CSCI 262
Data Structures

19 – Binary Trees

CS@Mines

Trees

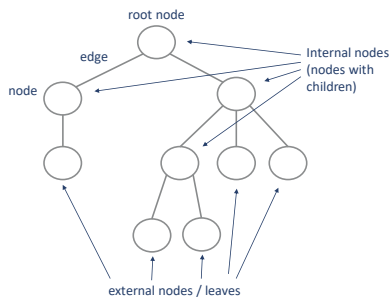
A (rooted) tree is defined recursively:

a tree is (empty)

a *root node* with one or more children, each of which is a tree.

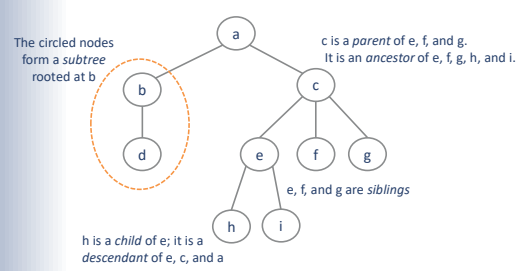
CS@Mines

Tree Terminology



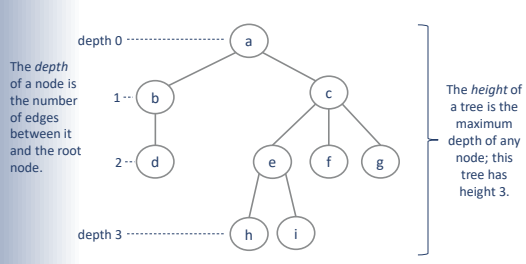
CS@Mines

More Tree Terminology



CS@Mines

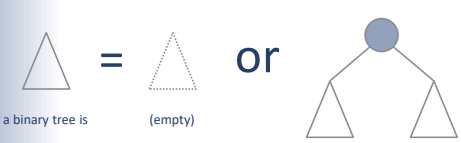
More Tree Terminology



CS@Mines

Binary Trees

A binary tree is defined recursively:

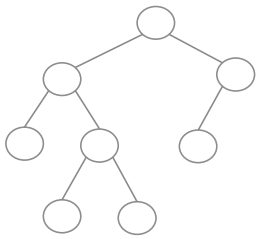


a binary tree is (empty)

a *root node* with a *left* child and a *right* child, each of which is a binary tree.

CS@Mines

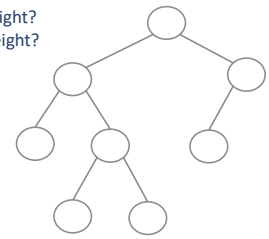
Binary Trees



A binary tree

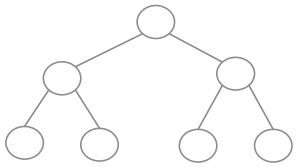
Height of a Binary Tree

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*

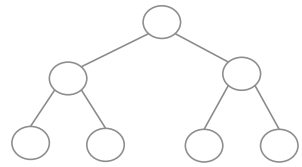


*This is sometimes called a *full* tree.

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

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


Implementing the Binary Tree

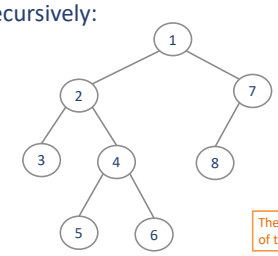
- For now, we'll just implement a tree as nodes
- Tree functions will be free functions
- Can also encapsulate specific kinds of binary trees as classes/class templates
  - Many operations (e.g. insert/delete) vary depending on use, so no one "binary tree implementation"

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

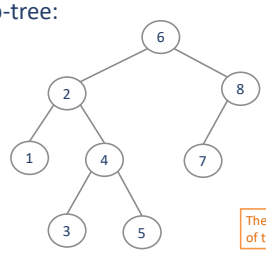
Visit the root first, then the left and right sub-trees recursively:



The numbers give the order of the visited nodes.

In-Order Traversal

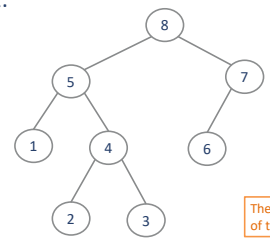
Visit the left sub-tree, the root, and then the right sub-tree:



The numbers give the order of the visited nodes.

Post-Order Traversal

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



The numbers give the order of the visited nodes.

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:

```
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):

```
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:

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

CS@Mines

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…

CS@Mines

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)

CS@Mines