Balanced BST

Observation
- BST: the shallower the better!
- For a BST with $n$ nodes
  - Average height is $\Theta(\log n)$
  - Worst case height is $\Theta(n)$
- Simple cases such as insert(1, 2, 3, ..., $n$)
  lead to the worst case scenario

Solution: Require a Balance Condition that
1. ensures depth is $\Theta(\log n)$ — strong enough!
2. is easy to maintain — not too strong!

Potential Balance Conditions

1. Left and right subtrees of the root
   have equal number of nodes
2. Left and right subtrees of the root
   have equal height

The AVL Balance Condition

Left and right subtrees of every node
have heights differing by at most 1

Define: $\text{balance}(x) = \text{height}(x.\text{left}) - \text{height}(x.\text{right})$

AVL property: $-1 \leq \text{balance}(x) \leq 1$, for every node $x$

- Ensures small depth
  - Will prove this by showing that an AVL tree of height $h$
    must have a lot of (i.e. $\Theta(2^h)$) nodes
- Easy to maintain
  - Using single and double rotations
The AVL Tree Data Structure

Structural properties:
1. Binary tree property
2. Balance property: balance of every node is between -1 and 1
Result:
   Worst case depth is $\Theta(\log n)$

Ordering property:
   - Same as for BST

Proving Shallowness Bound

Let $S(h)$ be the min # of nodes in an AVL tree of height $h$
Claim: $S(h) = S(h-1) + S(h-2) + 1$
Solution of recurrence: $S(h) = \Theta(2^n)$
   (like Fibonacci numbers)

Testing the Balance Property

We need to be able to:
1.  
2.  
3.  

$\text{NULLs have height } -1$

An AVL Tree

Beautiful Balance

Insert(mid)
Insert(small)
Insert(large)

Bad Case #1

Insert(large)
Insert(mid)
Insert(small)
General Single Rotation

• Are the blue and red heights the only possibilities? Yes!
• Height of this whole tree same as it was before insert!
• Height of all ancestors unchanged. So?

General Double Rotation

• Are the blue and red heights the only possibilities? Yes!
• Height of subtree still the same as it was before insert!

Great! Now we can fix imbalance!

• Single rotation for the “zig-zig” case
• Double rotation for the “zig-zag” case

Both rotations keep the subtree height unchanged. Hence only one rotation is sufficient!
So what does AVL mean anyway??

* Let’s vote!!*

- Automatically Virtually Levelled
- Architecture for inVisible Leveling (the “in” is inVisible)
- All Very Low
- Absolut Vodka Logarithms
- Amazingly Vexing Letters

AVL Tree Operations

- Find(x)
- Insert(x)
- Delete(x)
- buildTree

Θ(log n)

Θ(n log n)

Insertion into AVL tree

1. Find spot for new key
2. Hang new node there with this key
3. Search back up the path for imbalance
4. If there is an imbalance:
   - case #1: Perform single rotation and exit
   - case #2: Perform double rotation and exit

Should we loop to fix all problems?

Easy Insert

Insert(3)

Unbalanced?

Hard Insert (Bad Case #1)

Insert(33)

Unbalanced?

How to fix?

How did we know?

Single Rotation
Hard Insert (Bad Case #2)

Unbalanced?
How to fix?
How did we know?

Single Rotation (oops!)

How did we know?
How to fix?

Double Rotation (Step #1)

Double Rotation (Step #2)

AVL Insert Algorithm Revisited

Recursive
1. Search downward for spot
2. Insert node
3. On the way back, correct 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?

Deletion in AVL Tree

Recall deletion in BST:
- What’s the order change in the tree?
  - Can this affect balance?
- What’s the structural change?
  - Can this affect balance?
**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);
}
```

**Double Rotation Completed**

**To Do**

- Written homework #1
- Read chapter 4