CSE 326: Data Structures
Lecture #9
Amazingly Vexing Letters

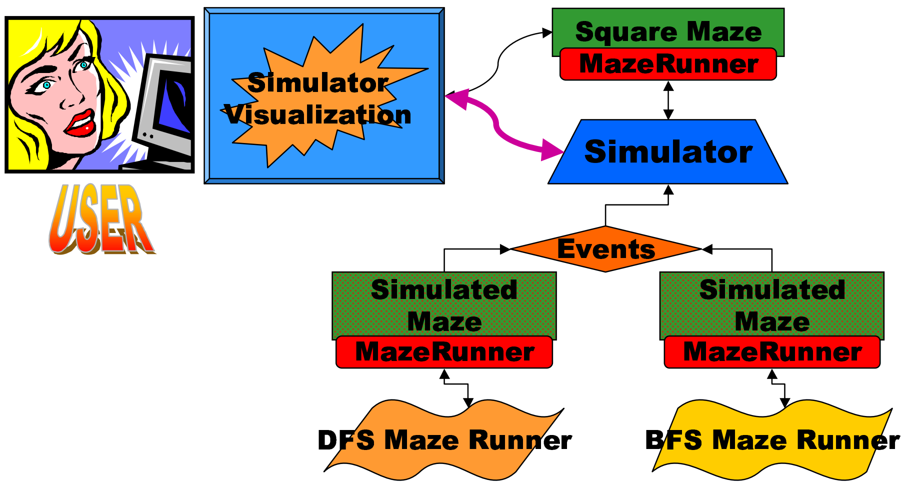
Bart Niswonger

Summer Quarter 2001

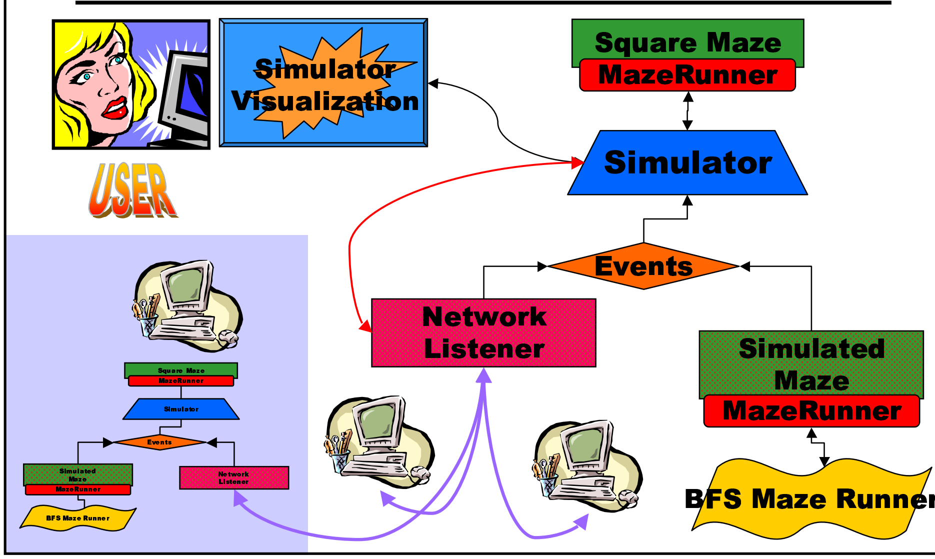
Today's Outline

- Project II Discussion
  - Testing
  - "Software Engineering"
  - Threads
  - Role of Documentation
- AVL Trees
  - Rotations
  - Insertions

Architecture



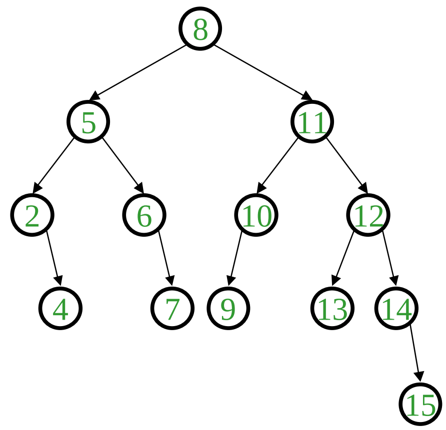
Networked Architecture



AVL Tree
Dictionary Data Structure

- Binary search tree properties
  - binary tree property
  - search tree property
- Balance property
  - balance of every node is:

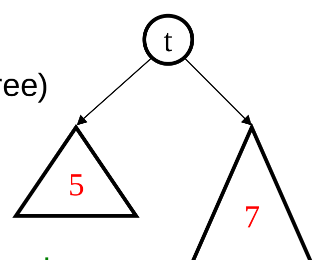
    $-1 \leq b \leq 1$
  - result:
    - depth is $\Theta(\log n)$



Balance

- Balance:

  height(left subtree) - height(right subtree)



Ø zero everywhere ⇒ perfectly balanced

Ø small everywhere ⇒ balanced enough

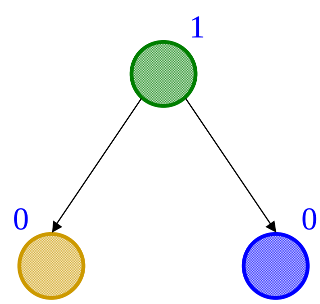
Balance between -1 and 1 everywhere ⇒
maximum height of 1.44 log n

But, How Do We Stay Balanced?

- I need:
  - the smallest person in the class
  - the tallest person in the class
  - the averagest (?) person in the class

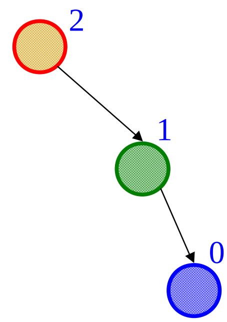
Beautiful Balance

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

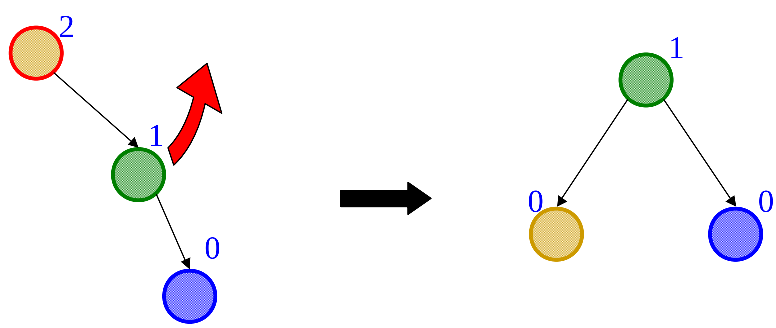


Bad Case #1

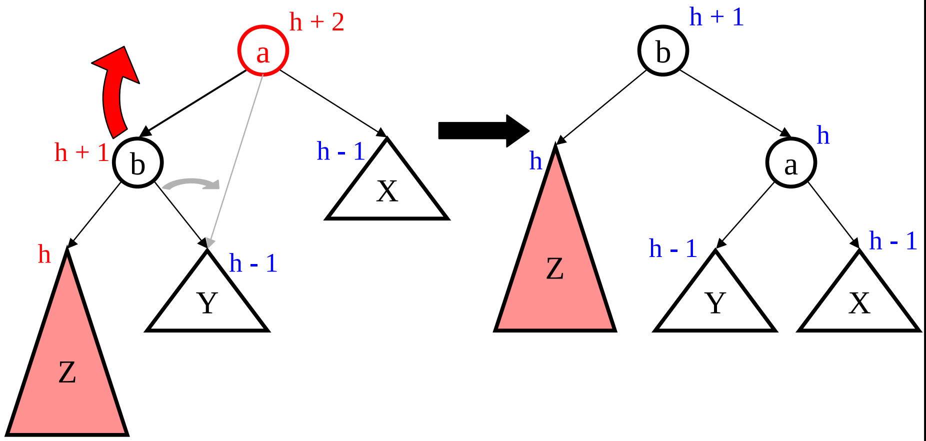
Insert(small)
Insert(middle)
Insert(tall)



Single Rotation



General Single Rotation

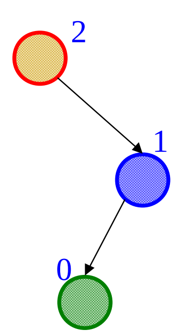


- Height of subtree same as it was before insert!
- Height of all ancestors unchanged.

