

Day-6

Name – Ishita

Faculty Name- Er. Rajni Devi

UID- 22BCS15353

Date- 23 Dec ,2024

Section – 620-B

Very Easy

Ques 1:

Binary Tree Inorder Traversal

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

Code:

```
#include <iostream>

#include <vector>

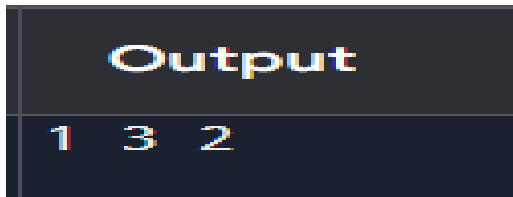
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

void inorder(TreeNode *root, vector<int> &result) {
    if (!root) return;
    inorder(root->left, result);
    result.push_back(root->val);
    inorder(root->right, result);
}

int main() {
    TreeNode *root = new TreeNode(1);
    root->right = new TreeNode(2);
    root->right->left = new TreeNode(3);
    vector<int> result;
    inorder(root, result);
    for (int val : result) cout << val << " ";
    return 0;
}
```

Output:



Ques 2:

Count Complete Tree Nodes

Given the root of a complete binary tree, return the number of the nodes in the tree.

Code:

```
#include <iostream>
#include <cmath>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};
int getDepth(TreeNode *node) {
    int depth = 0;
    while (node) {
        depth++;
        node = node->left;
    }
    return depth;
}
int countNodes(TreeNode *root) {
    if (!root) return 0;

    int leftDepth = getDepth(root->left);
    int rightDepth = getDepth(root->right);
    if (leftDepth == rightDepth)
        return (1 << leftDepth) + countNodes(root->right);
```

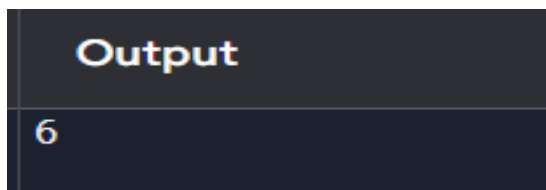
```

else
    return (1 << rightDepth) + countNodes(root->left);
}

int main() {
    TreeNode *root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->left = new TreeNode(6);
    cout << countNodes(root);
    return 0;
}

```

Output:



Easy

Ques 3:

Same Tree

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Code:

```

#include <iostream>

using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

bool isSameTree(TreeNode *p, TreeNode *q) {
    if (!p && !q) return true;

```

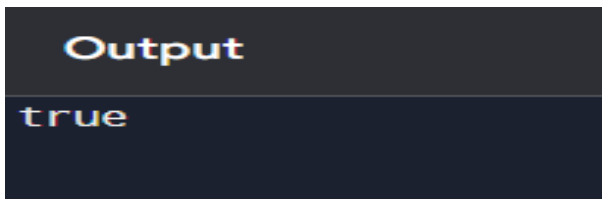
```

    if (!p || !q || p->val != q->val) return false;
    return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
}

int main() {
    TreeNode *p = new TreeNode(1);
    p->left = new TreeNode(2);
    p->right = new TreeNode(3);
    TreeNode *q = new TreeNode(1);
    q->left = new TreeNode(2);
    q->right = new TreeNode(3);
    cout << (isSameTree(p, q) ? "true" : "false") << endl;
    return 0;
}

```

Output:



Ques 4:

Invert Binary Tree

Given the root of a binary tree, invert the tree, and return its root.

Code:

```

#include <iostream>
#include <queue>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};
TreeNode* invertTree(TreeNode *root) {
    if (!root) return NULL;
    swap(root->left, root->right);
}

```

```

invertTree(root->left);
invertTree(root->right);
return root;}

void printTree(TreeNode *root) {
    if (!root) return;
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
        TreeNode *node = q.front();
        q.pop();
        if (node) {
            cout << node->val << " ";
            q.push(node->left);
            q.push(node->right);
        } else {
            cout << "-1 ";}}}

int main() {
    TreeNode *root = new TreeNode(4);
    root->left = new TreeNode(2);
    root->right = new TreeNode(7);
    root->left->left = new TreeNode(1);
    root->left->right = new TreeNode(3);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(9);
    root = invertTree(root);
    printTree(root);
    return 0;}

```

Output:

Output

```
4 7 2 9 6 3 1 -1 -1 -1 -1 -1 -1 -1
```

```
=== Code Execution Successful ===
```

Medium

Ques 5:

Construct Binary Tree from Preorder and Inorder Traversal

Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree.

