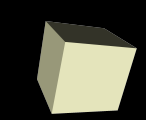
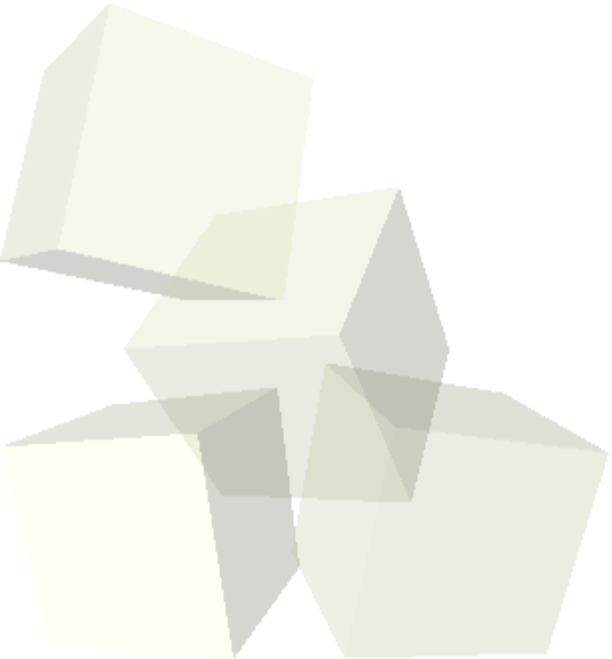


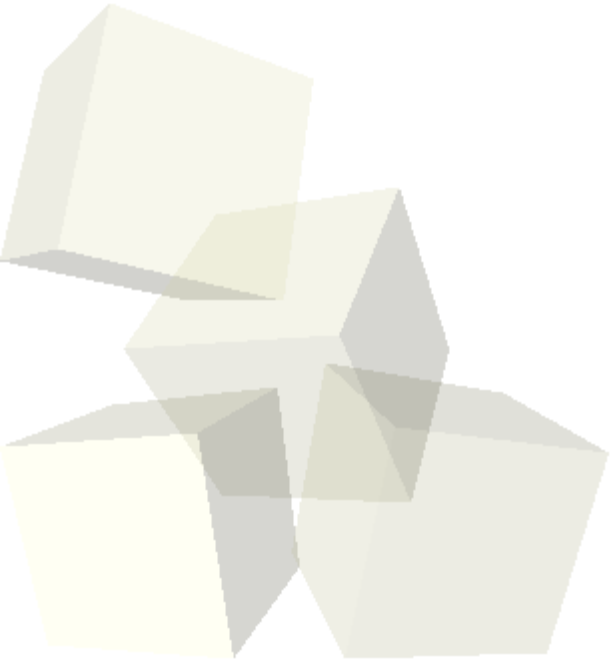
# Binary Trees

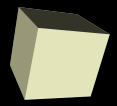
4-6-2006



# Opening Discussion

- What did we talk about last class?
- Base case for longest path.
- Do you have any questions about the assignment?





# Faster Sorting

- Let's review the faster sorts and look at code for the second.
  - Mergesort repeatedly divides the array in two doing little work until it gets down to one element. Then it merges sorted arrays as it comes back up. This can't be done in place. Always O(n log n).
  - Quicksort picks a pivot and puts all other elements before or after the pivot then recurses on sides. Does work going down. Generally O(n log n).

# What is a Tree?

- You are all familiar with what normal trees look like. In CS we use the term somewhat differently, and more formally.
- To describe trees we need some basic terminology
  - Node - an element of a tree. One node is designated as the “root”
  - Edge - a directed connection from one node to another.

# Tree Criteria

- Every node, C, has exactly one incoming edge from another node, P.  P is said to be the parent of child node C. Root has 0.
- There is a unique path from the root to any node. The number of edges on that path is called the path length.  It is also called the depth of the node.
- A node with no children is called a leaf.  The path length from a node to the deepest leaf in the height of that node.

