

Name – Dependra Singh
Section – KPIT-901
UID – 22BCS10256

DSA Questions(Trees)

Very Easy:

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 2^h 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

Example 2:

Input: root = []

Output: 0

Example 3:

Input: root = [1]

Output: 1

Constraints:

The number of nodes in the tree is in the range $[0, 5 * 10^4]$.

$0 \leq \text{Node.val} \leq 5 * 10^4$

The tree is guaranteed to be complete.

Reference: <https://leetcode.com/problems/count-complete-tree-nodes/description/>

Code:-

```
#include <iostream>
#include <cmath>
#include <cstddef> // For nullptr
using namespace std;

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

class Solution {
public:
    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);
        }
    }

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

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);
```

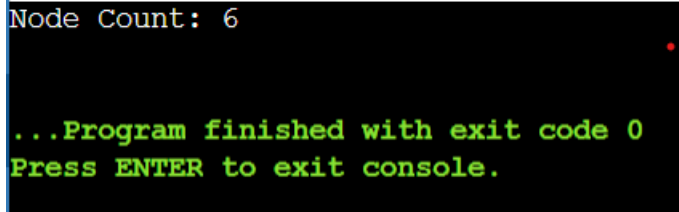
```

    Solution solution;
    cout << "Node Count: " << solution.countNodes(root) << endl;

    return 0;
}

```

Output:-



```

Node Count: 6

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

```

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

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 <= Node.val <= 104

Reference: <https://leetcode.com/problems/same-tree/description/?envType=study-plan-v2&envId=top-interview-150>

Code:-

```

#include <iostream>
#include <cstdint> // For nullptr
using namespace std;

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

```

```

};

class Solution {
public:
    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);

    Solution solution;
    cout << "Are Trees Same? " << (solution.isSameTree(p, q) ? "Yes" : "No") <<
endl; // Output: Yes

    return 0;
}

```

Output:-

```

Are Trees Same? Yes

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

```

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]

Example 2:

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

Output: [-1]

Constraints:

1 <= preorder.length <= 3000

inorder.length == preorder.length

-3000 <= preorder[i], inorder[i] <= 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.

Reference: <https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/description/?envType=study-plan-v2&envId=top-interview-150>

Code:-

```
#include <iostream>
#include <vector>
#include <cstdint> // For nullptr
using namespace std;

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

class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        return build(preorder, 0, preorder.size() - 1, inorder, 0, inorder.size() - 1);
    }

private:
    TreeNode* build(vector<int>& preorder, int preStart, int preEnd,
                    vector<int>& inorder, int inStart, int inEnd) {
        if (preStart > preEnd || inStart > inEnd) return nullptr;
```

```

    int rootVal = preorder[preStart];
    TreeNode* root = new TreeNode(rootVal);

    int inIndex = inStart;
    while (inorder[inIndex] != rootVal) ++inIndex;

    int leftSize = inIndex - inStart;

    root->left = build(preorder, preStart + 1, preStart + leftSize, inorder,
inStart, inIndex - 1);
    root->right = build(preorder, preStart + leftSize + 1, preEnd, inorder,
inIndex + 1, inEnd);

    return root;
}
};

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

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

    Solution solution;
    TreeNode* root = solution.buildTree(preorder, inorder);

    cout << "Constructed Tree (Preorder): ";
    printTree(root); // Output: 3 9 20 15 7

    return 0;
}

```

Output:-

```

Are Trees Same? Yes

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

```

Hard :

5. Kth Smallest Element in a BST (Binary Search Tree)

Given a binary search tree (BST), write a function to find the kth smallest element in the tree.

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

Output: 1

Explanation: The inorder traversal of the BST is [1, 2, 3, 4], and the 1st smallest element is 1.

Input: root = [5,3,6,2,4,null,null,1], k = 3

Output: 3

Explanation: The inorder traversal of the BST is [1, 2, 3, 4, 5, 6], and the 3rd smallest element is 3.

Constraints:

The number of nodes in the tree is in the range [1, 1000].

$-10^4 \leq \text{Node.val} \leq 10^4$.