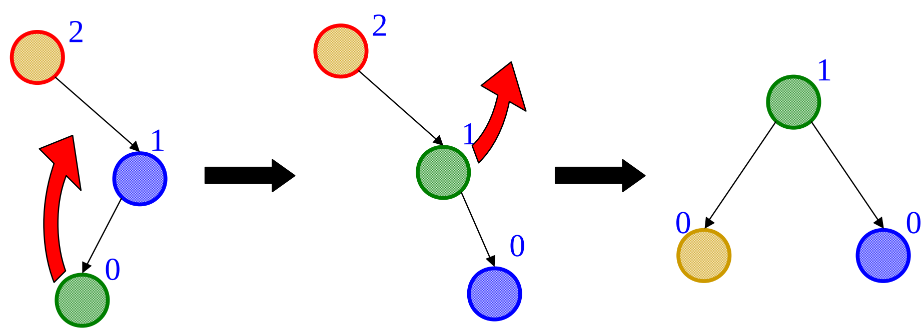
So?

Bad Case #2

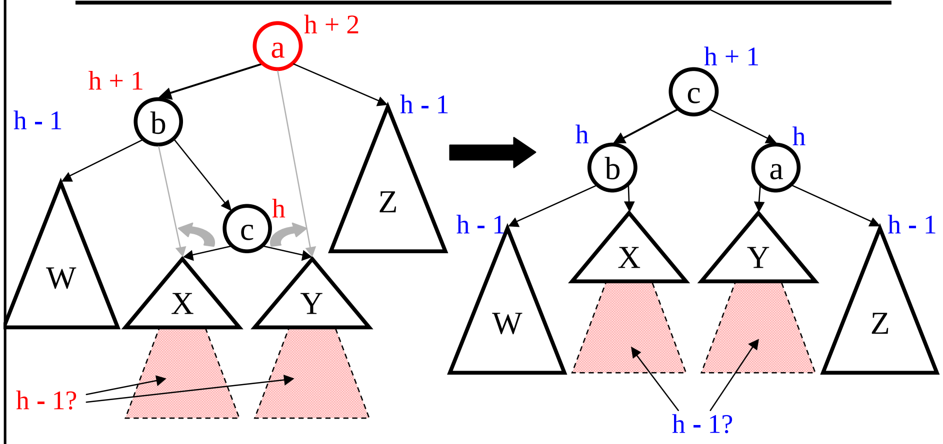
Insert(small)
Insert(tall)
Insert(middle)



Double Rotation



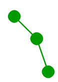
General Double Rotation

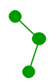


- Height of subtree **still** the same as it was before insert!
- Height of all ancestors unchanged.

Insert Algorithm

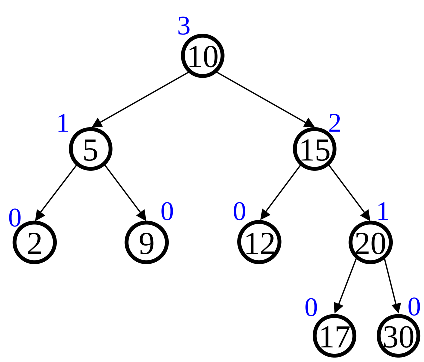
- Find spot for value
- Hang new node
- Search back up for imbalance
- If there is an imbalance:


case #1: Perform single rotation and exit


case #2: Perform double rotation and exit

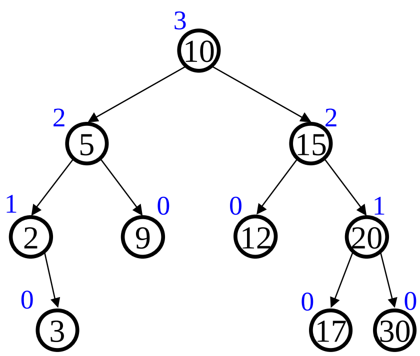
Easy Insert

Insert(3)