Code:

```
#include <iostream>

using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

TreeNode* buildTreeHelper(int preorder[], int& preIndex, int inorder[], int inStart, int inEnd)
{
    if (inStart > inEnd) return NULL;
    int rootVal = preorder[preIndex++];
    TreeNode* root = new TreeNode(rootVal);
    int inIndex = inStart;
    while (inorder[inIndex] != rootVal) {
        inIndex++;
    }
    root->left = buildTreeHelper(preorder, preIndex, inorder, inStart, inIndex - 1);
    root->right = buildTreeHelper(preorder, preIndex, inorder, inIndex + 1, inEnd);
    return root;
}

TreeNode* buildTree(int preorder[], int inorder[], int n) {
    int preIndex = 0;
    return buildTreeHelper(preorder, preIndex, inorder, 0, n - 1);
}
```

```

void printTree(TreeNode* root) {
    if (!root) {
        cout << "null ";
        return;
    }
    cout << root->val << " ";
    printTree(root->left);
    printTree(root->right);
}

int main() {
    int preorder[] = {3, 9, 20, 15, 7};
    int inorder[] = {9, 3, 15, 20, 7};
    int n = sizeof(preorder) / sizeof(preorder[0]);
    TreeNode* root = buildTree(preorder, inorder, n);
    printTree(root);
    return 0;
}

```

Output:

```

Output
3 9 null null 20 15 null null 7 null null

```

Ques 6:

Sum Root to Leaf Numbers

You are given the root of a binary tree containing digits from 0 to 9 only.

Each root-to-leaf path in the tree represents a number.

Code:

```

#include <iostream>
using namespace std;

struct TreeNode {
    int val;

```

```

TreeNode* left;
TreeNode* right;
TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
    int sumNumbers(TreeNode* root) {
        return dfs(root, 0);}
private:
    int dfs(TreeNode* node, int currentSum) {
        if (!node) {
            return 0;
        }
        currentSum = currentSum * 10 + node->val;
        if (!node->left && !node->right) {
            return currentSum;
        }
        return dfs(node->left, currentSum) + dfs(node->right, currentSum);}
};

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    Solution solution;
    int result = solution.sumNumbers(root);
    cout << "Sum of all root-to-leaf numbers: " << result << endl; // Output: 25
    return 0;
}

```

Output:

```

Output
Sum of all root-to-leaf numbers: 25

```


Hard

Ques 7:

Binary Tree Right Side View

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.

Code:

```
#include <iostream>

#include <vector>

#include <queue>

using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int> result;
        if (!root) {
            return result;
        }
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            int size = q.size();
```

```

        for (int i = 0; i < size; i++) {
            TreeNode* node = q.front();

            q.pop();

            if (i == size - 1) {
                result.push_back(node->val);
            }

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

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

    return result;
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->right = new TreeNode(5);
    root->right->right = new TreeNode(4);

    Solution solution;

    vector<int> result = solution.rightSideView(root);

```

```

    cout << "Right side view of the tree: ";

    for (int val : result) {

        cout << val << " ";

    }

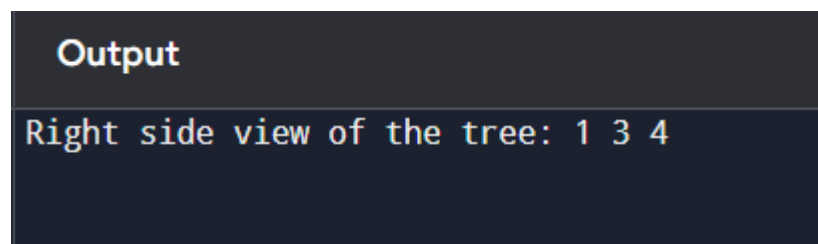
    cout << endl; // Output: [1, 3, 4]

    return 0;

}

```

Output:



```

Output
Right side view of the tree: 1 3 4

```

Ques 8:

Binary Tree Maximum Path Sum

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.

Code:

```

#include <iostream>

#include <algorithm>

#include <climits>

using namespace std;

struct TreeNode {

    int val;

    TreeNode* left;

    TreeNode* right;

    TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};

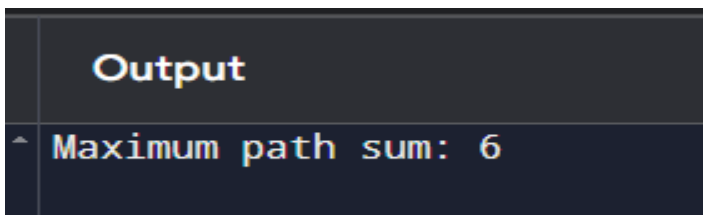
```

```

class Solution {
public:
    int maxPathSum(TreeNode* root) {
        int maxSum = INT_MIN;
        maxGain(root, maxSum);
        return maxSum;
    }
private:
    int maxGain(TreeNode* node, int& maxSum) {
        if (!node) return 0;
        int leftGain = max(maxGain(node->left, maxSum), 0);
        int rightGain = max(maxGain(node->right, maxSum), 0);
        int currentSum = node->val + leftGain + rightGain;
        maxSum = max(maxSum, currentSum);
        return node->val + max(leftGain, rightGain);}
int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    Solution solution;
    int result = solution.maxPathSum(root);
    cout << "Maximum path sum: " << result << endl;
    return 0;
}

