;

inorder(root->left, result);

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

inorder(root->right, result);

}

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

vector<int> result;

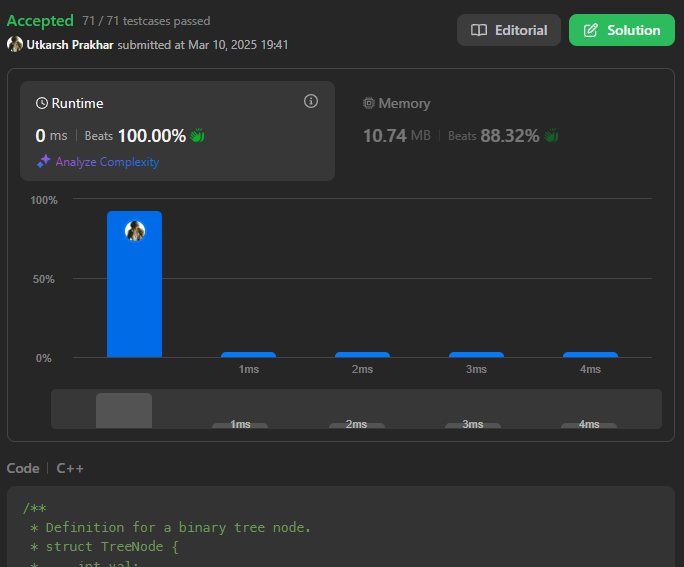
inorder(root, result);

return result;

}

};

**Output:**

****

1. **Code: (Binary Tree Level Order Traversal)**

class Solution {

public:

bool isSymmetric(TreeNode\* root) {

if (!root) return true;

queue<TreeNode\*> q;

q.push(root->left);

q.push(root->right);

while (!q.empty()) {

TreeNode\* t1 = q.front(); q.pop();

TreeNode\* t2 = q.front(); q.pop();

if (!t1 && !t2) continue;

if (!t1 || !t2) return false;

if (t1->val != t2->val) return false;

q.push(t1->left);

q.push(t2->right);

q.push(t1->right);

q.push(t2->left);

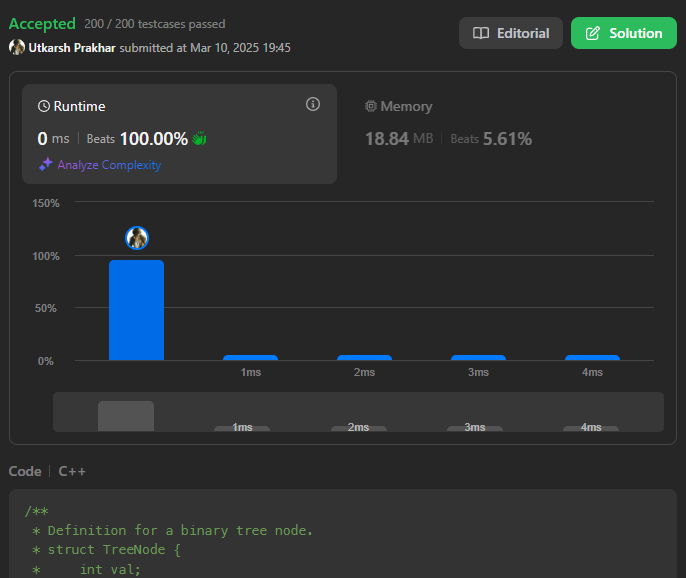
}

return true;

}

};

**Output:**

****

1. **Code: (Kth Samllest Element in a BST)**

class Solution {

public:

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

stack<TreeNode\*> st;

TreeNode\* curr = root;

int count = 0;

while (!st.empty() || curr) {

while (curr) {

st.push(curr);

curr = curr->left;

}

curr = st.top();

st.pop();

count++;

if (count == k) return curr->val;

curr = curr->right;