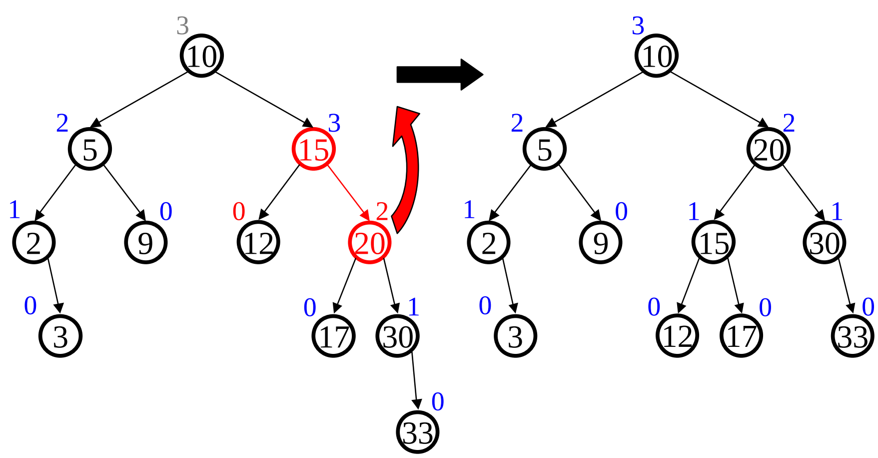


Hard Insert (Bad Case #1)

Insert(33)

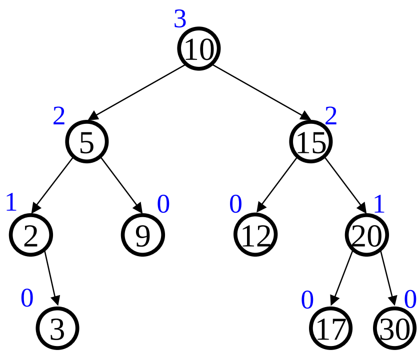


Single Rotation

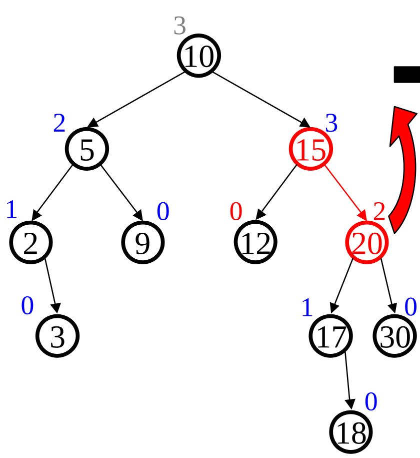


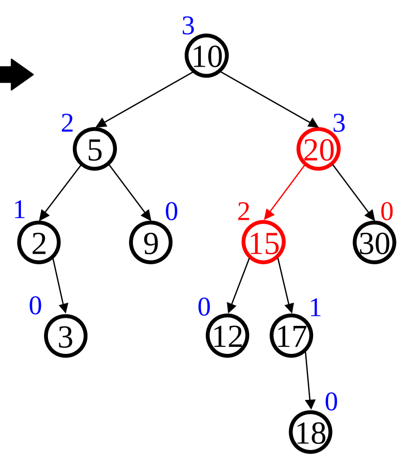
Hard Insert (Bad Case #2)

Insert(18)

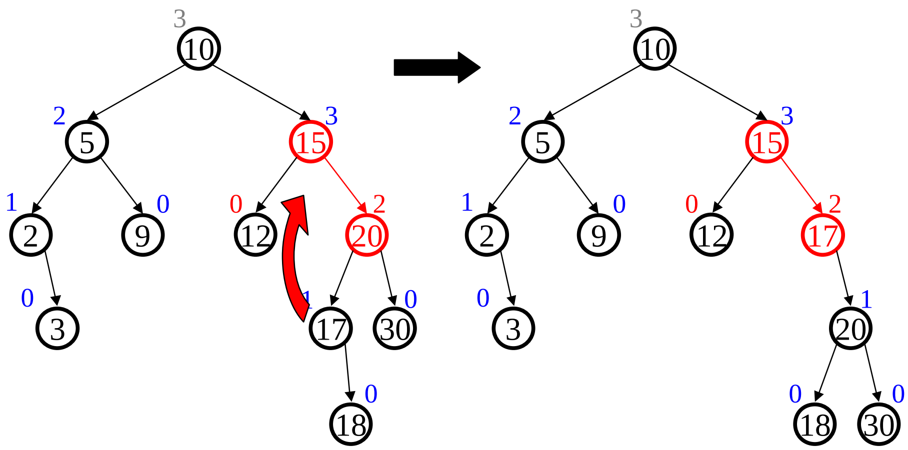


Single Rotation (oops!)

