



Department of Computer Science and Engineering

Name:- Ripunjai Tiwari

UID:- 22BCS10216

Section:- 22KPIT-901-A

DAY-6

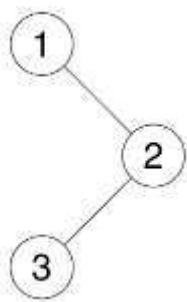
Que 1. Given the root of a binary tree, return the inorder traversal of its nodes' values.

Example 1:

Input: root = [1,null,2,3]

Output: [1,3,2]

Explanation:

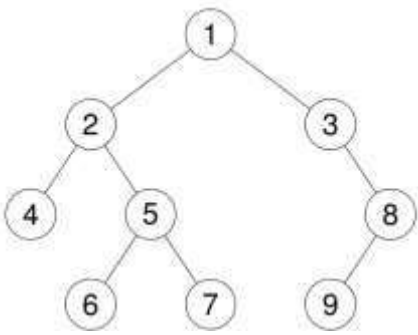


Example 2:

Input: root = [1,2,3,4,5,null,8,null,null,6,7,9]

Output: [4,2,6,5,7,1,3,9,8]

Explanation:



Constraints: The number of nodes in the tree is in the range $[0, 100]$.

$-100 \leq \text{Node.val} \leq 100$

- $1 \leq k \leq 10^6$

Program Code:-

```

#include <iostream>
#include <vector>
#include <stack>
using namespace std;
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) { }
};
vector<int> inorderTraversal(TreeNode* root) {
    vector<int> result;
    stack<TreeNode*> st;
    TreeNode* current = root;

    while (current || !st.empty()) {
        while (current) {
            st.push(current);
            current = current->left;
        }
        current = st.top();
    }
}

```

```

    st.pop();
    result.push_back(current->val);
    current = current->right;
}

return result;
}


int main() {
    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 << " ";
    }

    return 0;
}

```

Output:-



```

1 3 2
...Program finished with exit code 0
Press ENTER to exit console.

```

QUES2: 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

Example 2:

Input: p = [1,2], q = [1,null,2]

Output: false

Constraints:

The number of nodes in both trees is in the range [0, 100].

$-104 \leq \text{Node.val} \leq 104$

Program Code:-

```
#include <bits/stdc++.h>
using namespace std;

struct Node {
    int data;
    Node *left, *right;

    Node(int val) {
        data = val;
        left = right = nullptr;
    }
};

bool isIdentical(Node* r1, Node* r2) {
    if (r1 == nullptr && r2 == nullptr)
        return true;

    if (r1 == nullptr || r2 == nullptr)
        return false;

    queue<Node*> q1, q2;
    q1.push(r1);
    q2.push(r2);

    while (!q1.empty() && !q2.empty()) {
        Node* node1 = q1.front();
        Node* node2 = q2.front();
        q1.pop();
        q2.pop();
```

```

    if (node1->data != node2->data)
        return false;

    if (node1->left && node2->left) {
        q1.push(node1->left);
        q2.push(node2->left);
    } else if (node1->left || node2->left) {
        return false;
    }
    if (node1->right && node2->right) {
        q1.push(node1->right);
        q2.push(node2->right);
    } else if (node1->right || node2->right) {
        return false;
    }
}
return q1.empty() && q2.empty();
}

