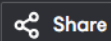


main.cpp



Run

Output

```
1 #include <iostream>
2 #include <vector>
3 #include <stack>
4 using namespace std;
5 struct TreeNode {
6     int val;
7     TreeNode* left;
8     TreeNode* right;
9     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
10 };
11 void inorderRecursive(TreeNode* root, vector<int>& result) {
12     if (root) {
13         inorderRecursive(root->left, result);
14         result.push_back(root->val);
15         inorderRecursive(root->right, result);
16     }
17 }
18
19 vector<int> inorderTraversalRecursive(TreeNode* root) {
20     vector<int> result;
21     inorderRecursive(root, result);
22     return result;
23 }
24 vector<int> inorderTraversalIterative(TreeNode* root) {
25     vector<int> result;
26     stack<TreeNode*> s;
27     TreeNode* current = root;
28
29     while (current || !s.empty()) {
30         while (current) {
```

Recursive Inorder Traversal: 1 3 2
Iterative Inorder Traversal: 1 3 2

=== Code Execution Successful ===

main.cpp



Share

Run

Output

```
1  #include <iostream>
2  using namespace std;
3  struct TreeNode {
4      int val;
5      TreeNode* left;
6      TreeNode* right;
7      TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
8  };
9  bool isSameTree(TreeNode* p, TreeNode* q) {
10     if (!p && !q) return true;
11     if (!p || !q || p->val != q->val) return false;
12     return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
13 }
14 int main() {
15     TreeNode* p = new TreeNode(1);
16     p->left = new TreeNode(2);
17     p->right = new TreeNode(3);
18
19     TreeNode* q = new TreeNode(1);
20     q->left = new TreeNode(2);
21     q->right = new TreeNode(3);
22
23     if (isSameTree(p, q)) {
24         cout << "Output: true" << endl;
25     } else {
26         cout << "Output: false" << endl;
27     }
28
29     return 0;
30 }
```

Output: true

=== Code Execution Successful ===

main.cpp



Share

Run

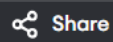
Output

```
1 #include <iostream>
2 #include <unordered_map>
3 #include <vector>
4 using namespace std;
5 struct TreeNode {
6     int val;
7     TreeNode* left;
8     TreeNode* right;
9     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
10 };
11 TreeNode* buildTreeHelper(vector<int>& preorder, int preStart, int preEnd,
12                           vector<int>& inorder, int inStart, int inEnd,
13                           unordered_map<int, int>& inMap) {
14     if (preStart > preEnd || inStart > inEnd) return nullptr;
15     TreeNode* root = new TreeNode(preorder[preStart]);
16     int inRoot = inMap[root->val];
17     int numsLeft = inRoot - inStart;
18     root->left = buildTreeHelper(preorder, preStart + 1, preStart + numsLeft,
19                                inorder, inStart, inRoot - 1, inMap);
20     root->right = buildTreeHelper(preorder, preStart + numsLeft + 1, preEnd,
21                                 inorder, inRoot + 1, inEnd, inMap);
22
23     return root;
24 }
25 TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
26     unordered_map<int, int> inMap;
27     for (int i = 0; i < inorder.size(); i++) {
28         inMap[inorder[i]] = i;
29     }
30 }
```

Output: 3 9 20 null null 15 7 null null null null

=== Code Execution Successful ===

main.cpp



Run

Output

```
1 #include <iostream>
2 using namespace std;
3 struct Node {
4     int val;
5     Node* left;
6     Node* right;
7     Node* next;
8     Node(int x) : val(x), left(nullptr), right(nullptr), next(nullptr) {}
9 };
10 Node* connect(Node* root) {
11     if (!root) return nullptr;
12
13     Node* leftmost = root;
14     while (leftmost->left) {
15         Node* head = leftmost;
16
17         while (head) {
18             head->left->next = head->right;
19             if (head->next) {
20                 head->right->next = head->next->left;
21             }
22             head = head->next;
23         }
24         leftmost = leftmost->left;
25     }
26
27     return root;
28 }
29 void printTreeWithNext(Node* root) {
30     Node* level = root;
```

Tree with next pointers:

1 -> NULL

2 -> 3 3 -> NULL

4 -> 5 5 -> 6 6 -> 7 7 -> NULL

=== Code Execution Successful ===

main.cpp



Share

Run

Output

```
1 #include <iostream>
2 #include <vector>
3 #include <unordered_map>
4 using namespace std;
5
6 void dfs(int node, int mask, vector<vector<int>>& tree, string& s,
    unordered_map<int, int>& maskCount, long long& result) {
7     result += maskCount[mask];
8     for (int i = 0; i < 26; i++) {
9         result += maskCount[mask ^ (1 << i)];
10    }
11    maskCount[mask]++;
12    for (int child : tree[node]) {
13        dfs(child, mask ^ (1 << (s[child] - 'a')), tree, s, maskCount, result
            );
14    }
15    maskCount[mask]--;
16 }
17
18 long long countPalindromePaths(vector<int>& parent, string s) {
19     int n = parent.size();
20     vector<vector<int>> tree(n);
21     for (int i = 1; i < n; i++) {
22         tree[parent[i]].push_back(i);
23     }
24
25     unordered_map<int, int> maskCount;
26     maskCount[0] = 1;
27
28     long long result = 0;
```

Number of palindrome paths: 11

=== Code Execution Successful ===