DOMAIN WINTER WINNING CAMP

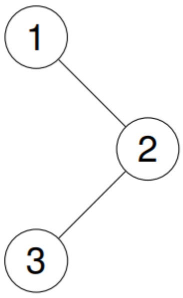
Student Name: Rajat Katiyar
Branch: CSE
UID: 22BCS15928
Section/Group: 22BCS_FL_IOT-603/B

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:



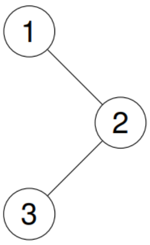
CODE:
```
#include <iostream>
#include <vector>
using namespace std;

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

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;
}

int main() {
    // Create tree: [1, null, 2, 3]
    TreeNode* root = new TreeNode(1);
    root->right = new TreeNode(2);
    root->right->left = new TreeNode(3);

    vector<int> result = inorderTraversal(root);
    for (int val : result) {
        cout << val << " ";
    }
    cout << endl;

    return 0;
}
```

2.Binary Tree Preorder Traversal

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

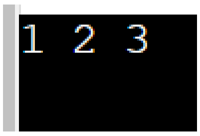
Example 1:
Input: root = [1,null,2,3]
Output: [1,2,3]
Explanation:



CODE:

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

struct TreeNode {
    int val;
    TreeNode *left;
```

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

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

vector<int> preorderTraversal(TreeNode* root) {
    vector<int> result;
    preorder(root, result);
    return result;
}

int main() {
    // Create tree: [1, null, 2, 3]
    TreeNode* root = new TreeNode(1);
    root->right = new TreeNode(2);
    root->right->left = new TreeNode(3);

    vector<int> result = preorderTraversal(root);
    for (int val : result) {
        cout << val << " ";
    }
    cout << endl;

    return 0;
}
```



3. Binary Tree - Sum of All Nodes

Given the root of a binary tree, you need to find the sum of all the node values in the binary tree.

Example 1:
Input: root = [5, 2, 6, 1, 3, 4, 7]
Output: 28
Explanation: The sum of all nodes is 5 + 2 + 6 + 1 + 3 + 4 + 7 = 28.
CODE:

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

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

int sumOfAllNodes(TreeNode* root) {
    if (!root) return 0;
    return root->val + sumOfAllNodes(root->left) + sumOfAllNodes(root->right);
}

int main() {
    // Create tree: [5, 2, 6, 1, 3, 4, 7]
    TreeNode* root = new TreeNode(5);
    root->left = new TreeNode(2);
    root->right = new TreeNode(6);
    root->left->left = new TreeNode(1);
    root->left->right = new TreeNode(3);
    root->right->left = new TreeNode(4);
    root->right->right = new TreeNode(7);

    cout << "Sum of all nodes: " << sumOfAllNodes(root) << endl;

    return 0;
}
```

```
Sum of all nodes: 28
```

Easy:

1. Same Tree

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

Example 1:
Input: p = [1,2,3], q = [1,2,3]
Output: true
<u>CODE:</u>

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

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
```

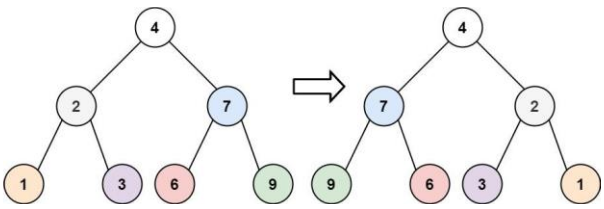
```
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
bool isSameTree(TreeNode* p, TreeNode* q) {
    if (!p && !q) return true; // Both trees are empty
    if (!p || !q || p->val != q->val) return false; // Structure or value mismatch
    return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
}

int main() {
    // Create tree p: [1, 2, 3]
    TreeNode* p = new TreeNode(1);
    p->left = new TreeNode(2);
    p->right = new TreeNode(3);

    // Create tree q: [1, 2, 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;
}.
```

```
true
```

2. Invert Binary Tree

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

Example 1:



Input: root = [4,2,7,1,3,6,9]
Output: [4,7,2,9,6,3,1]
<u>CODE:</u>

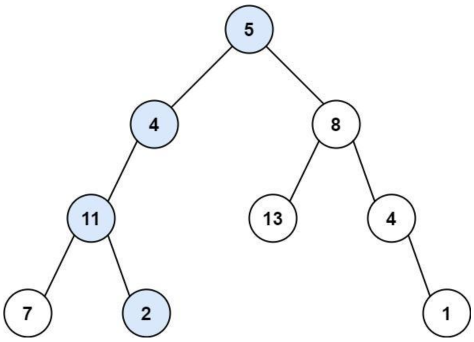
```
#include <iostream>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
```