# More Terms

- Following the parent-child analogy, children of the same node are called siblings.  We also call any node on a path below a given node a descendant and any above an ancestor.
- You might also hear the size of a node referred to as the number of descendants of a node, including itself.
- We can also define a tree as either empty, or a root with zero or more subtrees where the root connects to the roots of those subtrees.

# General Tree Implementation

- In a general tree, each node can have zero or more children. That is a lot of flexibility. We want a class to represent nodes. To get this flexibility we can use a linked list. Each node has pointers to a first child and the next sibling.
- It might be just as easy to have the child member be a Vector that we put Nodes in. File systems are a good example of this.

# Traversals

- As with any data structure one of the things you want to be able to do is to traverse through all the elements.
- Think for a while about how you would do this? There is even a question about the order you traverse them in. Do you want to process a node before you process its children or after? If before we call it a preorder traversal. If after it is a postorder traversal.

# Traversals and Recursion

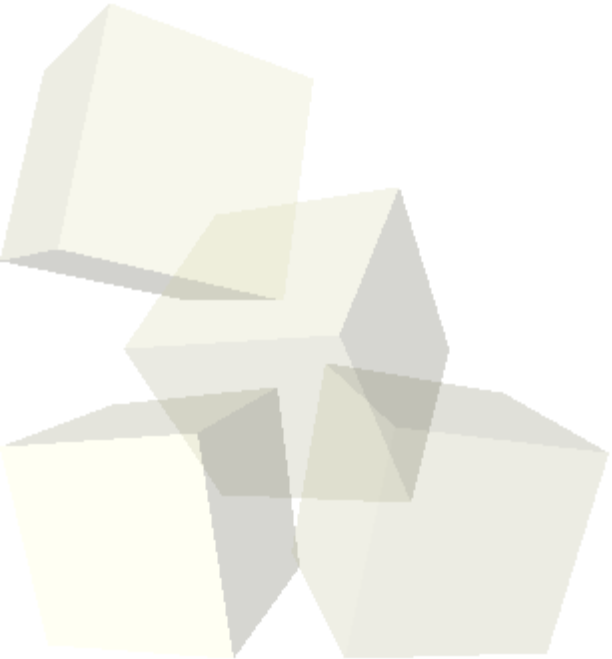
- The simplest way to do a traversal is through recursion. If you want to do it with a loop you have to implement a data structure to store some nodes or have the tree specially set up.
- The traverse function takes a node and calls itself once with each child node.  It also does whatever the visit operation is.
- Preorder does a visit before going to children and postorder visits after going to children.
- Breadth first uses a queue, not recursion.

# Binary Trees

- Sometimes we want to limit how many children a node has.  One of the most commonly used trees in programming is the binary tree where no node has more than 2 children.
- The children are often called left and right.  Your book has a fair bit more discussion of binary trees that we won't go into right now but you should look at.

# In-order Traversal

- For a binary tree there is an extra type of traversal called an in-order traversal where the node is visited between the recursion down left and right.
- Equations are great examples of trees.  We typically write them out in the in-order.  We could just as well write them out in post-order or pre-order.



# Sorted Binary Trees

- One of the best uses of binary trees is the sorted binary tree.  In this type of tree, we store data in every node and below any node we put lesser values to the left and greater values to the right.
- We find elements by going down the tree always going left or right.  This gives us behavior like a binary search, but the tree is more flexible because adds and removes are quite efficient as well.

# Adding and Removing

- The code for both adding and removing from a binary tree begins like a search that keeps track of previous (much like a linked list).
  - The add always goes to a leaf and adds the new element to the proper side.
  - The remove replaces the node we are removing with either the greatest node on the left or the smallest node on the right.

# Minute Essay

- What would a binary tree look like if you added the numbers 5,3,8,1,6,2,9 in that order?
- Remember that the design for assignment #6 is due today.