🌳 Module 12: Trees

---

1. Tree – Terminology

- **Tree**: Hierarchical data structure with nodes.
- **Root**: Top-most node.
- **Parent/Child**: Nodes connected directly.
- **Leaf**: Node with no children.
- **Subtree**: Tree inside another tree.
- **Height**: Longest path from root to leaf.
- **Depth**: Distance from root to a node.

👉 **C++ Node structure:**

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

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

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

---

2. Height vs Nodes Formulas

- **For a binary tree of height h**:

- Minimum nodes = h + 1
- Maximum nodes = 2^(h+1) - 1

- **For a binary tree with n nodes**:
  - Minimum height = ⌈log2(n+1)⌉ - 1
  - Maximum height = n - 1 (like a linked list)

👉 ***C++ code to find height:***

```
int height(Node* root) {
   if (root == NULL) {
      return -1; // height of empty tree = -1
   }
   int leftH = height(root->left);
   int rightH = height(root->right);
   return max(leftH, rightH) + 1;
}
```

---

3. Internal and External Nodes

- **Internal node**: Node with at least 1 child.
- **External node (leaf)**: Node with no children.

👉 ***C++ code to count:***

```
int countLeaves(Node* root) {
   if (root == NULL) return 0;
   if (root->left == NULL && root->right == NULL) return 1;
   return countLeaves(root->left) + countLeaves(root->right);
}

int countInternal(Node* root) {
   if (root == NULL || (root->left == NULL && root->right == NULL))
      return 0;
   return 1 + countInternal(root->left) + countInternal(root->right);
}
```

---

4. Strict Binary Tree

- Each node has **0 or 2 children only**.

👉 **Check strict binary tree:**

```
bool isStrict(Node* root) {
    if (root == NULL) return true;
    if ((root->left == NULL && root->right != NULL) ||
        (root->left != NULL && root->right == NULL))
        return false;
    return isStrict(root->left) && isStrict(root->right);
}
```

---

5. n-ary Trees

- A tree where each node can have **n children** (not just 2).

👉 Simple structure:

```
#include <vector>
struct NNode {
    int data;
    vector<NNode*> children;
    NNode(int val) {
        data = val;
    }
};
```

---

6. Representation of Binary Tree

- **Pointer representation** (linked nodes).
- **Array representation**:
    - Root at index 0.
    - Left child at `2*i+1`, right child at `2*i+2`.

---

7. Full vs Complete Binary Tree

- **Full BT**: Every node has 0 or 2 children.
- **Complete BT**: All levels filled except possibly last, and last level filled left to right.

---

8. Strict vs Complete Binary Tree

- **Strict**: 0 or 2 children.
- **Complete**: Balanced filling, last level left to right.

---

9. Creating a Tree

👉 Manually create tree:

```
Node* root = new Node(1);
root->left = new Node(2);
root->right = new Node(3);
root->left->left = new Node(4);
root->left->right = new Node(5);
```

---

10. Creating Binary Tree DEMO

(Same as above – construct manually or dynamically from input).

---

11. Binary Tree Traversals

- **Inorder**: Left → Root → Right
- **Preorder**: Root → Left → Right
- **Postorder**: Left → Right → Root

👉 **Code:**

```
void inorder(Node* root) {
```

```cpp
    if (root == NULL) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

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

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

---

12. Iterative Traversals DEMO

👉 Inorder using stack:

```cpp
#include <stack>
void inorderIterative(Node* root) {
    stack<Node*> st;
    Node* curr = root;
    while (curr != NULL || !st.empty()) {
        while (curr != NULL) {
            st.push(curr);
            curr = curr->left;
        }
        curr = st.top();
        st.pop();
        cout << curr->data << " ";
        curr = curr->right;
    }
}
```

---

13. Level Order Traversal DEMO

👉 BFS using queue:

```
#include <queue>
void levelOrder(Node* root) {
    if (root == NULL) return;
    queue<Node*> q;
    q.push(root);
    while (!q.empty()) {
        Node* curr = q.front();
        q.pop();
        cout << curr->data << " ";
        if (curr->left != NULL) q.push(curr->left);
        if (curr->right != NULL) q.push(curr->right);
    }
}
```

---

14. Generate Binary Tree from Traversals

👉 Using Inorder + Preorder. (Classic problem).

```
int search(int inorder[], int start, int end, int value) {
    for (int i = start; i <= end; i++) {
        if (inorder[i] == value) return i;
    }
    return -1;
}

Node* buildTree(int inorder[], int preorder[], int inStart, int inEnd, int &preIndex) {
    if (inStart > inEnd) return NULL;

    Node* root = new Node(preorder[preIndex++]);
    if (inStart == inEnd) return root;

    int inIndex = search(inorder, inStart, inEnd, root->data);

    root->left = buildTree(inorder, preorder, inStart, inIndex - 1, preIndex);
    root->right = buildTree(inorder, preorder, inIndex + 1, inEnd, preIndex);

    return root;
}
```

---

15. Generate BT from Traversals DEMO

(Same as above, but with user input/output).

---

16. Height and Count of BT DEMO

👉 Count nodes:

```
int countNodes(Node* root) {
    if (root == NULL) return 0;
    return 1 + countNodes(root->left) + countNodes(root->right);
}
```

👉 Already wrote height function earlier.

---

17. Count Leaf Nodes in BT DEMO

👉 Already wrote in **3rd topic**:

```
int countLeaves(Node* root) {
    if (root == NULL) return 0;
    if (root->left == NULL && root->right == NULL) return 1;
    return countLeaves(root->left) + countLeaves(root->right);
}
```