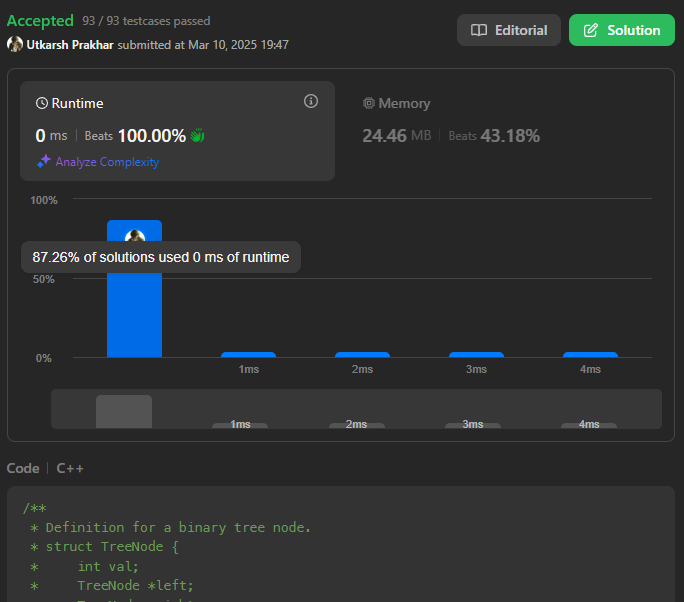
}

return -1;

}

};

**Output:**

****

1. **Code: (Populating Next Right Pointers in Each Node)**

class Solution {

public:

Node\* connect(Node\* root) {

if (!root) return nullptr;

Node\* leftmost = root;

while (leftmost->left) {

Node\* current = leftmost;

while (current) {

current->left->next = current->right;

if (current->next) {

current->right->next = current->next->left;

}

current = current->next;

}

leftmost = leftmost->left;

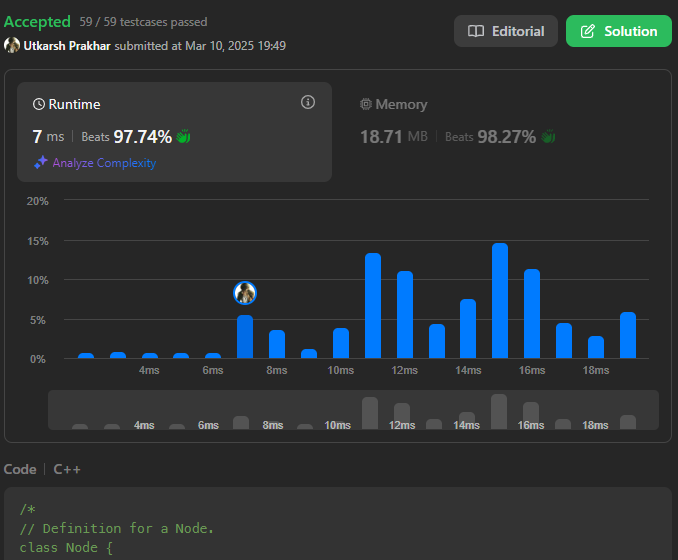
}

return root;

}

};

**Output:**

****

1. **Code: (Sum of Left Leaves)**

class Solution {

public:

int sumOfLeftLeaves(TreeNode\* root) {

if (!root) return 0;

int sum = 0;

queue<TreeNode\*> q;

q.push(root);

while (!q.empty()) {

TreeNode\* node = q.front();

q.pop();

if (node->left && !node->left->left && !node->left->right) {

sum += node->left->val;

}

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

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

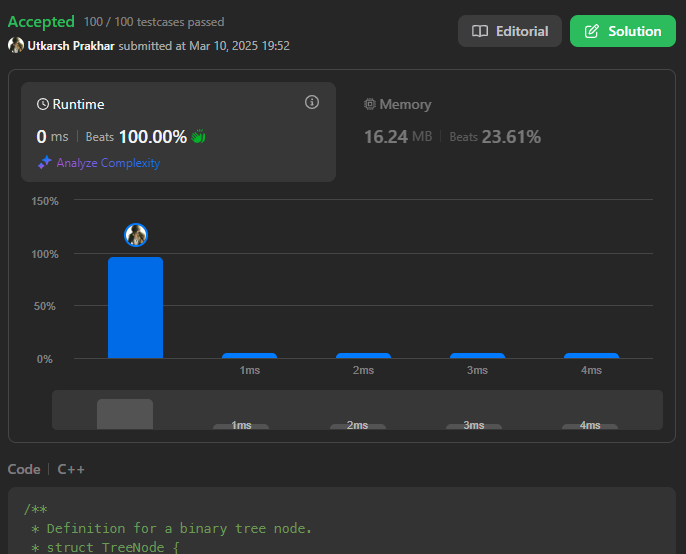
}

return sum;

}

};

**Output:**

****