

DOMAIN WINTER CAMP

(Department of Computer Science and Engineering)

Name: Aman Bansal UID: 22BCS13365 Section: KPIT 901-B

DAY-6

Q 1. Binary Tree Inorder Traversal

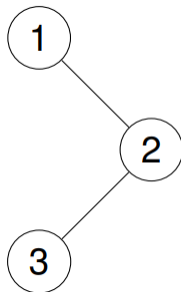
Given the root of a binary tree, return the inorder traversal of its nodes' values.

Example 1:

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

Output: [1,3,2]

Explanation:

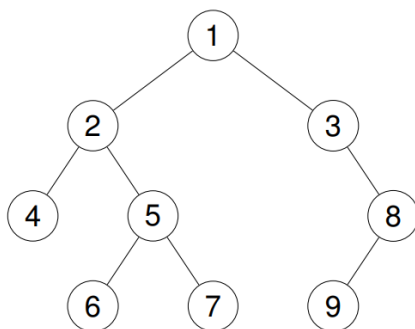


Example 2:

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].

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

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

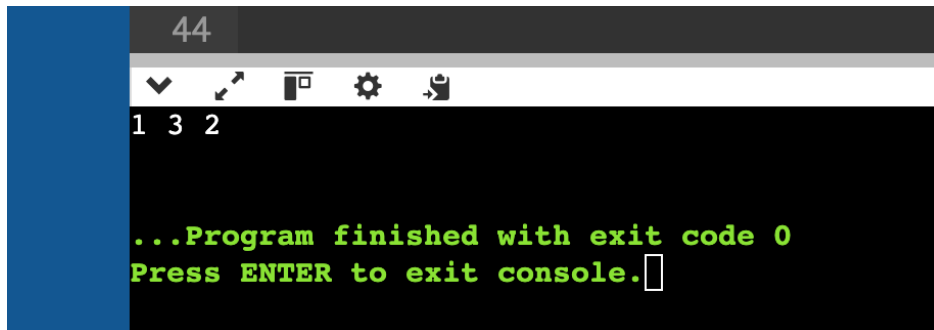
    vector<int> result = inorderTraversal(root);

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

```
// Free allocated memory
delete root->right->left;
delete root->right;
delete root;

return 0;
}
```

Output:



```
44
1 3 2
...Program finished with exit code 0
Press ENTER to exit console.
```

Q 2 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

Program code:

```
#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;
    if (!p || !q) return false;
    if (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;
```

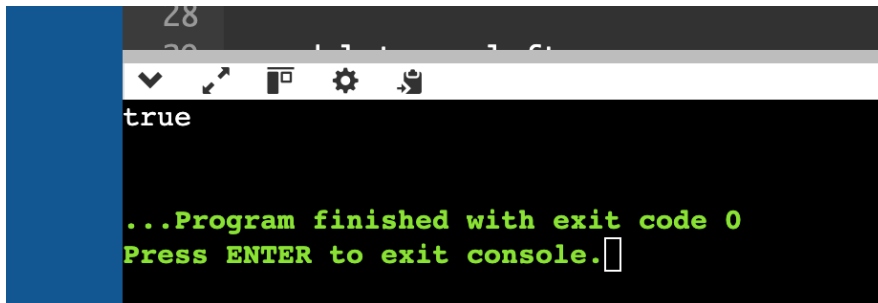
```

delete p->left;
delete p->right;
delete p;
delete q->left;
delete q->right;
delete q;

return 0;
}

```

Output:



```

true

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

```

Q 3. 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.

