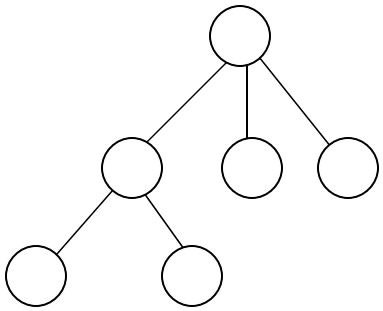
Trees

There are many, many types of trees, and they are used EVERYWHERE in software!

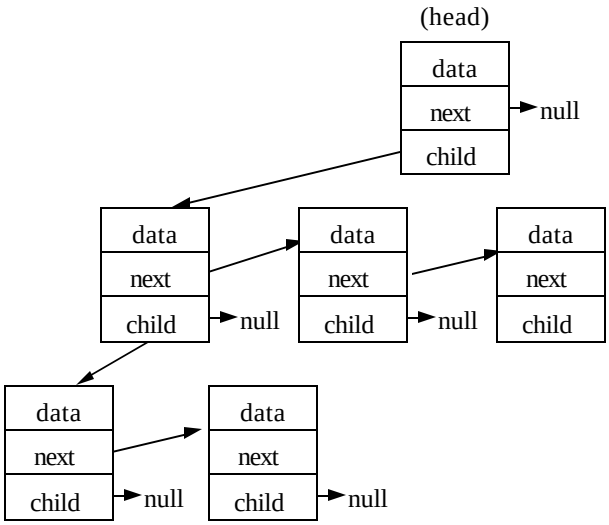
Here's some of the types you should know:

Non-Specialized Trees:

A regular tree may look like this:



Which in reality would be represented in code like this:



Does this have any special properties?

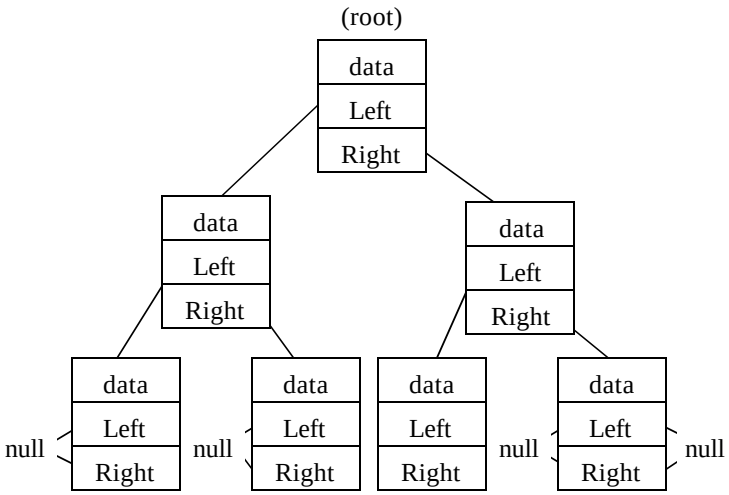
What would we use a tree like this for?

How would we move around, or *traverse* this tree? (Is there more than one way?)

Trees, part 2

Binary Trees:

Binary trees are a type of Trees. Binary trees are important because we know exactly how many children each node can have: 0, 1, or 2. A binary tree in code would look like this:



This property allows us to assume that each node is directly reachable by its parent, enabling us to move around the tree in different ways.

Now, lets look at a few specialized types of Binary Trees with some more requirements on them.

Example:

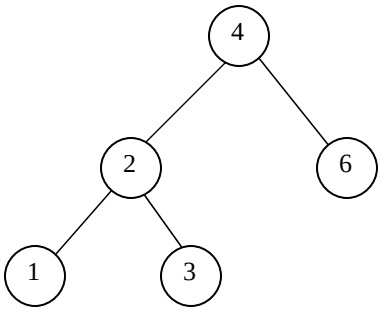
Binary Search Trees

Trees, part 3

Binary Search Trees (BST):

A Binary Search Tree is a Binary Tree that has the requirement that for all nodes, children on the left of each node have a value that is **less** than that node, while children on the right have values **greater** than each node.

Example:



Given this, how would I ALWAYS find the node that has the next-greater value than the root?

If I wanted to find the contents of one node in a regular Binary Tree, how much time would it take? Why?

Best Case: O( ) *Worst Case: O( )* *Average Case: O( )*

Now, compare this to the time it would take for a Binary SEARCH Tree to find one node:

Best Case: O( ) *Worst Case: O( )* *Average Case: O( )*

You can see the power in Binary Search TreesTrees. Now, lets look at a few specialized types with

Now, an interesting question: **If I said there was a way to make a Binary Search Tree that would have a worst case find time that was ALWAYS the same as the average case find time, would you believe me?**

What would I have to do in order to do this?