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

TreeNode* invertTree(TreeNode* root) {
    if (!root) return nullptr;
    swap(root->left, root->right); // Swap left and right child
    invertTree(root->left);
    invertTree(root->right);
    return root;
}

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

int main() {
    // Create tree: [4, 2, 7, 1, 3, 6, 9]
    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);

    preorderTraversal(root); // Expected output: 4 7 9 6 2 3 1
    cout << endl;

    return 0;
}
```

3. Path Sum

Given a binary tree and a sum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. Return false if no such path can be found.

Example 1:



Input: root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22
Output: true
Explanation: The root-to-leaf path with the target sum is shown.
<u>CODE:</u>

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

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

bool hasPathSum(TreeNode* root, int targetSum) {
    if (!root) return false;
    if (!root->left && !root->right) return root->val == targetSum; // Check if leaf
    return hasPathSum(root->left, targetSum - root->val) || hasPathSum(root->right, targetSum -
root->val);
}

int main() {
    // Create tree: [5, 4, 8, 11, null, 13, 4, 7, 2, null, null, null, 1]
    TreeNode* root = new TreeNode(5);
    root->left = new TreeNode(4);
    root->right = new TreeNode(8);
    root->left->left = new TreeNode(11);
    root->left->left->left = new TreeNode(7);
    root->left->left->right = new TreeNode(2);
    root->right->left = new TreeNode(13);
    root->right->right = new TreeNode(4);
    root->right->right->right = new TreeNode(1);

    int targetSum = 22;
    cout << (hasPathSum(root, targetSum) ? "true" : "false") << endl;

    return 0;
```

```
}
```

```
true
```

Medium:

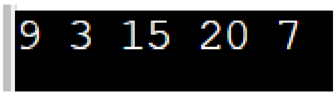
1. Construct Binary Tree from Preorder and Inorder Traversal

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.

Example 1:
Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
Output: [3,9,20,null,null,15,7]
<u>CODE:</u>

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

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

TreeNode* buildTreeHelper(vector<int>& preorder, int preStart, int preEnd,
                          vector<int>& inorder, int inStart, int inEnd,
                          unordered_map<int, int>& inMap) {
    if (preStart > preEnd || inStart > inEnd) return nullptr;

    TreeNode* root = new TreeNode(preorder[preStart]);
    int inRoot = inMap[root->val];
    int numsLeft = inRoot - inStart;

    root->left = buildTreeHelper(preorder, preStart + 1, preStart + numsLeft, inorder, inStart, inRoot - 1, inMap);
    root->right = buildTreeHelper(preorder, preStart + numsLeft + 1, preEnd, inorder, inRoot + 1, inEnd, inMap);

    return root;
}

TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
    unordered_map<int, int> inMap;
    for (int i = 0; i < inorder.size(); i++) inMap[inorder[i]] = i;
    return buildTreeHelper(preorder, 0, preorder.size() - 1, inorder, 0, inorder.size() - 1, inMap);
```

```
}

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

int main() {
    vector<int> preorder = {3, 9, 20, 15, 7};
    vector<int> inorder = {9, 3, 15, 20, 7};

    TreeNode* root = buildTree(preorder, inorder);

    printInorder(root); // Output: 9 3 15 20 7
    cout << endl;