Program Code:

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

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

unordered_map<int, int> inorderMap;
int preorderIndex = 0;

TreeNode* buildTreeHelper(vector<int>& preorder, vector<int>& inorder, int left,
int right) {
    if (left > right) return nullptr;
    int rootValue = preorder[preorderIndex++];
    TreeNode* root = new TreeNode(rootValue);
    int inorderIndex = inorderMap[rootValue];
    root->left = buildTreeHelper(preorder, inorder, left, inorderIndex - 1);
    root->right = buildTreeHelper(preorder, inorder, inorderIndex + 1, right);
    return root;
}

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

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};

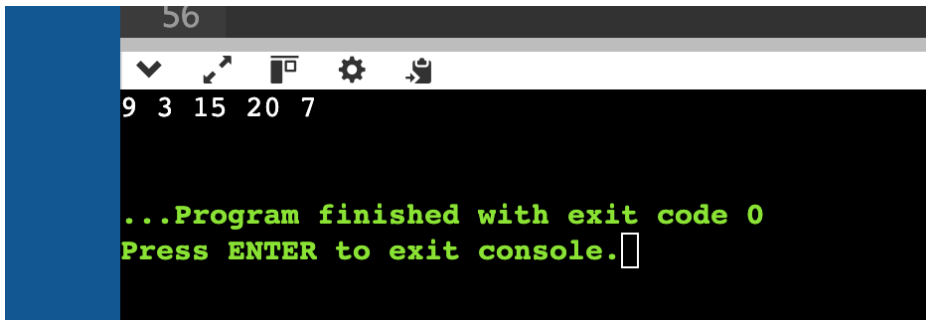
    TreeNode* root = buildTree(preorder, inorder);
    printInOrder(root);
    cout << endl;

    delete root->left;
    delete root->right->left;
    delete root->right->right;
    delete root->right;
    delete root;

    return 0;
}

```

Output:



```

56
9 3 15 20 7

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

```

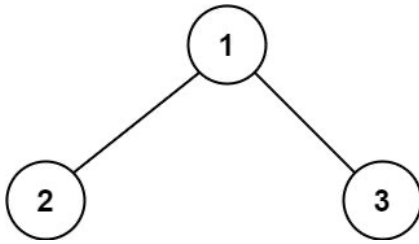
Q 4. Binary Tree Maximum Path Sum

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:

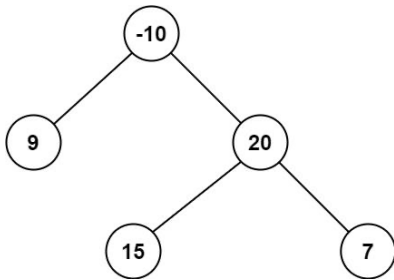


Input: root = [1,2,3]

Output: 6

Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of $2 + 1 + 3 = 6$.

Example 2:



Input: root = [-10,9,20,null,null,15,7]

Output: 42

Explanation: The optimal path is 15 -> 20 -> 7 with a path sum of $15 + 20 + 7 = 42$.

Constraints:

The number of nodes in the tree is in the range [1, $3 * 10^4$].

$-1000 \leq \text{Node.val} \leq 1000$

Program Code:

```

#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 maxSum = INT_MIN;

int findMaxPathSum(TreeNode* root) {
  
```



```

    if (!root) return 0;
    int left = max(0, findMaxPathSum(root->left));
    int right = max(0, findMaxPathSum(root->right));
    maxSum = max(maxSum, left + right + root->val);
    return root->val + max(left, right);
}

int maxPathSum(TreeNode* root) {
    findMaxPathSum(root);
    return maxSum;
}

int main() {
    TreeNode* root = new TreeNode(-10);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);

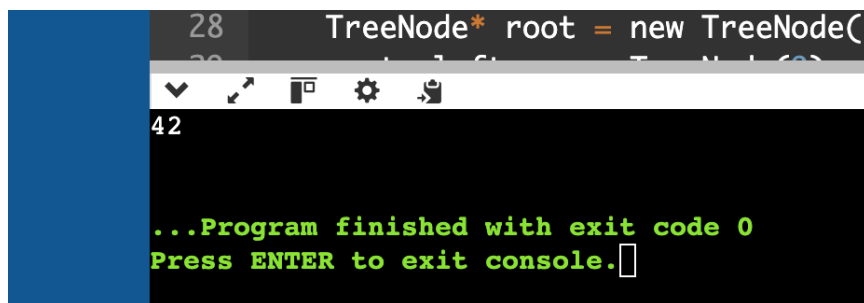
    cout << maxPathSum(root) << endl;

    delete root->right->left;
    delete root->right->right;
    delete root->right;
    delete root->left;
    delete root;

    return 0;
}

```

Output:



```

28      TreeNode* root = new TreeNode(-10);
29      root->left = new TreeNode(9);
30      root->right = new TreeNode(20);
31      root->right->left = new TreeNode(15);
32      root->right->right = new TreeNode(7);
33
34      cout << maxPathSum(root) << endl;
35
36      delete root->right->left;
37      delete root->right->right;
38      delete root->right;
39      delete root->left;
40      delete root;
41
42      return 0;
43  }
44
45  ...Program finished with exit code 0
46  Press ENTER to exit console.

```

Q 5 Count Number of Possible Root Nodes

Alice has an undirected tree with n nodes labeled from 0 to $n - 1$. The tree is represented as a 2D integer array `edges` of length $n - 1$ where `edges[i] = [ai, bi]` indicates that there is an edge between nodes a_i and b_i in the tree.

Alice wants Bob to find the root of the tree. She allows Bob to make several guesses about her tree. In one guess, he does the following:

Chooses two distinct integers u and v such that there exists an edge $[u, v]$ in the tree.

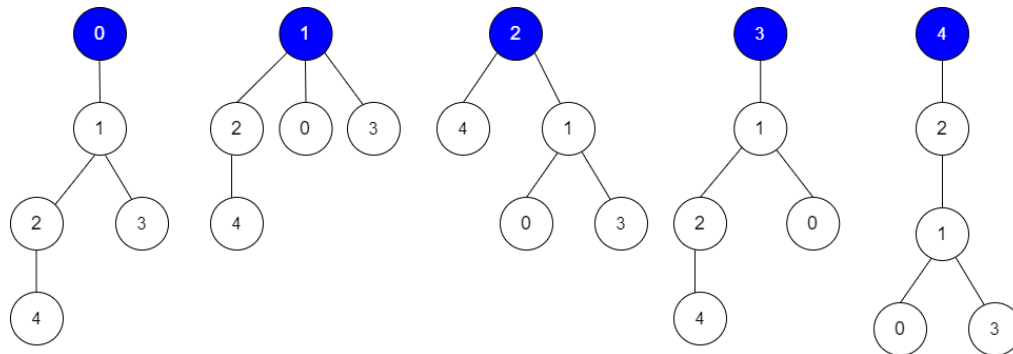
He tells Alice that u is the parent of v in the tree.

Bob's guesses are represented by a 2D integer array `guesses` where `guesses[j] = [uj, vj]` indicates Bob guessed u_j to be the parent of v_j .

Alice being lazy, does not reply to each of Bob's guesses, but just says that at least k of his guesses are true.

Given the 2D integer arrays `edges`, `guesses` and the integer k , return the number of possible nodes that can be the root of Alice's tree. If there is no such tree, return 0.

Example 1:



Input: `edges = [[0,1],[1,2],[1,3],[4,2]]`, `guesses = [[1,3],[0,1],[1,0],[2,4]]`, $k = 3$ **Output:**

3**Explanation:**

Root = 0, correct guesses = [1,3], [0,1], [2,4]

Root = 1, correct guesses = [1,3], [1,0], [2,4]

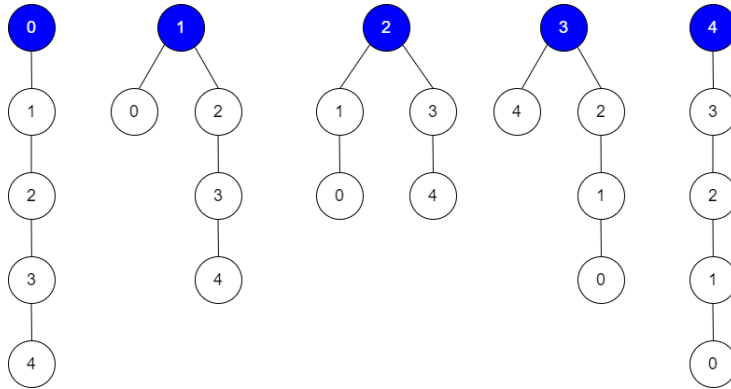
Root = 2, correct guesses = [1,3], [1,0], [2,4]

Root = 3, correct guesses = [1,0], [2,4]

Root = 4, correct guesses = [1,3], [1,0]

Considering 0, 1, or 2 as root node leads to 3 correct guesses.

Example 2:



Input: edges = [[0,1],[1,2],[2,3],[3,4]], guesses = [[1,0],[3,4],[2,1],[3,2]], k = 1 **Output:**

5Explanation:

Root = 0, correct guesses = [3,4]

Root = 1, correct guesses = [1,0], [3,4]

Root = 2, correct guesses = [1,0], [2,1], [3,4]

