Trees

Overview

Trees are a data structure that utilizes nodes and parent-child relationships between nodes to store data in a tree-like structure. Here is an example class for a binary tree (tree with two children nodes).

```
public class Tree {
    Node root;
    private class Node {
        int val;
        Node left;
        Node right;

        public Node(int v, Node l, Node r) {
            val = v;
            left = l;
            right = r;
        }
    }

    public Tree(Node n) {
        root = n;
    }

    ... // rest of class
}
```

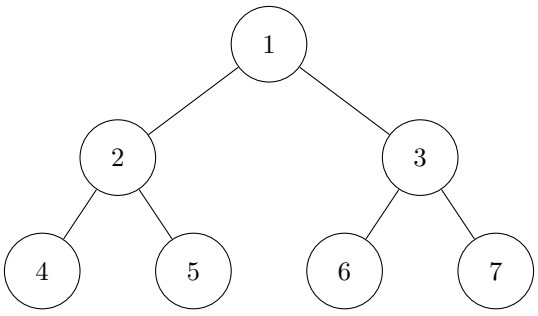
It may seem strange that we wrap the Node class within the Tree class, but it makes operating on the tree a lot easier when the data structure is implemented this way. The Tree class only keeps record of the root of the tree, while each individual Node will keep track of the structure of the tree.

Trees alone aren't very helpful, but they are the building blocks of several useful data structures including, Binary Search Trees, Balanced Trees, Game Trees. They are also useful in understanding search algorithms in graphs.

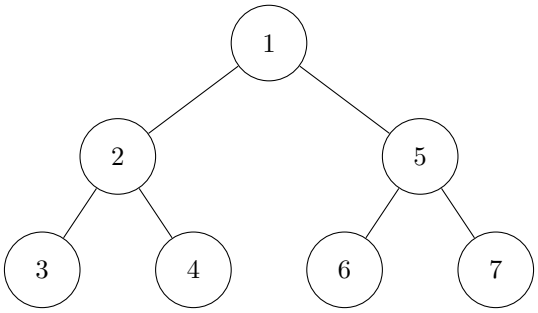
Tree Traversals

When we talk about trees, it is important to consider the different ways we can traverse the trees. There are two main tree traversals that we talk about in CS61B, BFS and DFS.

Breath First Search (BFS)
For BFS, we traverse a tree by the levels from left to right. If we had a complete tree with three levels, here is how BFS would traverse the tree with the numbers inside the node being the order of our traversal. The BFS traversal of this tree would be 1, 2, 3, 4, 5, 6, 7.



Depth First Search (DFS)
For DFS, we traverse a branch all the way until we hit a leaf, and then we keep going up a level and traversing until we hit another leaf and so forth until the tree is entirely traversed. A picture will make this easier to understand.



DFS will visit node 1 first, then it'll go to node 2, then node 3, then back to node 2, then node 4, then back to node 2, then back to node 1, then to node 5, then to node 6, then back to node 5, then to node 7.

When we talk about DFS, there are three main ways we write out the traversals, preorder, postorder, and inorder.

Preorder: Root then Children (left, right)
Postorder: Children (left, right) then Root
Inorder (only for binary trees): Left, Root, Right

To conduct a preorder DFS traversal on this tree, we start by visiting the first node. We then recursively go through the left and the right until we are done. The process looks something like this.

Preorder(1): 1, preorder(2), preorder(5)
Preorder(2): 2, preorder(3), preorder(4)
Preorder(3): 3
Preorder(4): 4
Preorder(2): 2, 3, 4
Preorder(1): 1, 2, 3, 4, preorder(5)
Preorder(5): 5, preorder(6), preorder(7)
Preorder(6): 6
Preorder(7): 7
Preorder(5): 5, 6, 7
Preorder(1): 1, 2, 3, 4, 5, 6, 7

The process for postorder and inorder is no different. Where you make the recursive call is what changes, which will change the final output.

Postorder process
Postorder(1): postorder(2), postorder(5), 1
⋮
Postorder(1): 3, 4, 2, 6, 7, 5, 1

Inorder process
Inorder(1): inorder(2), 1, inorder(5)
⋮
Inorder(1): 3, 2, 4, 1, 6, 5, 7

TL;DR

1. **DBS:** traverse levels from top to bottom, left to right
2. **DFS:** traverse until reach leaf, then up then leaf, repeat until done
   (a) **Preorder:** Root, Children (Left, Right)
   (b) **Postorder:** Children (Left, Right), Root
   (c) **Inorder:** Left, Root, Right