Trees & Binary Trees

Springboard

Trees

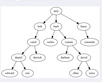
Goals

- Introduce terminology
- Create a tree class and methods
- Learn uses for trees

Terminology

node
basic unit
children
nodes directly below a node
descendants
nodes below a node
parent
node that is directly above a node
ancestor
node that is above a node
root node
node at the top of tree
leaf node
node without any children

An Org Chart is a Tree

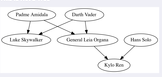


A Filesystem is a Tree

HTML DOM is a Tree

A Taxonomy is a Tree

This Is Not a Tree



- Trees need a root node — we don't have one!
- A node can only have one parent

Binary Trees/Binary Search Trees

These are different—and we'll cover later!

General trees are sometimes called "n-ary" trees, since they can have n (any) number of children.

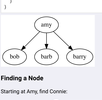
Trees in JavaScript

Node Class

```
class Node {
  constructor(val, children = []) {
    this.val = val;
    this.children = children;
  }
}
```

```
let amy = new Node("amy");

amy.children.push(new Node("bob"));
amy.children.push(new Node("barb"));
amy.children.push(new Node("barry"));
```

```
let amy = new Node("amy",
  [new Node("bob"),
   new Node("barb"),
   new Node("barry")])
```



Finding a Node

Starting at Amy, find Connie:

demo/trees.js

```
  find(val) {
    let toVisitStack = [this];

    while (toVisitStack.length) {
      let current = toVisitStack.pop();

      if (current.val === val)
        return current;

      for (let child of current.children)
        toVisitStack.push(child)
    }
  }
```

"Depth First Search" (uses stack)

Highest-Ranking Consuela

Say we hire another Consuela, a VP, & we want to find her

demo/trees.js

```
  findBFS(val) {
    let toVisitQueue = [this];

    while (toVisitQueue.length) {
      let current = toVisitQueue.shift();

      if (current.val === val)
        return current;

      for (let child of current.children)
        toVisitQueue.push(child)
    }
  }
```

"Breadth First Search" (uses queue)

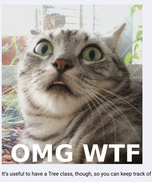
Tree Class

```
class Tree {

  constructor(root) {
    this.root = root;
  }
}
```

```
let org = new Tree(
  new Node("amy",
    [new Node("bob"),
     new Node("barb"),
     new Node("barry")]))
```

Do You Really Need a Tree Class?

Each node is, itself, a tree



It's useful to have a Tree class, though, so you can keep track of the head node!

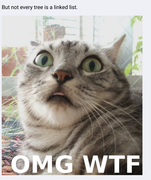
Can delegate to the head node for many operations:

```
class Tree {
  constructor(root) {
    this.root = root;
  }

  /** findInTree: return node in tree w/this val */

  findInTree(val) {
    return this.root.find(val)
  }

  /** findInTreeBFS: return node in tree w/this val */

  findInTreeBFS(val) {
    return this.root.findBFS(val)
  }
}
```

Also

Every linked list is a tree

But not every tree is a linked list.



Binary Trees

General n-ary trees have nodes with 0+ children.

Binary tree nodes can have 0, 1, or 2 children.

Binary tree nodes are usually structured with ***left*** and ***right*** properties, rather than an array of ***children***:

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

What Are They Good For?

Sometimes they're used to store data in a normal hierarchy, like a tree.

Often times, they have a "rule" about the arrangement:

- binary search trees
- min/max heap

Other Trees

Less commonly, there are other *n* trees

One example is "quad-trees", often used for geographic programs, to keep track of N/S/E/W information from a node.

Advanced Ideas

Moving Up


```
barb.children // carlos and connie nodes
```

It's not possible to "move up":

```
barb.findParents()   // can't do easily
```

Some Trees Point Up


```
class ReverseNode {
  constructor(parent) {
    this.parent = parent;
  }
}
```

Of course, then you can't move down!

So you'd need to keep a list of all leaf nodes

Some Point Both Ways


```
class BidirectionalNode {
  constructor(parent, children = []) {
    this.parent = parent;
    this.children = children;
  }
}
```

Resources

How to Not Be Stumped By Trees