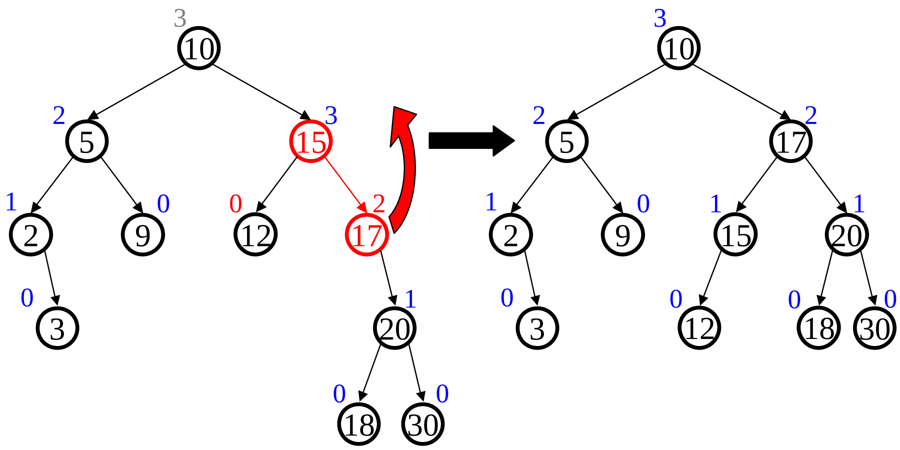




Double Rotation (Step #1)



Double Rotation (Step #2)



AVL Algorithm Revisited

Recursive

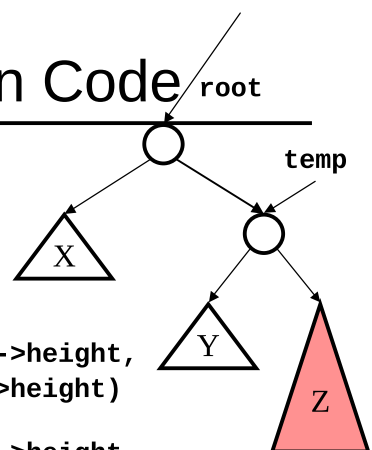
1. Search downward for spot
2. Insert node
3. Unwind stack, correcting heights
   a. If imbalance #1, single rotate
   b. If imbalance #2, double rotate

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

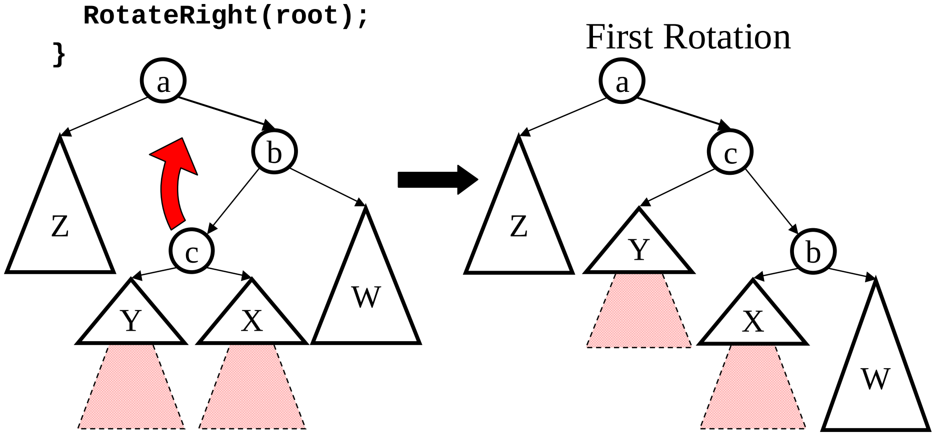
Single Rotation Code

```
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;
}
```

