A (rooted) tree is defined recursively:

- A tree is either empty or consists of a root node with one or more children, each of which is a tree.

**Tree Terminology**
- **Root node**: The topmost node in a tree.
- **Node**: Any tree element that may have zero or more child nodes.
- **Edge**: The connections between two nodes.
- **Internal nodes**: Nodes with children.
- **External nodes / Leaves**: Nodes with no children.

**More Tree Terminology**
- **Child**: A node below another in the tree structure.
- **Parent**: A node above another in the tree structure.
- **Descendant**: A node that is a child of another node.
- **Ancestor**: A node that has a child node.
- **Siblings**: Nodes that have the same parent.
- **Subtree**: A tree consisting of a node and all its descendants.

**Binary Trees**

A binary tree is defined recursively:

- A binary tree is either empty or consists of a root node with a left child and a right child, each of which is a binary tree.

**Depth**
- The depth of a node is the number of edges between it and the root node.

**Height**
- The height of a tree is the maximum depth of any node; this tree has height 3.
Binary Trees

A binary tree

Height of a Binary Tree

What is the min height?
What is the max height?

Minimum Height of a Binary Tree

If we pack the maximum number of nodes into a binary tree of height $k$, then we have

$$1 + 2 + 4 + \ldots + 2^k = 2^{k+1} - 1$$

nodes, which means...

Minimum Height of a Binary Tree

... the minimum height of a binary tree with $n$ nodes is $O(\log_2 n)$.

Implementing the Binary Tree

Just follow the recursive definition to get a simple implementation:

```cpp
template <class T>
class binary_tree_node {
    public:
        T data;
        binary_tree_node<T>* left;
        binary_tree_node<T>* right;
};
```

Implementing the Binary Tree

- For now, we'll just implement a tree as nodes
- Tree functions will be free functions
- Can also encapsulate specific kinds of binary trees as classes/class templates
  - Many operations (e.g. insert/delete) vary depending on use, so no one "binary tree implementation"
Binary Tree Traversals

- A traversal of a tree is the act of visiting every node in the tree once.
- There are three traversal orders:
  - Pre-order
  - In-order
  - Post-order

Pre-Order Traversal

Visit the root first, then the left and right sub-trees recursively:

In-Order Traversal

Visit the left sub-tree, the root, and then the right sub-tree:

Post-Order Traversal

Visit the left and right sub-trees first and the root last:

Pre-Order Traversal Implementation

Note naturally recursive description: visit the root first, then the left and right sub-trees.

So we get a naturally recursive implementation:

```cpp
template <class T>
void do_preorder(binary_tree_node<T>* root) {
    if (root != NULL) {
        // do something with root->data
        do_preorder(root->left);
        do_preorder(root->right);
    }
}
```

Other Implementations

Can you write the in-order and post-order traversal code?
Some Traversal Applications

- Print all nodes (in a particular order):
  
  ```cpp
  template <class T>
  void print_preorder(binary_tree_node<T>* root) {
    if (root != NULL) {
      cout << root->data << " ";
      print_preorder(root->left);
      print_preorder(root->right);
    }
  }
  ```

- Count nodes:
  ```cpp
  template <class T>
  int count(binary_tree_node<T>* root) {
    if (root == NULL) return 0;
    return 1 + count(root->left) + count(root->right);
  }
  ```

Tree Applications

- Decision trees
  - A kind of structure used in AI
  - See next project – Animal (20 Questions)
- Sets/Maps
  - Using Binary Search Trees (next lecture)
  - Compression/encoding (Huffman encoding)
  - Organizing high-dimensional spaces (k-d trees)
- Spelling dictionary (Tries)
- Many more...

Up Next

- Wednesday, April 10
  - Binary search trees
  - Project 4 due
- Friday, April 12 – NO LAB (E-DAYS)
- Monday, April 15
  - Midterm review
  - Lab 11 due
- Wednesday, April 17
  - Midterm 2 (in class)