AVL Trees

Reasoning: With a normal binary search tree, the best case is $O(\log n)$. If we are inserting "n" elements & we follow this best case, then insertion time will take $n \log n$ time. However, this isn't always guaranteed. Take for example the following pseudocode:

```
for i from 0 to n:
    insert "i" to tree
```

With the insertion method used on a tree, this for-loop would iterate through the entire tree down to the leaf. This will cause our tree to devolve into a linked list - $O(n)$ traversal w/ $O(n)$ insertion elements will give us $O(n^2)$. So an AVL tree allows us to maintain a BST property while fixing this issue.

Terminology:

Balanced - All leaves are located at tree levels h & $h-1$.

Node Height - The longest edge path to reach a leaf from the current node.

Ancestor - All nodes on the path from the current node up to the root. All nodes are ancestors & descendants of themselves.

How it works:

The primary operation which drives the balancing is an rotation.

Each node keeps track of its height and something called the balance factor.

The balance factor is defined as:

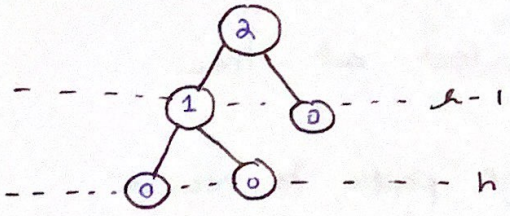
$$bf = h_{LS} - h_{RS}$$

Where h_{LS} is the height of the left subtree & h_{RS} is the height of the right subtree.

An imbalance occurs when "bf" is less than -1, not 0, or more than 1.

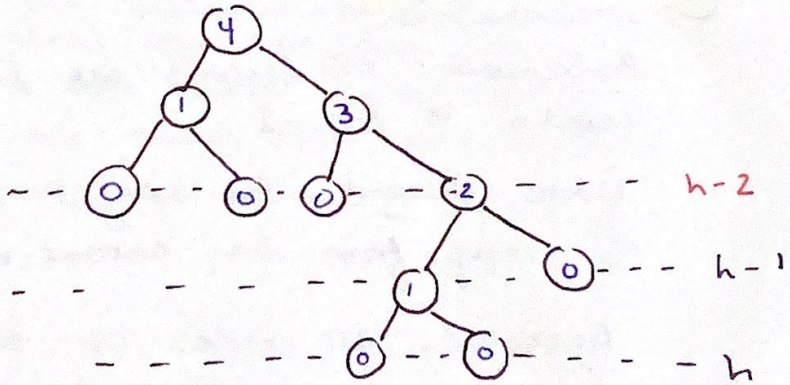
* Remember that a tree is balanced when leaves are at h & h-1; In other words, if the height differs by more than 1 (or less than -1), we have an extra level which indicates leaves (possibly) in more than one place. It's worth mentioning one more time that the leaves can only be at most one level apart.

(Balanced)



■ = node height

(Imbalanced)



■ = node height

Applying Rotations:

In general, rotations occur opposite to that of an imbalance.

Rotation Info:

Left Rotation - Left subtree of the new root becomes the right subtree of the old root (if not null).

Right Rotation - Right subtree of the new root becomes the left subtree of the old root (if not null).

When we insert a new node, we need to observe and fix any imbalances working our way up the tree to the root.

If there is an imbalance, there are two things which need observing:

1.) If the balance factor of the current node & its child are both the same sign (pos/neg) then only one rotation is needed to reverse the imbalance.

2.) If the balance factors are a different sign then two rotations need to take place; one to make them the same sign, and another to reverse the imbalance.

Which nodes get rotated?:

If it's a double rotation, we first need to rotate around the child to make the signs match.

If it's a single rotation, we rotate around the root, opposite to the direction of the imbalance, to reverse it.

Which way do we rotate?

It's as simple as knowing the direction of the imbalance. Remember the balance factor and how it applies here:

$$b.f. = h_{LS} - h_{RS}$$

If we have a negative imbalance, then it means that the height is larger to the right. If we have a positive imbalance, then the height is larger to the left.

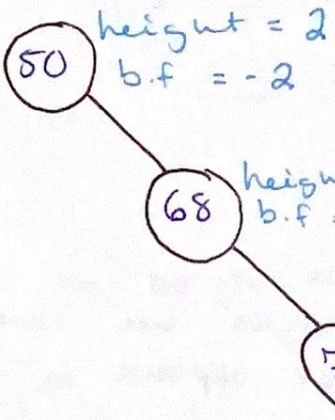
→ For a single rotation (i.e. the signs are the same), you rotate opposite to the imbalance; negative = left rotate, positive = right rotate.

→ For a double rotation, we target the correct subtree based on the imbalance sign. Since they are different, here is the general idea:

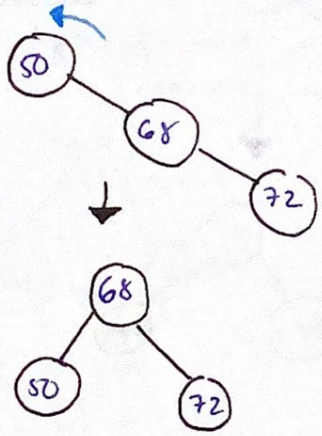
imbalance = 2 (child is negative), then we rotate the child left to make the signs match.

imbalance = -2 (child is positive) then we rotate the child right to make the signs match.

