**ADVANCED PROGRAMMING LAB-2 ASSIGNMENT**

**Submitted By:**

**Name- Ansh Gupta**

**UID- 22BCS14765**

**Section-22BCS\_IOT\_605-B**

1. **Binary Tree Inorder Traversal**

class Solution {

public:

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

if (!node) return;

inorder(node->left, result);

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

inorder(node->right, result);

}

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

vector<int> result;

inorder(root, result);

return result;

}

};

1. **Binary Tree Level Order traversal**

class Solution {

public:

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

vector<vector<int>> result;

if (!root) return result;

queue<TreeNode\*> q;

q.push(root);

while (!q.empty()) {

int levelSize = q.size();

vector<int> level;

while (levelSize--) {

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);

}

return result;

}

};

1. **Symmetric tree**

class Solution {

public:

bool isMirror(TreeNode\* left, TreeNode\* right) {

if (!left || !right) return left == right;

return (left->val == right->val) &&

isMirror(left->left, right->right) &&

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

}

bool isSymmetric(TreeNode\* root) {

return isMirror(root, root);

}

};

1. **Maximum Depth of Binary Tree**

class Solution {

public:

int maxDepth(TreeNode\* root) {

if (!root) return 0;

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

}

};

1. **Validate Binary Search Tree**

class Solution {

public:

bool isValidBST(TreeNode\* root) {

stack<TreeNode\*> nodes;

TreeNode\* prev = nullptr;

while (!nodes.empty() || root) {

while (root) {

nodes.push(root);

root = root->left;

}

root = nodes.top();

nodes.pop();

if (prev && root->val <= prev->val) return false;

prev = root;

root = root->right;

}

return true;

}

};

1. **Kth smallest element in a BST**

class Solution {

public:

int count = 0, result = 0;

void inorder(TreeNode\* root, int k) {

if (!root) return;

inorder(root->left, k);

if (++count == k) {

result = root->val;

return;

}

inorder(root->right, k);

}

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

inorder(root, k);

return result;

}

};

1. **Binary Tree Level Order traversal II**

class Solution {

public:

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

vector<vector<int>> result;

if (!root) return result;

queue<TreeNode\*> q;

q.push(root);

while (!q.empty()) {

int levelSize = q.size();

vector<int> level;

for (int i = 0; i < levelSize; 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.insert(result.begin(), level);

}

return result;

}

};

1. **Binary Tree Zigzag Level Oder Traversal**

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();

vector<int> level;

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

}

if (!leftToRight) reverse(level.begin(), level.end());

result.push\_back(level);

leftToRight = !leftToRight;

}

return result;

}

};

1. **Binary Tree Right Side View**

class Solution {

public:

void dfs(TreeNode\* node, int level, vector<int>& result) {

if (!node) return;

if (level == result.size()) result.push\_back(node->val);

dfs(node->right, level + 1, result);

dfs(node->left, level + 1, result);

}

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

vector<int> result;

dfs(root, 0, result);

return result;

}

};

1. **Construct Binary Tree From Inorder And Postorder Traversal**

class Solution {

public:

unordered\_map<int, int> indexMap;

int postIndex;

TreeNode\* build(vector<int>& inorder, vector<int>& postorder, int left, int right) {

if (left > right) return nullptr;

TreeNode\* root = new TreeNode(postorder[postIndex--]);

int inIndex = indexMap[root->val];

root->right = build(inorder, postorder, inIndex + 1, right);

root->left = build(inorder, postorder, left, inIndex - 1);

return root;

}

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

postIndex = postorder.size() - 1;

for (int i = 0; i < inorder.size(); i++)

indexMap[inorder[i]] = i;

return build(inorder, postorder, 0, inorder.size() - 1);

}

};

1. **Find Bottom Left Tree Value**

class Solution {

public:

int findBottomLeftValue(TreeNode\* root) {

queue<TreeNode\*> q;

q.push(root);

int leftmostValue = 0;

while (!q.empty()) {

int levelSize = q.size();

leftmostValue = q.front()->val;

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

TreeNode\* node = q.front();

q.pop();

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

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

}

}

return leftmostValue;

}

};

1. **Binary Tree Maximum Path Sum**

class Solution {

public:

int maxSum = INT\_MIN;

int dfs(TreeNode\* node) {

if (!node) return 0;

int left = max(0, dfs(node->left));

int right = max(0, dfs(node->right));

maxSum = max(maxSum, left + right + node->val);

return node->val + max(left, right);

}

int maxPathSum(TreeNode\* root) {

dfs(root);

return maxSum;

}

};

1. **Vertical Order Traversal Of A Binary Tree**

class Solution {

public:

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

map<int, map<int, multiset<int>>> nodes;

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

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

while (!q.empty()) {

auto [node, pos] = q.front();

q.pop();

nodes[pos.first][pos.second].insert(node->val);

if (node->left) q.push({node->left, {pos.first - 1, pos.second + 1}});

if (node->right) q.push({node->right, {pos.first + 1, pos.second + 1}});

}

vector<vector<int>> result;

for (auto& [x, col] : nodes)

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

return result;

}

};