

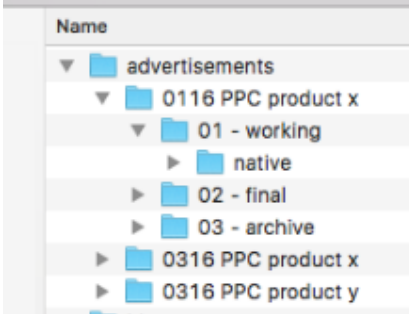
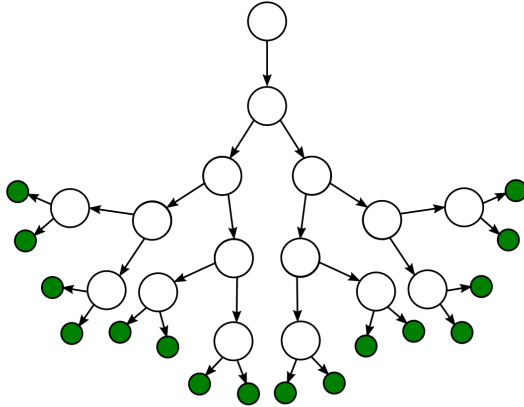
Trees Introduction

Trees are a linked data structure for representing hierarchies. They are similar to linked lists:

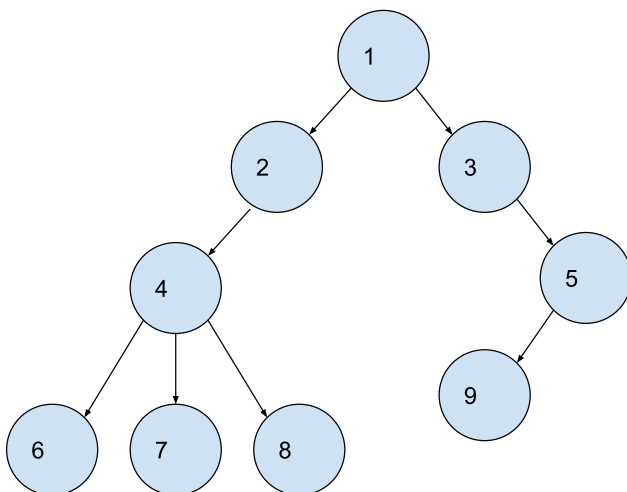
Recall Linked Lists: Node X, which holds a value, can point to another Node (like Node Y). Node X itself is pointed at by some other node.

In trees, Node X can hold multiple pointer reference variables, but it itself is only pointed at by exactly 1 other node.

Examples:

<p>Folder Systems:</p>  <p>Notice: “0116 PPC product x” points directly to multiple things:</p> <ul style="list-style-type: none">- 01 - working- 02 - final- 03 - archive <p>Something like “native” is only directly pointed at by 1 folder: “01-working”.</p>	<p>Flowcharts:</p> 
---	--

Tree Vocabulary



Edges: the direct path between two nodes.

There is an edge $1 \rightarrow 3$, but no edge $2 \rightarrow 8$

Branches: the multiple links you could follow from some node.

The branches of 4 lead you to 6,7,8.

Root Node: the only node in the tree that is not pointed at by any other node (the node at the very top). It has no parents.

The root node is 1.

Parent: The node with the start of the arrow. Node X's parent has a reference variable to node X.

The parent of 9 is 5.

Child: The node with the end of the arrow. Node Y is a child of Node Z if Z has a reference variable to Y.

The child of 2 is 4.

Siblings: if Nodes Y and Z have the same parents.

6, 7, 8 are siblings. 2 and 3 are siblings.

Leaf: Node with no children

9 is a leaf. 6,7,8 are also leaves.

Inner Node: Node with at least 1 child

4 is an inner node. 9 is not an inner node.

Depth: Number of ancestors a node has (number of steps it takes to the root).

- This could be counted as number of edges
- Or number of nodes passed along the way from the node to the root.

4 has an edge depth of 2 and a node depth of 3.

1 has an edge depth of 0, and a node depth of 1.

Height: Depth of the tree deepest node.

The height of the tree is 4 (counting by nodes), or 3 (counting by edges). Note there are multiple deepest nodes.

Subtree: Every node and all the nodes below it is considered a subtree

- Except for the root node. The root node and all the nodes below is the actual tree!

The subtree of 2 includes 2,4,6,7,8. The subtree of 9 is just 9.

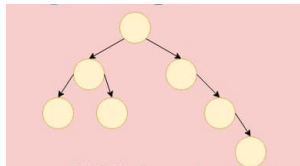
Binary Trees and Special Types of Trees

Binary Tree: all node have ≤ 2 children.

Full Tree: Every node has 0 children or 2 children.

Balanced Tree: difference in height between left and right subtrees ≤ 1 for every node.

Complete: Subtype of balanced tree where the only "missing" nodes come from the bottom right.



