DSA Questions(Trees)

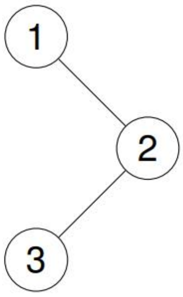
MEGHA SHREE
UID - 22BCS10381

Very Easy:

1. Binary Tree Inorder Traversal

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

Example 1:
Input: root = [1,null,2,3]
Output: [1,3,2]
Explanation:



Constraints:
The number of nodes in the tree is in the range [0, 100].
-100 <= Node.val <= 100

CODE -

```
#include <iostream>
#include <vector>
using namespace std;

struct TreeNode {
   int val;
   TreeNode* left;
   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(right)
{}
};

class Solution {
public:
   vector<int> inorderTraversal(TreeNode* root) {
      vector<int> result;
```

```
        inorder(root, result);
        return result;
    }

private:
    void inorder(TreeNode* node, vector<int>& result) {
        if (node == nullptr) return;
        inorder(node->left, result);
        result.push_back(node->val);
        inorder(node->right, result);
    }
};

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

    Solution sol;
    vector<int> result = sol.inorderTraversal(root);

    for (int val : result) {
        cout << val << " ";
    }
    return 0;
}
```

OUTPUT -

2. Count Complete Tree Nodes

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

According to Wikipedia, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes inclusive at the last level h.

Design an algorithm that runs in less than O(n) time complexity.

Example 1:
Input: root = [1,2,3,4,5,6]
Output: 6

CODE-

```
#include <iostream>
#include <cmath>
using namespace std;

struct TreeNode {
   int val;
   TreeNode* left;
   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(right)
{}
};

class Solution {
public:
   int countNodes(TreeNode* root) {
      if (!root) return 0;
      int leftHeight = getHeight(root->left);
      int rightHeight = getHeight(root->right);
      if (leftHeight == rightHeight)
         return (1 << leftHeight) + countNodes(root->right);
      else
         return (1 << rightHeight) + countNodes(root->left);
   }

private:
   int getHeight(TreeNode* node) {
      int height = 0;
      while (node) {
         height++;
         node = node->left;
      }
      return height;
   }
};

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);
```

```
    Solution sol;
    cout << sol.countNodes(root) << endl;

    return 0;
}
```

OUTPUT -

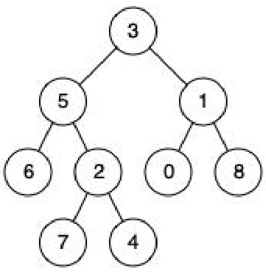
Medium:

3. Lowest Common Ancestor of a Binary Tree

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself).

Example 1:



Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.

CODE-

```
#include <iostream>
using namespace std;
```

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

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;
    }
};

int main() {
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(5);
    root->right = new TreeNode(1);
    root->left->left = new TreeNode(6);
    root->left->right = new TreeNode(2);
    root->right->left = new TreeNode(0);
    root->right->right = new TreeNode(8);
    root->left->right->left = new TreeNode(7);
    root->left->right->right = new TreeNode(4);

    TreeNode* p = root->left;
    TreeNode* q = root->right;

    Solution sol;
    TreeNode* lca = sol.lowestCommonAncestor(root, p, q);
    cout << "LCA: " << lca->val << endl;

    return 0;
}
```

OUTPUT-

4. Sum Root to Leaf Numbers

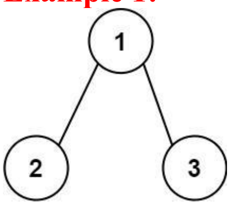
You are given the root of a binary tree containing digits from 0 to 9 only.

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

For example, the root-to-leaf path 1 -> 2 -> 3 represents the number 123.
Return the total sum of all root-to-leaf numbers. Test cases are generated so that the answer will fit in a 32-bit integer.

A leaf node is a node with no children.

Example 1:



Input: root = [1,2,3]Output: 25
Explanation:
The root-to-leaf path 1->2 represents the number 12.
The root-to-leaf path 1->3 represents the number 13.
Therefore, sum = 12 + 13 = 25.

CODE-

```
#include <iostream>
using namespace std;

struct TreeNode {
   int val;
   TreeNode *left;
   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(right) {}
};

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

    int sumNumbers(TreeNode* root) {
        return dfs(root, 0);
    }
};

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

    Solution solution;
    cout << solution.sumNumbers(root) << endl;

    return 0;
}
```

OUTPUT-

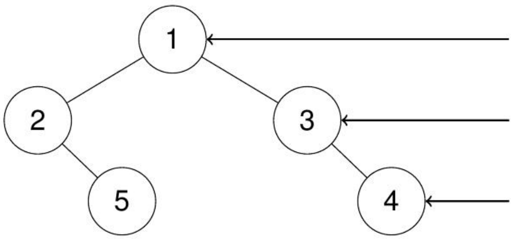
Hard :

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.

Example 1:

Input: root = [1,2,3,null,5,null,4]
Output: [1,3,4]



Explanation:

CODE =

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    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(right)
{}
};

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 it's the rightmost node at the current level
                if (i == size - 1) {
                    result.push_back(node->val);
                }

                // Push left and right children to the queue
                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);

    for (int val : result) {
        cout << val << " ";
    }

    return 0;
}
```

OUTPUT -

```
1 3 4

...Program finished with exit code 0
Press ENTER to exit console.
```