Root = 3, correct guesses = [1,0], [2,1], [3,2], [3,4]

Root = 4, correct guesses = [1,0], [2,1], [3,2]

Considering any node as root will give at least 1 correct guess.

Constraints:

edges.length == n - 1

2 <= n <= 105

1 <= guesses.length <= 105

0 <= ai, bi, uj, vj <= n - 1

ai != bi

uj != vj

edges represents a valid tree.

guesses[j] is an edge of the tree.

guesses is unique.

0 <= k <= guesses.length

Program Code:

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

vector<vector<int>> tree;
unordered_set<string> guessSet;
int result = 0;

int countCorrectGuesses(int node, int parent) {
    int count = 0;
```

```

    for (int neighbor : tree[node]) {
        if (neighbor == parent) continue;
        if (guessSet.count(to_string(node) + "," + to_string(neighbor))) {
            count++;
        }
        count += countCorrectGuesses(neighbor, node);
    }
    return count;
}

void dfs(int node, int parent, int correctGuesses, int k) {
    if (correctGuesses >= k) result++;
    for (int neighbor : tree[node]) {
        if (neighbor == parent) continue;
        int nextCorrectGuesses = correctGuesses;
        if (guessSet.count(to_string(node) + "," + to_string(neighbor)))
            nextCorrectGuesses--;
        if (guessSet.count(to_string(neighbor) + "," + to_string(node)))
            nextCorrectGuesses++;
        dfs(neighbor, node, nextCorrectGuesses, k);
    }
}

int countPossibleRoots(vector<vector<int>>& edges, vector<vector<int>>& guesses,
int k) {
    int n = edges.size() + 1;
    tree.assign(n, vector<int>());
    for (auto& edge : edges) {
        tree[edge[0]].push_back(edge[1]);
        tree[edge[1]].push_back(edge[0]);
    }
    for (auto& guess : guesses) {
        guessSet.insert(to_string(guess[0]) + "," + to_string(guess[1]));
    }
    int initialCorrectGuesses = countCorrectGuesses(0, -1);
    dfs(0, -1, initialCorrectGuesses, k);
    return result;
}

int main() {
    vector<vector<int>> edges = {{0,1},{1,2},{1,3},{4,2}};
    vector<vector<int>> guesses = {{1,3},{0,1},{1,0},{2,4}};

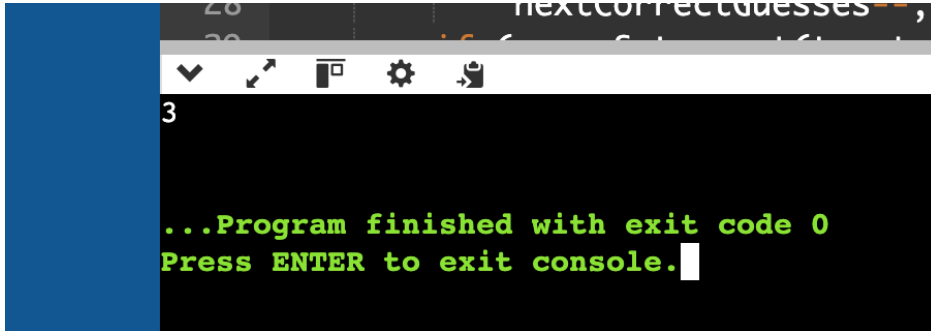
```

```
int k = 3;

cout << countPossibleRoots(edges, guesses, k) << endl;

return 0;
}
```

Output:



```
3

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