main.cpp

Share Run

```cpp
#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) {}
};
void inorderRecursive(TreeNode* root, vector<int>& result) {
    if (root) {
        inorderRecursive(root->left, result);
        result.push_back(root->val);
        inorderRecursive(root->right, result);
    }
}

vector<int> inorderTraversalRecursive(TreeNode* root) {
    vector<int> result;
    inorderRecursive(root, result);
    return result;
}
vector<int> inorderTraversalIterative(TreeNode* root) {
    vector<int> result;
    stack<TreeNode*> s;
    TreeNode* current = root;

    while (current || !s.empty()) {
        while (current) {
```

Output

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


=== Code Execution Successful ===
```

main.cpp

Share Run

Output

```cpp
#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 || 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);

    if (isSameTree(p, q)) {
        cout << "Output: true" << endl;
    } else {
        cout << "Output: false" << endl;
    }

    return 0;
}
```

```
Output: true


=== Code Execution Successful ===
```

main.cpp

Share Run

```cpp
#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;
    }

```

Output

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


=== Code Execution Successful ===
```

main.cpp

Share Run

```cpp
#include <iostream>
using namespace std;
struct Node {
    int val;
    Node* left;
    Node* right;
    Node* next;
    Node(int x) : val(x), left(nullptr), right(nullptr), next(nullptr) {}
};
Node* connect(Node* root) {
    if (!root) return nullptr;

    Node* leftmost = root;
    while (leftmost->left) {
        Node* head = leftmost;

        while (head) {
            head->left->next = head->right;
            if (head->next) {
                head->right->next = head->next->left;
            }
            head = head->next;
        }
        leftmost = leftmost->left;
    }

    return root;
}
void printTreeWithNext(Node* root) {
    Node* level = root;
```

Output

```
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

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

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

long long countPalindromePaths(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);
    }

    unordered_map<int, int> maskCount;
    maskCount[0] = 1;

    long long result = 0;
```

```
Number of palindrome paths: 11


=== Code Execution Successful ===
```