Chapter 10 - Trees

MAT 102 - Data Structures
Wednesday, April 9, 2003
Binary Trees

• A binary tree has nodes which each have at most two children (left and right).
• The data stored in a node can be any data type, an integer, a string, or any other class.
• A node is called a “leaf” if it has no children.
• The top node is the “root” node and nodes between the root and the leaf nodes are called “internal nodes”.
A Binary Tree

Level 1

Level 2

Level 3

Level 4
A Binary Search Tree

- A binary search tree is a binary tree with comparison operations on the items.

- It is ordered so that the value of a node is greater than all values in its left subtree and less than all values in its right subtree.

- Also, both the left and right subtrees are similarly ordered so that they are both binary search trees.
Terms

• *height* - the number of levels in the tree, also the number of nodes on the longest path from the root to a leaf
• *full* - a full binary tree is one where every non-leaf node has two children and all the leaves are at the bottom level
• *complete* - a complete binary tree of height “h” is one that is full down to level (h-1) and level h is filled from left to right.
• *balanced* - a binary tree is balanced if the height of any node’s right subtree is differs from the height of the node’s left subtree by at most one.
A full binary tree (height = 4)
A complete binary tree
A balanced binary tree
Binary Tree Operations

- Create an empty binary tree
- Create a one-node binary tree, given an item
- Create a binary tree, given an item and two binary trees for the root’s subtrees
- Destroy a binary tree
- Determine whether a binary tree is empty
- Determine or change the data in the binary tree’s root
- Attach a left or right child to the binary tree’s root
- Attach a left or right subtree to the binary tree’s root
- Detatch the left or right subtree of a binary tree’s root
- Return a copy of the left or right subtree of the binary tree’s root
- Traverse the nodes in a binary tree in preorder, inorder, or postorder
Sample code

- C&P have code for a pointer based implementation of the ADT tree.
- You can find this code on the website, modified to remove a few concepts that we have not covered.
- See sample code TreeNode.h, BinaryTree.h, BinaryTree.cpp, and useBinaryTree.cpp for more information.
Traversing Trees

• There are three ways to move to every node on a tree: preorder, inorder, and postorder.

• preorder - visit the root first, then the left subtree, then the right subtree

• inorder - visit the left subtree first, then the root, then the right subtree

• postorder - visit the left subtree first, then the right subtree, then the root
The “visit” function

- There is something new in C&P’s implementation of the binary tree, a FunctionType.
- FunctionType allows the class definition to use a function in the details of the class, but have the function defined later.
- For example, the BinaryTree class uses the function “visit” in various places. Visit, however, is a argument, so we are able to pass the visit function to the class function call.
- Note that we have defined the function “display” in useBinaryTree.cpp and we pass it to the inorderTraverse function in order to display the entire tree.
Multiple visit functions

• Using the visit function allows you to define a variety of functions (display, writeToFile, etc.) that you might want to call on each node of a tree.

• You simply pass the function to the appropriate Traverse function and the function you pass will be applied to each node on the tree.