Reference: <http://leetcode.com/problems/kth-smallest-element-in-a-bst/>

Code:-

```
#include <iostream>
#include <cstdint> // For nullptr
using namespace std;

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

class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        int count = 0, result = -1;
        inorder(root, k, count, result);
        return result;
    }
};
```

```

private:
    void inorder(TreeNode* node, int k, int& count, int& result) {
        if (!node) return;

        inorder(node->left, k, count, result);
        if (++count == k) {
            result = node->val;
            return;
        }
        inorder(node->right, k, count, result);
    }
};

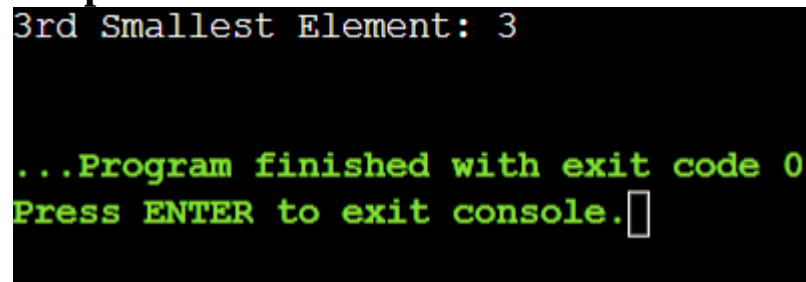
int main() {
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(1);
    root->right = new TreeNode(4);
    root->left->right = new TreeNode(2);

    Solution solution;
    cout << "3rd Smallest Element: " << solution.kthSmallest(root, 3) << endl; //
Output: 3

    return 0;
}

```

Output:-



```

3rd Smallest Element: 3

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

```

Very Hard :

1. Count Paths That Can Form a Palindrome in a Tree

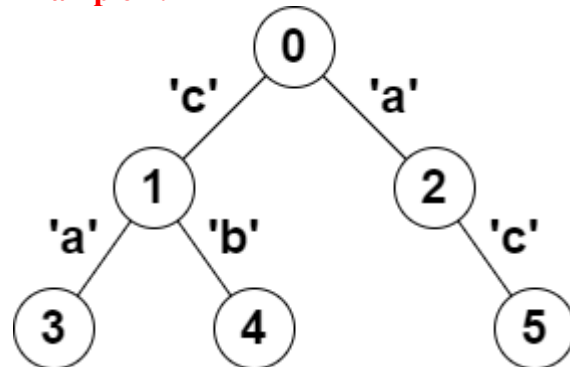
You are given a tree (i.e. a connected, undirected graph that has no cycles) rooted at node 0 consisting of n nodes numbered from 0 to $n - 1$. The tree is represented by a 0-indexed array `parent` of size n , where `parent[i]` is the parent of node i . Since node 0 is the root, `parent[0] == -1`.

You are also given a string s of length n , where $s[i]$ is the character assigned to the edge between i and $\text{parent}[i]$. $s[0]$ 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: $\text{parent} = [-1, 0, 0, 1, 1, 2]$, $s = \text{"acaabc"}$

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".

Example 2:

Input: $\text{parent} = [-1, 0, 0, 0, 0]$, $s = \text{"aaaaa"}$

Output: 10

Explanation: Any pair of nodes (u,v) where $u < v$ is valid.

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.

s consists of only lowercase English letters.

Reference : <https://leetcode.com/problems/count-paths-that-can-form-a-palindrome-in-a-tree/description/?envType=problem-list-v2&envId=tree>

Code:-

```

#include <iostream>
#include <vector>

```

```

#include <unordered_map>
using namespace std;

class Solution {
public:
    int countPalindromePaths(vector<int>& parent, string s) {
        unordered_map<int, int> maskCounts;
        vector<vector<int>>> tree(parent.size());
        for (int i = 1; i < parent.size(); ++i) {
            tree[parent[i]].push_back(i);
        }

        int count = 0;
        dfs(0, 0, s, tree, maskCounts, count);
        return count;
    }

private:
    void dfs(int node, int mask, string& s, vector<vector<int>>& tree,
            unordered_map<int, int> maskCounts, int& count) {
        count += maskCounts[mask];
        for (int i = 0; i < 26; ++i) {
            count += maskCounts[mask ^ (1 << i)];
        }
        ++maskCounts[mask];
        for (int child : tree[node]) {
            dfs(child, mask ^ (1 << (s[child] - 'a')), s, tree, maskCounts, count);
        }
        --maskCounts[mask];
    }
};

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

    Solution solution;
    cout << "Palindrome Paths Count: " << solution.countPalindromePaths(parent, s)
    << endl; // Output: 8

    return 0;
}

```

Output:-

```
Palindrome Paths Count: 5
```

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