    return 0;
}
```



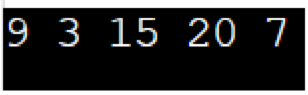
2. Construct Binary Tree from Inorder and Postorder Traversal

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

Example 1:
Input: inorder = [9,3,15,20,7], postorder = [9,15,7,20,3]
Output: [3,9,20,null,null,15,7]
CODE:

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

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}};
TreeNode* buildTreeHelper(vector<int>& inorder, int inStart, int inEnd,
                          vector<int>& postorder, int postStart, int postEnd,
                          unordered_map<int, int>& inMap) {
```

```
    if (inStart > inEnd || postStart > postEnd) return nullptr;

    TreeNode* root = new TreeNode(postorder[postEnd]);
    int inRoot = inMap[root->val];
    int numsLeft = inRoot - inStart;

    root->left = buildTreeHelper(inorder, inStart, inRoot - 1, postorder, postStart, postStart + numsLeft
- 1, inMap);
    root->right = buildTreeHelper(inorder, inRoot + 1, inEnd, postorder, postStart + numsLeft,
postEnd - 1, inMap);

    return root;
}

TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
    unordered_map<int, int> inMap;
    for (int i = 0; i < inorder.size(); i++) inMap[inorder[i]] = i;

    return buildTreeHelper(inorder, 0, inorder.size() - 1, postorder, 0, postorder.size() - 1, inMap);
}

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

int main() {
    vector<int> inorder = {9, 3, 15, 20, 7};
    vector<int> postorder = {9, 15, 7, 20, 3};

    TreeNode* root = buildTree(inorder, postorder);

    printInorder(root); // Output: 9 3 15 20 7
    cout << endl;

    return 0;
}
```



3. Sum Root to Leaf Numbers

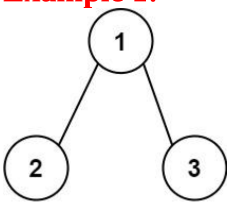
You are given the root of a binary tree containing digits from 0 to 9 only.

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:

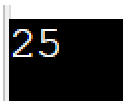


Input: root = [1,2,3]Output: 25
Explanation:
The root-to-leaf path 1->2 represents the number 12.
The root-to-leaf path 1->3 represents the number 13.
Therefore, sum = 12 + 13 = 25.

CODE:

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

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

int sumNumbersHelper(TreeNode* root, int currentSum) {
    if (!root) return 0;
    currentSum = currentSum * 10 + root->val;
    if (!root->left && !root->right) return currentSum; // Leaf node
    return sumNumbersHelper(root->left, currentSum) + sumNumbersHelper(root->right,
currentSum);
}

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

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

```
    cout << sumNumbers(root) << endl; // Output: 25

    return 0;
}
```



Hard:

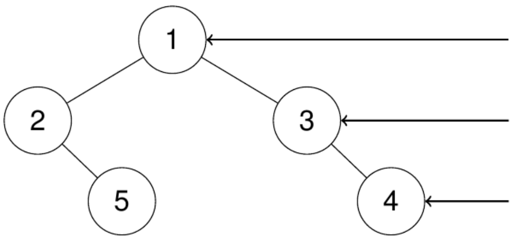
1. Binary Tree Right Side View

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:

<u>CODE:</u>

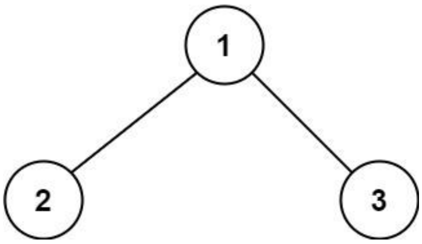
```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

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

vector<int> rightSideView(TreeNode* root) {
    vector<int> result;
    if (!root) return result;
```

```
    queue<TreeNode*> q;
    q.push(root);

    while (!q.empty()) {
        int levelSize = q.size();
        for (int i = 0; i < levelSize; i++) {
            TreeNode* current = q.front();
            q.pop();

            // Add the last node of the current level to the result
            if (i == levelSize - 1) result.push_back(current->val);

            if (current->left) q.push(current->left);
            if (current->right) q.push(current->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);

    vector<int> result = rightSideView(root);
    for (int val : result) {
        cout << val << " ";
    }
    // Output: 1 3 4
    return 0;
}
```

2. **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.

Example 1:



Input: root = [1,2,3]
Output: 6
Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.
<u>CODE:</u>

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

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

int maxPathSumHelper(TreeNode* root, int& maxSum) {
    if (!root) return 0;

    // Compute maximum path sums for left and right subtrees
    int leftMax = max(0, maxPathSumHelper(root->left, maxSum));
    int rightMax = max(0, maxPathSumHelper(root->right, maxSum));

    // Update the overall maximum path sum
    maxSum = max(maxSum, leftMax + rightMax + root->val);

    // Return the maximum path sum including the current node
    return max(leftMax, rightMax) + root->val;
}

int maxPathSum(TreeNode* root) {
    int maxSum = INT_MIN;
    maxPathSumHelper(root, maxSum);
    return maxSum;
}