```

```

int main() {
    Node* r1 = new Node(1);
    r1->left = new Node(2);
    r1->right = new Node(3);
    r1->left->left = new Node(4);

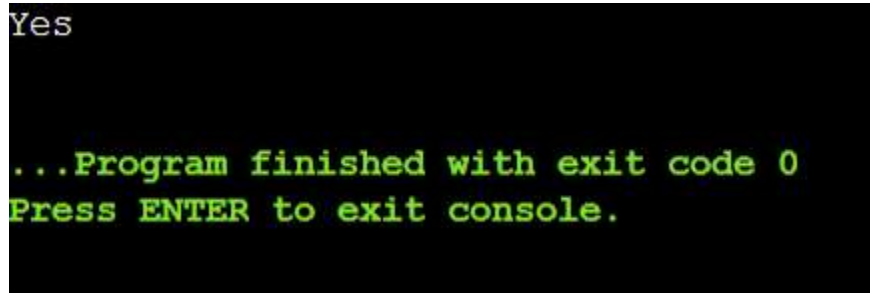
    Node* r2 = new Node(1);
    r2->left = new Node(2);
    r2->right = new Node(3);
    r2->left->left = new Node(4);

    if (isIdentical(r1, r2))
        cout << "Yes\n";
    else
        cout << "No\n";

    return 0;
}

```

Output:-



```
Yes

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

QUES 3: 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]

Example 2:

Input: preorder = [-1], inorder = [-1]

Output: [-1]

Constraints:

$1 \leq \text{preorder.length} \leq 3000$

$\text{inorder.length} == \text{preorder.length}$

$-3000 \leq \text{preorder}[i], \text{inorder}[i] \leq 3000$

preorder and inorder consist of unique values.

Each value of inorder also appears in preorder.

preorder is guaranteed to be the preorder traversal of the tree.

inorder is guaranteed to be the inorder traversal of the tree.

Program Code:-

```
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int data;
    Node *left, *right;
    Node(int x) {
        data = x;
        left = nullptr;
        right = nullptr;
    }
};

int searchValue(vector<int>& in, int value, int s, int e) {

    for (int i=s; i<=e; i++) {
        if (in[i] == value)
            return i;
    }
    return -1;
}

Node* buildTreeRecur(vector<int>& in, vector<int>& pre,
                    int &preIndex, int s, int e) {

    if (s > e) return nullptr;
    Node* root = new Node(pre[preIndex]);
    preIndex++;

    int index = searchValue(in, pre[preIndex-1], s, e);

    root->left = buildTreeRecur(in, pre, preIndex, s, index-1);
    root->right = buildTreeRecur(in, pre, preIndex, index+1, e);

    return root;
}

Node* buildTree(vector<int>& in, vector<int>& pre) {

    int n = pre.size();
    int preIndex = 0;
    Node* root = buildTreeRecur(in, pre, preIndex, 0, n-1);

    return root;
}
```

```

}
void printPostorder(Node* root) {
    if (root == nullptr)
        return;
    printPostorder(root->left);
    printPostorder(root->right);
    cout << root->data << " ";
}

//medium

int main() {
    vector<int> in = {3, 1, 4, 0, 5, 2};
    vector<int> pre = {0, 1, 3, 4, 2, 5};
    Node* root = buildTree(in, pre);

    printPostorder(root);