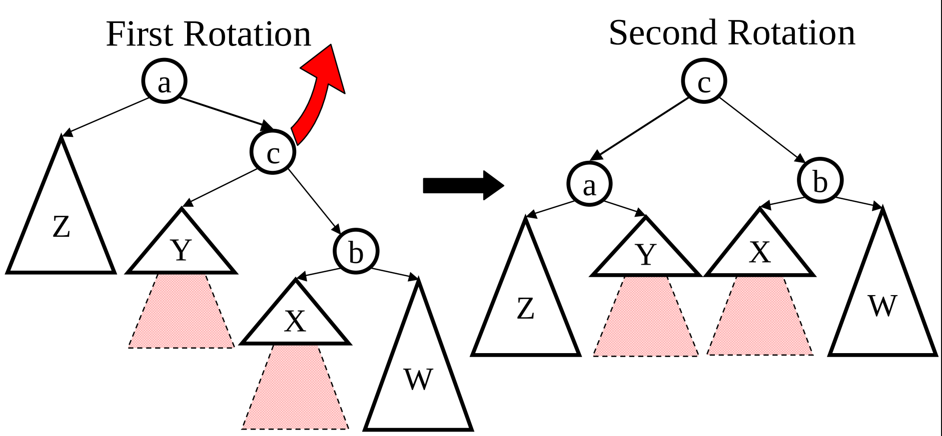


Double Rotation Code

```
void DoubleRotateRight(Node *& root) {
  RotateLeft(root->right);
  RotateRight(root);
}
```



Double Rotation Completed



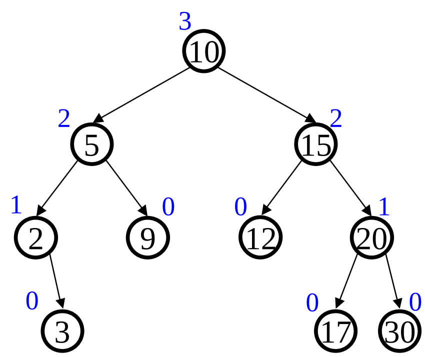
AVL

- Automatically Virtually Leveled
- Architecture for inVisible Leveling (the "in" is inVisible)
- All Very Low
- Articulating Various Lines
- Amortizing? Very Lousy!
- Absolut Vodka Logarithms
- Amazingly Vexing Letters

Adelson-Velskii Landis

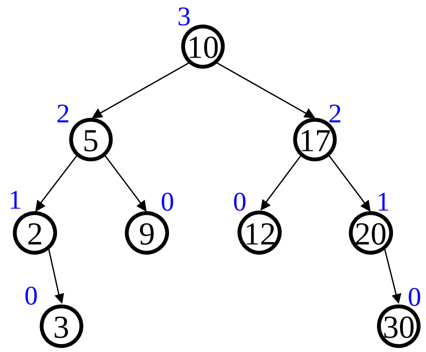
Bonus: Deletion (Easy Case)

Delete(15)

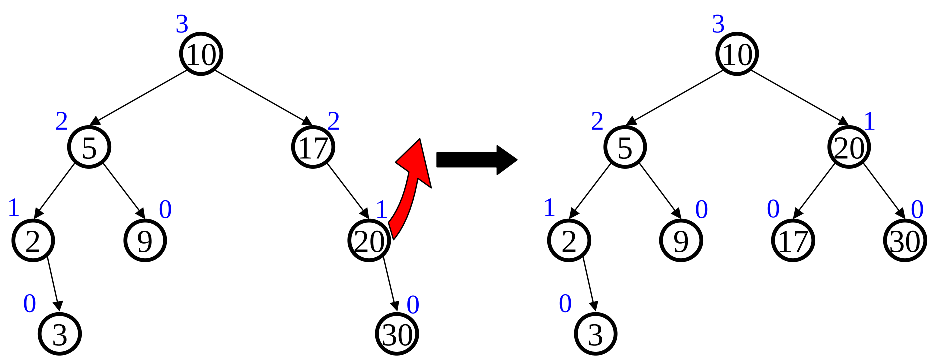


Deletion (Hard Case #1)

Delete(12)



Single Rotation on Deletion



Something *very* bad happened!

To Do

- Project II-A for Wednesday
- Read through section 4.7 in the book
- Comments & Feedback
- Homework III

Coming Up

- No Quiz Thursday
- Midterm next week
- Project II – the writeup!
- Even more balancing acts
- A **Huge** Search Tree Data Structure