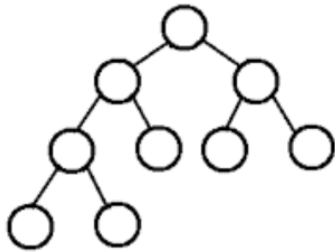
The above tree is binary because all nodes have 2 children or less.

The above tree is **not** full because 2 of its nodes have 1 child.

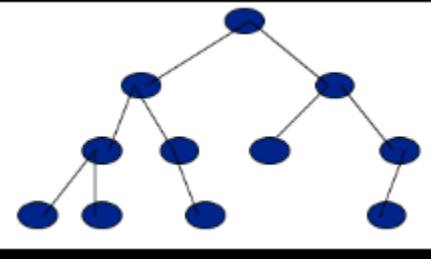
It is also not balanced.

- Even though the height difference between left and right from the root is 1, notice the height difference for the right child of the root. It's 2 edges - 0 edges = 2 > 1.

Since this tree is not balanced, it cannot be complete.



The above tree is binary, and is full (all nodes have 0 or 2 children). It is also balanced because the height difference between left and right subtrees at all nodes is 1 or less. It is even complete, because the only nodes missing come from the bottom right corner.



The above tree is binary and full. It is balanced also. However, it is not complete because the missing nodes come from the middle of the last row.

TreeNode

Since a tree is a hierarchy of nodes, here is a simple node class we will use for reference:

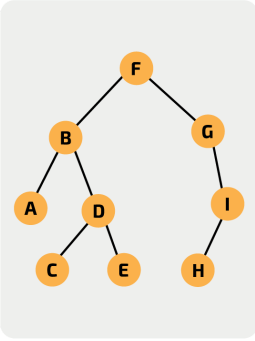
```
private static class IntTreeNode {  
    private int data;  
    private IntTreeNode leftchild;  
    private IntTreeNode rightchild;  
    public IntTreeNode(int data) {  
        this.data = data;  
    }  
}
```

Tree Traversal: BFS (breadth-first search)

This is the first way to examine every single node of a tree.

Read the tree like a book, top to bottom, left to right.

We will not learn how to implement this idea in code.



Reading this tree with BFS:

F B G A D I C E H

Tree Traversal: DFS (depth-first search)

This is the second way to examine every single node of a tree.

There are three types of depth-first search, but they all follow some ordering of the following steps:

1. Recurse on the left subtree
2. Recurse on the right subtree
3. Examine the node you are currently on.

Notice the first thing you should do in a tree traversal method is to handle the case where the node given is null.

Preorder

You look at the node before recursive calls

```
private void dfs(IntTreeNode root) {  
    if(root == null) {  
        return;  
    }  
    System.out.print(root.data + " ");  
    dfs(root.leftchild);  
    dfs(root.rightchild);  
}
```

Inorder

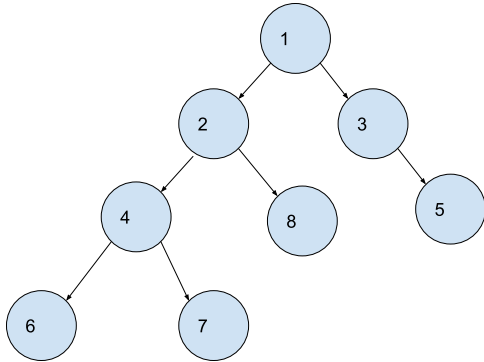
You look at the node in between recursive calls

```
private void dfs(IntTreeNode root) {  
    if(root == null) {  
        return;  
    }  
    dfs(root.leftchild);  
    System.out.print(root.data + " ");  
    dfs(root.rightchild);  
}
```

PostOrder

You look at the node after recursive calls

```
private void dfs(IntTreeNode root) {  
    if(root == null) {  
        return;  
    }  
    dfs(root.leftchild);  
    dfs(root.rightchild);  
    System.out.print(root.data + " ");  
}
```



Ex: Traverse the above tree using post-order:

6 7 4 8 2 5 3 1

Flow for the first few recursive calls:

dfs(1)

dfs(2) - from dfs(1)

dfs(4) - from dfs(2)

dfs(6) - from dfs(4)

dfs(null) - from dfs(6)

dfs(null) - from dfs(6)

print 6

dfs(7) - from dfs(4)

dfs(null) - from dfs(7)

dfs(null) - from dfs(7)

print 7

print 4

dfs(8) - from dfs(2)

And so on...

Ex: Write a method that accepts a node from a tree as a parameter and returns the maximum value found in the subtree based on that node.

We can use any traversal of our choice for this. Let's pick dfs with in-order:

```

public int maxTree(IntTreeNode node) {
    if(node == null) {
        return Integer.MIN_VALUE;
    }
    int max = maxTree(node.rightchild);
    max = Math.max(node.data, max);
    max = Math.max(maxTree(node.leftchild), max);
    return max;
}
  
```