    return;
}

```

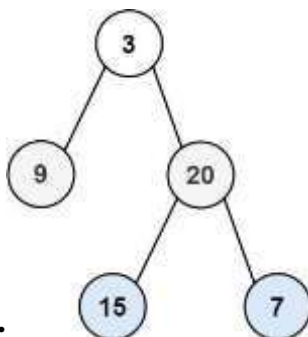
Output:-

```

3 4 1 5 2 0
...Program finished with exit code 0
Press ENTER to exit console.

```

QUES 4: Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).



Example 1:

Input: root = [3,9,20,null,null,15,7]

Output: [[3],[20,9],[15,7]]

Example 2:

Input: root = [1]

Output: [[1]]

Example 3:

Input: root = []

Output: []

Constraints:

The number of nodes in the tree is in the range [0, 2000].

-100 <= Node.val <= 100

Program Code :

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
class Node {
```

```
public:
```

```
    int data;
```

```
    Node *left;
```

```
    Node *right;
```

```
    Node(int x) {
```

```
        data = x;
```

```
        left = nullptr;
```

```
        right = nullptr;
```

```
    }
```

```
};
```

```
vector<int> zigZagTraversal(Node* root) {
```

```
    vector<int> ans;
```

```
    if (root == nullptr)
```

```
        return ans;
```

```
    stack<Node*> currentLevel;
```

```
    stack<Node*> nextLevel;
```

```
    currentLevel.push(root);
```

```

bool leftToRight = true;
while (!currentLevel.empty()) {

    int size = currentLevel.size();

    while (size-- > 0) {

        struct Node* curr = currentLevel.top();
        currentLevel.pop();

        ans.push_back(curr->data);

        if (leftToRight) {
            if (curr->left)
                nextLevel.push(curr->left);
            if (curr->right)
                nextLevel.push(curr->right);
        }
        else {
            if (curr->right)
                nextLevel.push(curr->right);
            if (curr->left)
                nextLevel.push(curr->left);
        }
    }

    leftToRight = !leftToRight;
    swap(currentLevel, nextLevel);
}

return ans;
}

void printList(vector<int> v) {
    int n = v.size();
    for (int i=0; i<n; i++) {
        cout << v[i] << " ";
    }
    cout << endl;
}

int main() {
    Node* root = new Node(20);
    root->left = new Node(8);
    root->right = new Node(22);
    root->right->right = new Node(11);
    root->left->left = new Node(4);
}

```

```

root->left->right = new Node(12);
root->left->right->left = new Node(10);
root->left->right->right = new Node(14);

vector<int> ans = zigZagTraversal(root);
printList(ans);

return 0;
}

```

Output :

```

20 22 8 4 12 11 14 10

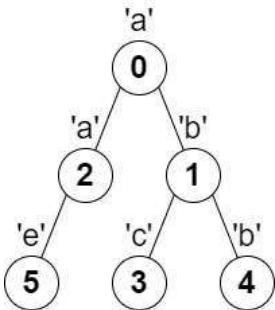
...Program finished with exit code 0
Press ENTER to exit console.

```

Que 5. 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.

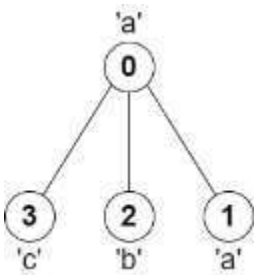
Example 1:



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.

Example 2:



Input: parent = [-1,0,0,0], s = "aabc" **Output:** 3**Explanation:** The longest path where each two adjacent nodes have different characters is the path: 2 -> 0 -> 3. The length of this path is 3, so 3 is returned.

Constraints:

$n == \text{parent.length} == s.\text{length}$

$1 \leq n \leq 105$

$0 \leq \text{parent}[i] \leq n - 1$ for all $i \geq 1$

$\text{parent}[0] == -1$

parent represents a valid tree.

Program Code :

```
#include <iostream>
```

```
#include <vector>
```

```
#include <string>
```

```
#include <algorithm>
```

```
using namespace std;
```

```
int dfs(int node, vector<vector<int>>& tree, string& s, int& maxPath) {
```

```
    int longest = 0, secondLongest = 0;
```

```
    for (int child : tree[node]) {
        int childPath = dfs(child, tree, s, maxPath);
```

```
        if (s[node] != s[child]) {
            if (childPath > longest) {
                secondLongest = longest;
                longest = childPath;
            } else if (childPath > secondLongest) {
                secondLongest = childPath;
            }
        }
    }
}
```

```
maxPath = max(maxPath, longest + secondLongest + 1);
```

```
return longest + 1;
```

```
}
```

```

int longestPath(vector<int>& parent, string s) {
    int n = parent.size();
    vector<vector<int>> tree(n);

    for (int i = 1; i < n; ++i) {
        tree[parent[i]].push_back(i);
    }

    int maxPath = 0;
    dfs(0, tree, s, maxPath);

    return maxPath;
}

int main() {
    vector<int> parent = {-1, 0, 0, 1, 1, 2};
    string s = "abacbe";

    cout << "Longest path length: " << longestPath(parent, s) << endl;

    return 0;
}

```

Output :

```

Longest path length: 3

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

```