Binary trees

Trees and graphs are probably the **most common** type of interview question (other than hash tables and arrays/strings). They show up everywhere both in the physical world and software world.

A node stores data; this data can be whatever you want - an integer, a boolean, a hash map, your own custom objects, or all of the above. A node also stores pointers to other nodes.

A **graph** is any collection of nodes and their pointers to other nodes. Linked lists and trees are types of graphs.

Even though a tree is a type of graph, they are considered different topics when it comes to algorithm problems. Graphs are more advanced so first we look at trees.

What is a tree?

There are multiple types of trees. We focus on binary trees in this course.

In a binary tree, all nodes have a **maximum** of two children. Every node has exactly one parent.

Some real life examples of trees:

- File systems
- A comment thread on an app like Reddit or Twitter
- A company's organization chart

The respective root nodes and children would be:

- The root directory, and subfolders/files
- The original post/tweet, and the comments and replies
- The CEO, and direct reports

Let's look at the company example. If we modeled the company as a tree, then each person is a node, and an edge exists from `A` to `B` if `A` manages `B`. In that case, the CEO would be the root because they are at the "top" of the company and are not managed by any other employee. Let's say the CEO has 6 reports - The people in the C-Suite (like CFO, COO, CTO). Thus, the CEO has 6 "children", so the tree is not binary. Each of the people in the C-Suite will

have people reporting to them, like VPs, and those VPs wil have directors reporting to them, and so on.

The important characteristic of the company that makes it a tree are that each person only has 1 manager (parent), and the entire tree is connected (if you start at anyone and continuously trace their managers, you will always end up at the CEO).

Tree terminology

The **depth** of a node is how far it is from the root node. The root thus has depth `0`. Every child has depth of `parentsDepth + 1`.

A **subtree** of a tree is a node and all its descendants. Trees are recursive - you can treat a subtree as if it was its own tree with the chosen node being the root. Let's look at the company example again. The entire company is **rooted** at the CEO. But what if we only cared about the engineering department? Let's say the CTO has a direct report who is a Senior Vice President of engineering, and all engineers are under this person. Take this SVPm and separate them from the rest of the company (remove their connection to the CTO). We are now left with a valid tree with the SVP as the **root**. This subtree represents the engineering department. This is the most fundamental idea for solving tree problems - **you can take any given node and treat it as its own tree**, so you can solve problems in a recursive manner.

Code representation

Just like with a linked list, binary trees are implemented using objects of a custom class:

Python3

```
class TreeNode:
    def __init__(self, val, left, right):
        self.val = val
        self.left = left
        self.right = right
```

JavaScript

```
class TreeNode {
    constructor(val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}
```

Java

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode (int val) {
        this.val = val;
    }
}
```

C++

```
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int val) : val(val), left(nullptr), right(nullptr) {}
};
```

In binary tree problems, you will be given a reference to the `root` of a binary tree as input. You can acess the root's left **subtree** with `root.left` and the right **subtree** with `root.right`. Like linked lists, each node carries a value `val` as data. If a node doesn't have a left child, then `node.left` will be `null`, and vice-versa with the right child. If both children are null, then the node is a **leaf**.