**CSCI 262**

**Data Structures**

20 – Binary Trees

---

**Trees**

A (rooted) tree is defined recursively:

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

---

**Tree Terminology**

- **node**
- **edge**
- **root node**
- **internal nodes** (nodes with children)
- **external nodes / leaves**

---

**More Tree Terminology**

- **a** is a parent of **e**, **f**, and **g**.
- **c** is an ancestor of **e**, **f**, **g**, **h**, and **i**.
- **h** is a child of **e**.
- **e**, **f**, and **g** are siblings.

---

**More Tree Terminology**

- The depth of a node is the number of edges between it and the root node.
- The height of a tree is the maximum depth of any node; this tree has height 3.

---

**Binary Trees**

A binary tree is defined recursively:

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

$$1 + 2 + 4 + ... + 2^k = 2^{k+1} - 1 \text{ nodes, which means...}$$

*This is sometimes called a full tree.

**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
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:

```c++
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?
Traversals 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 project 5 – 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 11
  - Binary search trees

- Friday, April 13
  - E-DAYS – NO CLASS

- Monday, April 16
  - Midterm review
  - APT 5 due

- Wednesday, April 18
  - Midterm 2 (in class)