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

    cout << maxPathSum(root) << endl; // Output: 6
```

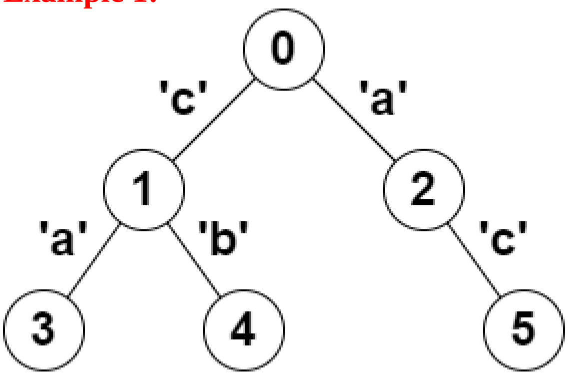
```
    return 0;
}
```

Very Hard:

1. 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.

Example 1:



Input: parent = [-1,0,0,1,1,2], s = "acaabc"
Output: 8
Explanation: The valid pairs are:
- All the pairs (0,1), (0,2), (1,3), (1,4) and (2,5) result in one character which is always a palindrome.
- The pair (2,3) result in the string "aca" which is a palindrome.
- The pair (1,5) result in the string "cac" which is a palindrome.
- The pair (3,5) result in the string "acac" which can be rearranged into the palindrome "acca".

<u>CODE:</u>

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

// Helper function to perform DFS and count palindromic paths
void dfs(int node, int mask, const vector<vector<int>>& graph, const string& s, unordered_map<int, int>& count, int& result) {
```

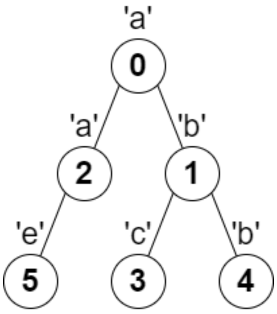
```
    // Update result based on the current mask
    result += count[mask];
    for (int i = 0; i < 26; ++i) {
        result += count[mask ^ (1 << i)];
    }

    // Increment the count for the current mask
    count[mask]++;

    // Recur for child nodes
    for (int child : graph[node]) {
        dfs(child, mask ^ (1 << (s[child] - 'a')), graph, s, count, result);
    }

    // Decrement the count to backtrack
    count[mask]--;
}

int countPalindromePaths(vector<int>& parent, string s) {
    int n = parent.size();
    vector<vector<int>> graph(n);
    for (int i = 1; i < n; ++i) {
        graph[parent[i]].push_back(i);
    }

    unordered_map<int, int> count;
    count[0] = 1; // Initial mask (no characters)
    int result = 0;
    dfs(0, 0, graph, s, count, result);
    return result;
}

int main() {
    vector<int> parent = {-1, 0, 0, 1, 1, 2};
    string s = "acaabc";
    cout << countPalindromePaths(parent, s) << endl; // Output: 8
    return 0;
}
```

```
8
```

2. 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.

Example 1:



Input: parent = [-1,0,0,1,1,2], s = "abacbe"
Output: 3
Explanation: The longest path where each two adjacent nodes have different characters in the tree is the path: 0 -> 1 -> 3. The length of this path is 3, so 3 is returned. It can be proven that there is no longer path that satisfies the conditions.

<u>CODE:</u>

```
from collections import defaultdict

def longestPath(parent, s):
    n = len(parent)
    adj = defaultdict(list)

    # Build the adjacency list of the tree
    for i in range(1, n):
        adj[parent[i]].append(i)

    # This will store the longest path starting from each node
    longest = [0] * n

    def dfs(node):
        first_max, second_max = 0, 0

        # Explore all the children of the current node
        for child in adj[node]:
            child_path = dfs(child)

            # Only consider the child's path if the characters are different
            if s[child] != s[node]:
                if child_path > first_max:
                    second_max = first_max
                    first_max = child_path
                elif child_path > second_max:
                    second_max = child_path

        # The longest path that passes through this node is the sum of first_max and second_max + 1
(for the current node itself)
        longest[node] = first_max + 1
```

```
        # Return the length of the longest path for the subtree rooted at this node
        return first_max + 1

    # Start DFS from the root (node 0)
    dfs(0)

    # The answer is the longest path in the tree
    return max(longest)

# Example usage
parent = [-1, 0, 0, 1, 1, 2]
s = "abacbe"
print(longestPath(parent, s))  # Output: 3
```