Binary Search Trees

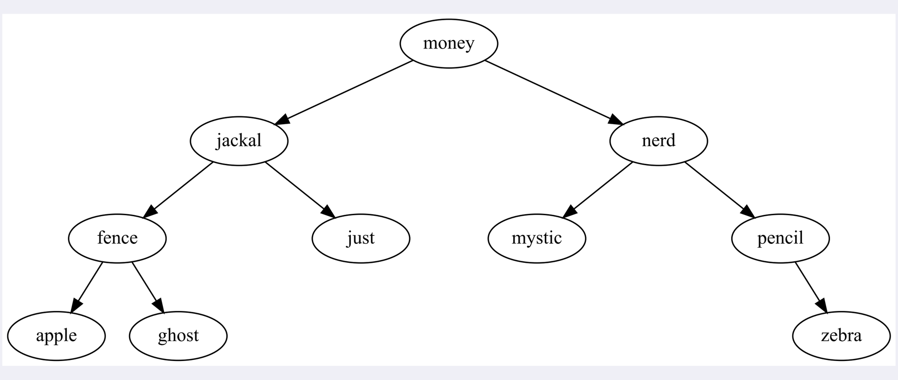
Download Demo Code

A List of Words

Imagine this list of words:

> *apple, fence, ghost, jackal, just, money, mystic, nerd, pencil, zebra*

Binary Search Tree



- Also a tree, made of nodes
- But each node has a left and right child
- Has a "rule" for arrangement
  - Often used for fast searching

Implementing BSTs

Node Class

Node class is same as any other binary Node class:

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

  // other methods here
}
```

Tree Class

Just like with n-ary trees, may not *always* need class for tree.

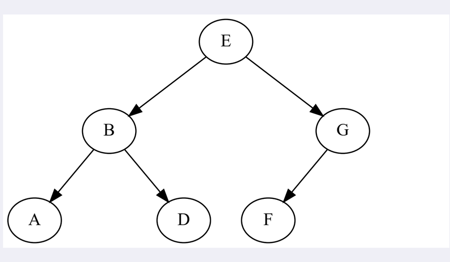
But it's very useful for keeping track of root of tree:

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

  // other methods here
}
```

Searching

Binary Search Tree Find



demo/bst.js

```
find(sought) {
  let current = this;

  while (current) {
    if (current.val === sought)
      return current;

    current = sought < current.val
              ? current.left
              : current.right;
  }
}
```

Starting at ***E***, looking for ***C***:

1. ***C*** comes before ***E***, so go left to ***B***
2. ***C*** comes after ***B***, so go right to ***D***
3. ***C*** comes before ***D***, so go left to `None`
4. Drop out of `while` loop and return `None`

Every choice we make reduces # options by half!

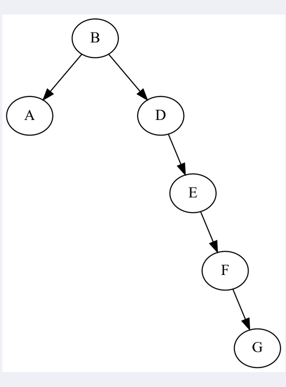
For ***n*** nodes, we need to search, at most O(log n) nodes

We can search >1,000 nodes in only 10 steps!

We can search >1,000,000 nodes in only 20 steps!

Balancing

Valid But Badly Balanced



- Can find ***A*** efficiently
- Can find missing ***C*** efficiently
- Can't find ***G*** efficiently
- Tree needs to be "balanced"

Balancing Trees

Easy ways to get reasonably balanced trees:

- shuffle values for tree randomly, and then insert
- or sort values, then insert from the middle working out

Self-Balancing Trees

There are structure/algorithm pairs for BSTs that can balance themselves:

AVL Trees
: Keeps balanced. Simpler algorithm but slightly less efficient.

Red/Black Trees
: Keeps "reasonably" balanced. More complex algorithm but can be more efficient.

Traversal

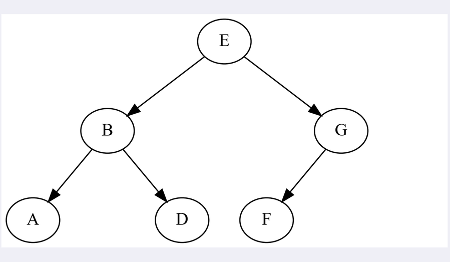
Often, you don't want to look at every node in a BST

That's the point — you can search without looking at each!

