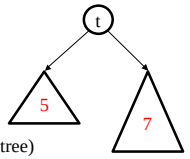
CSE 326: Data Structures
Topic #6: AVL Trees

Luke McDowell
Summer Quarter 2003

We want a "balanced" tree

- Left and right subtrees equal # nodes?
- Left and right subtrees equal height?

Balance

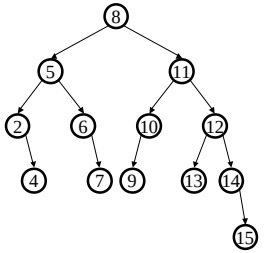


- Balance
  - height(left subtree) - height(right subtree)
  - zero everywhere ??
  - small everywhere ?

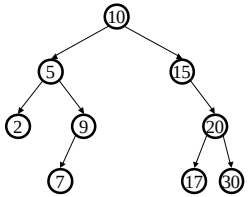
Balance between -1 and 1 everywhere ?
maximum height of 1.44 log n

AVL Tree
Search Data Structure

- Binary search tree properties
  - binary tree property
  - search tree property
- Balance property
  - balance of every node is: `-1`?? ***b*** ? `1`
  - result:
    - depth is ? **`(log n)`**



Testing the Balance Property

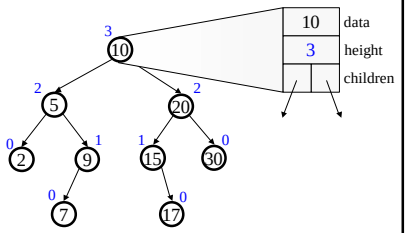


We need to be able to:

1.
2.
3.

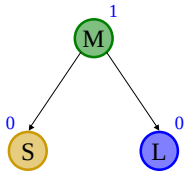
`NULL`s have height **`-1`**

An AVL Tree



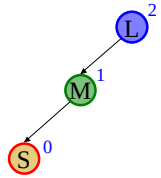
Beautiful Balance

Insert(middle)
Insert(short)
Insert(long)

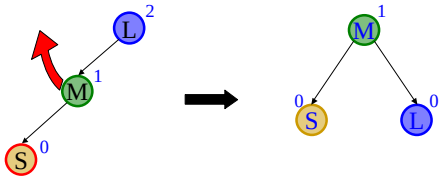


Bad Case #1

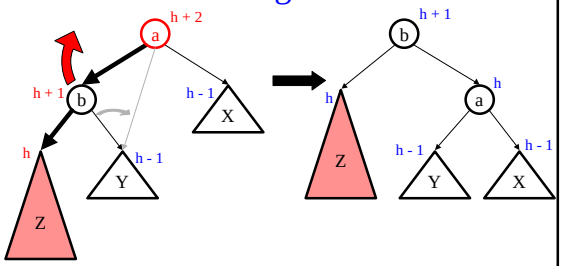
Insert(long)
Insert(middle)
Insert(short)



Single Rotation



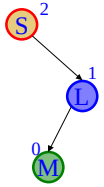
General Single Rotation



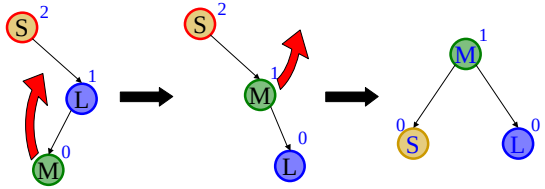
- Height of subtree same as it was before insert!
- Height of all ancestors unchanged. So?

Bad Case #2

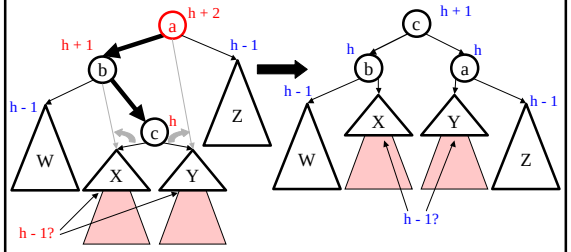
Insert(short)
Insert(long)
Insert(middle)



Double Rotation



General Double Rotation



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

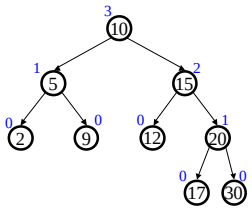
Insert Algorithm

- 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

Loop to fix all problems?

