Beauty is Only $\Theta(\log n)$ Deep

- Binary Search Trees are fast if they’re shallow:
  - perfectly complete
  - perfectly complete except the one level fringe (like a heap)
  - anything else?

What matters here? Problems occur when one branch is much longer than the other!

Balance

- Balance
  - $\text{height(left subtree)} - \text{height(right subtree)}$
  - zero everywhere $\Rightarrow$ perfectly balanced
  - small everywhere $\Rightarrow$ balanced enough

Balance between -1 and 1 everywhere $\Rightarrow$ maximum height of $1.44 \log n$

AVL Tree Dictionary Data Structure

- Binary search tree properties
  - binary tree property
  - search tree property
- Balance property
  - balance of every node is: $-1 \leq b \leq 1$
  - result:
    - depth is $\Theta(\log n)$

Testing the Balance Property

NULLs have height -1

An AVL Tree
**Beautiful Balance**

- Insert(middle)
- Insert(small)
- Insert(tall)

**Bad Case #1**

- Insert(small)
- Insert(middle)
- Insert(tall)

**Single Rotation**

**General Single Rotation**

- Height of subtree same as it was before insert!
- Height of all ancestors unchanged.

**Bad Case #2**

- Insert(small)
- Insert(tall)
- Insert(middle)

**Double Rotation**
General Double Rotation

- Height of subtree still the same as it was before insert!
- Height of all ancestors unchanged.

Insert Algorithm

- Find spot for value
- Hang new node
- Search back up for imbalance
- If there is an imbalance:
  - case #1: Perform single rotation and exit
  - case #2: Perform double rotation and exit

Easy Insert

Hard Insert (Bad Case #1)

Single Rotation

Hard Insert (Bad Case #2)
Single Rotation (oops!)

Double Rotation (Step #1)

Double Rotation (Step #2)

Look familiar?

AVL Algorithm Revisited

- Recursive
  1. Search downward for spot
  2. Insert node
  3. Unwind stack, correcting heights
     a. If imbalance #1, single rotate
     b. If imbalance #2, double rotate

- Iterative
  1. Search downward for spot, stacking parent nodes
  2. Insert node
  3. Unwind stack, correcting heights
     a. If imbalance #1, single rotate and exit
     b. If imbalance #2, double rotate and exit

Single Rotation Code

```c
void RotateRight(Node root) {
    Node temp = root.right
    root.right = temp.left
    temp.left = root
    root.height = max(root.right.height(), root.left.height()) + 1
    temp.height = max(temp.right.height(), temp.left.height()) + 1
    root = temp
}
```

Double Rotation Code

```c
void DoubleRotateRight(Node root) {
    RotateLeft(root.right)
    RotateRight(root)
}
```

First Rotation
Double Rotation Completed

AVL
- Automatically Virtually Leveled
- Architecture for inVisible Leveling (the "in" is inVisible)
- All Very Low
- Absolut Vodka Logarithms
- Amazingly Vexing Letters

Bonus: Deletion (Easy Case)

Deletion (Hard Case #1)

Single Rotation on Deletion

Coming Up
- Splay trees
- B-trees
- and even more trees!