But sometimes you will want to traverse entire tree

In Order Traversal

```
traverse(node) {
  if (node.left) traverse(node.left);
  console.log(node.val);
  if (node.right) traverse(node.right);
}
```

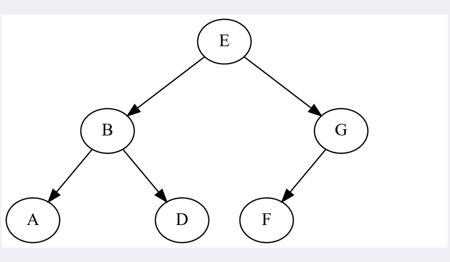


"traverse left, myself, traverse right" is "in-order":

A → B → D → E → F → G

Pre Order Traversal

```
traverse(node) {
  console.log(node.val);
  if (node.left) traverse(node.left);
  if (node.right) traverse(node.right);
}
```

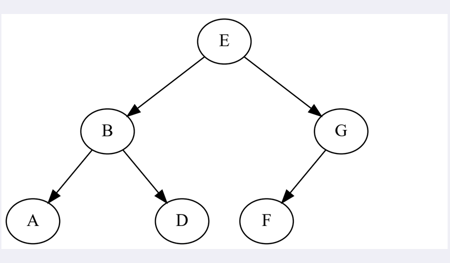


"myself, traverse left, traverse right" is "pre-order":

E → B → A → D → G → F

Post Order Traversal

```
traverse(node) {
  if (node.left) traverse(node.left);
  if (node.right) traverse(node.right);
  console.log(node.val);
}
```

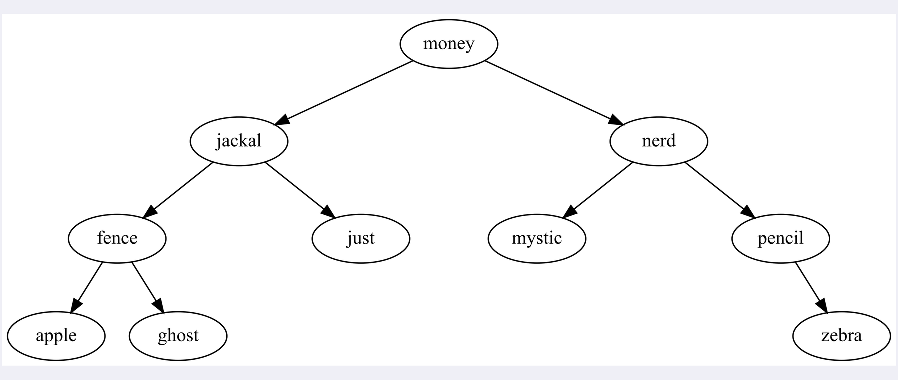


"traverse left, traverse right, myself" is "post-order":

A → D → B → F → G → E

Binary Trees vs Hashmap

How do they compare?



Hashmaps

- O(1) lookup/addition/deletion
- Have know exactly what you're looking for
- Can't find "first word equal or after banana"
- Can't find range of "words between car and cat"

Binary Search Trees

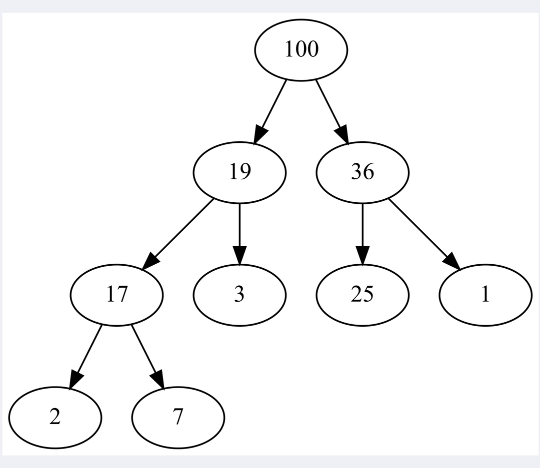
- O(log n) lookup/addition/deletion
- Can search for exact value, or inequalities
- Can search for ranges
- Often used to implement indexes in databases

Heaps

Another ordered binary tree is a ***MinHeap*** or ***MaxHeap***.

They're used to efficiently implement priority queues.

Their ordering rule is "parent must be lower *[for MaxHeap, larger]* than its children"



Resources

Leaf It Up To Binary Trees

The Little AVL Tree That Could

Trees & Binary Search Trees video