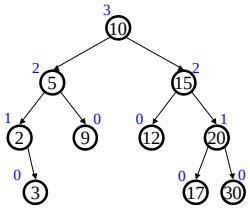
Easy Insert

Insert(3)



Hard Insert (Bad Case #1)

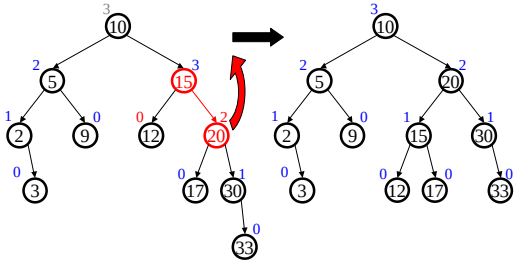
Insert(33)



How to fix?

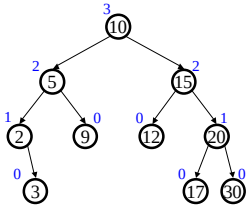
How did we know?

Single Rotation



Hard Insert (Bad Case #2)

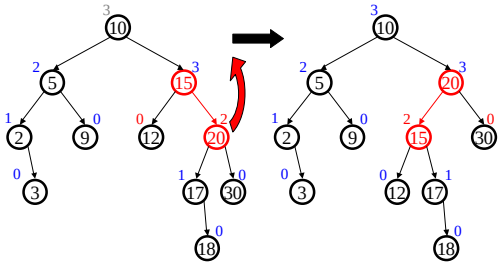
Insert(18)



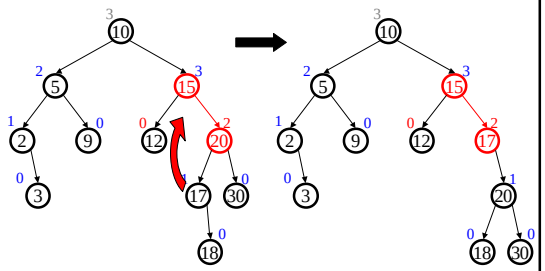
How to fix?

How did we know?

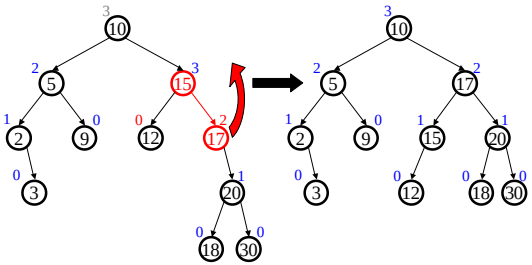
Single Rotation (oops!)



Double Rotation (Step #1)



Double Rotation (Step #2)



AVL Algorithm Revisited

- Recursive

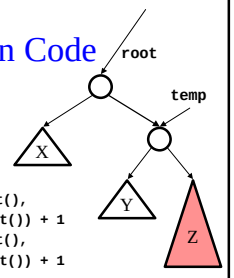
1. Search downward for spot
2. Insert node
3. Unwind stack, correcting heights
   a. If imbalance #1, single rotate
   b. If imbalance #2, double rotate

- 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

Why use a stack?

Single Rotation Code

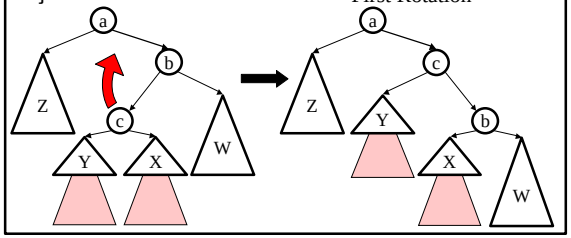



```
void RotateRight(Node root) {
  Node temp = root.right
  root.right = temp.left
  temp.left = root
  root.height = max(root.right.height(),
                    root.left.height()) + 1
  temp.height = max(temp.right.height(),
                    temp.left.height()) + 1
  root = temp
}
```

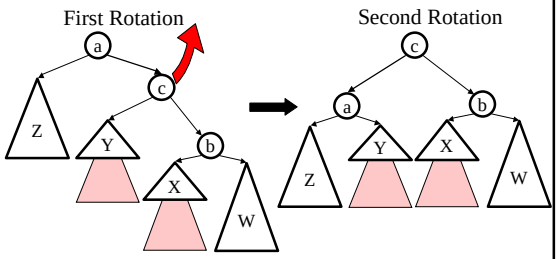
Double Rotation Code

```
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

Coming Up

- Splay trees
- B-trees
- Hashing

To do

- Homework 3:
  - Turn in part A (Thursday, 11 p.m.)
  - Read/start part B – bring questions to section
- Finish reading chapter 4