```

Output:



```

Output
Maximum path sum: 6

```

Very Hard

Ques 9:

Longest Path With Different Adjacent Characters

You are given a tree (i.e. a connected, undirected graph that has no cycles) rooted at node 0 consisting of n nodes numbered from 0 to n - 1. The tree is represented by a 0-indexed array parent of size n, where parent[i] is the parent of node i. Since node 0 is the root, parent[0] == -1.

You are also given a string s of length n, where s[i] is the character assigned to node i.

Return the length of the longest path in the tree such that no pair of adjacent nodes on the path have the same character assigned to them.

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

class Solution {

public:

    int longestPath(vector<int>& parent, string& s) {

        int n = parent.size();

        vector<vector<int>>> adj(n);

        for (int i = 1; i < n; ++i) {

            adj[parent[i]].push_back(i);

        }

        vector<int> dp(n, -1);

        int longest = 0;

        dfs(0, adj, s, dp, longest); // Start DFS from the root

        return longest;

    }

}
```

private:

```
int dfs(int node, const vector<vector<int>>& adj, const string& s, vector<int>& dp, int& longest) {
```

```
    int maxLength1 = 0; // First longest path
```

```
    int maxLength2 = 0; // Second longest path
```

```
    for (int neighbor : adj[node]) {
```

```
        int length = dfs(neighbor, adj, s, dp, longest);
```

```
        if (s[node] != s[neighbor]) {
```

```
            if (length > maxLength1) {
```

```
                maxLength2 = maxLength1;
```

```
                maxLength1 = length;
```

```
            } else if (length > maxLength2) {
```

```
                maxLength2 = length;
```

```
            }
```

```
        }
```

```
    }
```

```
    dp[node] = maxLength1 + 1;
```

```
    longest = max(longest, maxLength1 + maxLength2 + 1); // Update longest path found
```

```
    return maxLength1 + 1; // Return the longest path including the current node
```

```
}
```

```
};
```

```
int main() {
```

```
    Solution solution;
```

```
    vector<int> parent = {-1, 0, 0, 1, 1, 2};
```

```
    string s = "abacbe";
```

```
    int result = solution.longestPath(parent, s);
```

```
    cout << "Longest path with different adjacent characters: " << result << endl; // Output: 3
```

```
    return 0;
}
```

Output:

Output

```
Longest path with different adjacent characters: 3
```

Ques 10:

Count Paths That Can Form a Palindrome in a Tree

You are given a tree (i.e. a connected, undirected graph that has no cycles) rooted at node 0 consisting of n nodes numbered from 0 to $n - 1$. The tree is represented by a 0-indexed array `parent` of size n , where `parent[i]` is the parent of node i . Since node 0 is the root, `parent[0] == -1`.

You are also given a string `s` of length n , where `s[i]` is the character assigned to the edge between i and `parent[i]`. `s[0]` can be ignored.

Return the number of pairs of nodes (u, v) such that $u < v$ and the characters assigned to edges on the path from u to v can be rearranged to form a palindrome.

A string is a palindrome when it reads the same backwards as forwards.

Code:

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;
class Solution {
public:
    int countPalindromePaths(vector<int>& parent, string& s) {
        int n = parent.size();
        vector<vector<int>>> adj(n);
        for (int i = 1; i < n; ++i) {
            adj[parent[i]].push_back(i);
        }
    }
};
```

```

unordered_map<int, int> freqMap;
int result = 0;
int bitmask = 0;
freqMap[0] = 1;
dfs(0, adj, s, bitmask, freqMap, result);
return result;
}

```

private:

```

void dfs(int node, vector<vector<int>>& adj, string& s, int bitmask, unordered_map<int,
int>& freqMap, int& result) {
    bitmask ^= (1 << (s[node] - 'a'));
    result += freqMap[bitmask];
    for (int i = 0; i < 26; ++i) {
        result += freqMap[bitmask ^ (1 << i)];
    }
    freqMap[bitmask]++;
    for (int neighbor : adj[node]) {
        dfs(neighbor, adj, s, bitmask, freqMap, result);
    }
    freqMap[bitmask]--;
}

int main() {
    Solution solution;
    vector<int> parent = {-1, 0, 0, 1, 1, 2};
    string s = "acaabc";
    int result = solution.countPalindromePaths(parent, s);
    cout << "Number of valid palindrome paths: " << result << endl;
    return 0;
}

```

Output:

Output

Number of valid palindrome paths: 9