Example #1 - Single Rotation

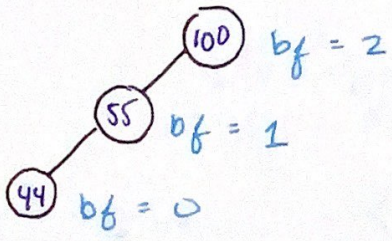


The imbalance is negative, so the child I choose is to the right. The signs are the same, so I rotate opposite to the imbalance.

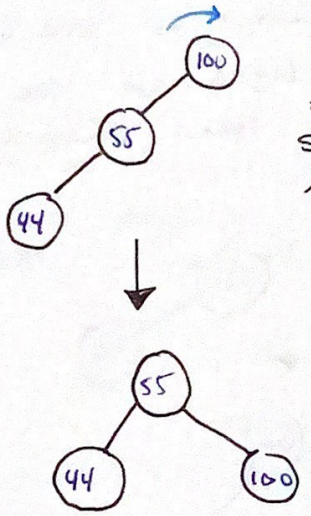


*left Rotation: left subtree of new root is right subtree of old root.

Example #2 - Single rotation:

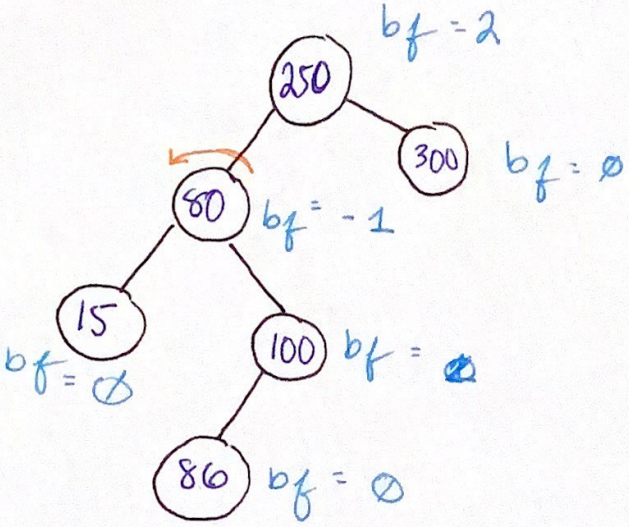


Here the imbalance is positive, so we know it's to the left. Since the signs are the same, we rotate around the root opposite to the direction of the imbalance.

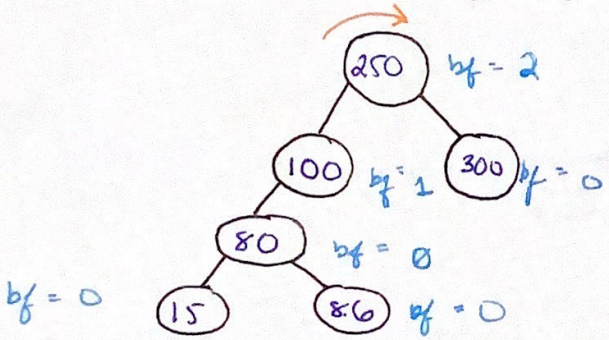


* Right Rotation: Right subtree of new root is left subtree of old root.

Example #1 - Double Rotation:

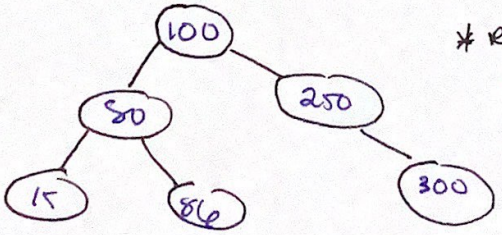


The first imbalance occurs at the root. We have a positive imbalance, so we target the left child to rotate around. The signs are different, and since the root is positive & the child is negative, we do a left rotation.



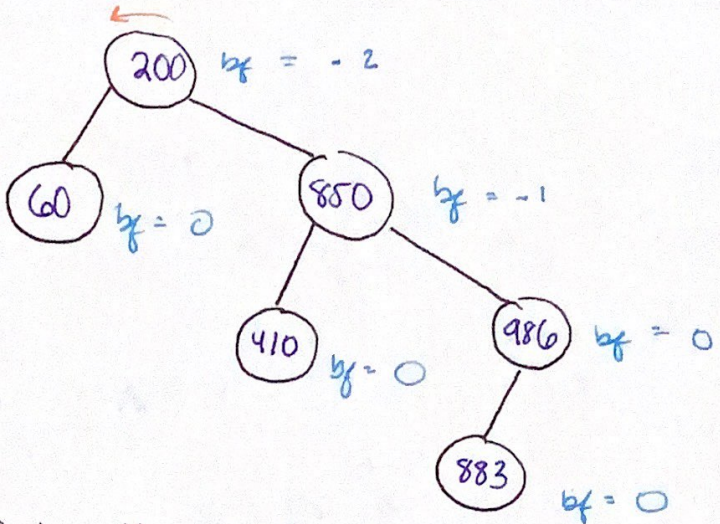
* Left Rotation: left subtree of new root is right subtree of old root.

Now the signs are the same so we rotate in the opposite direction of the imbalance.



* Right Rotation: Right subtree of new root is left subtree of old.

Example #2 - Double Rotation:



This situation is interesting but the rules still apply. Since the imbalance is negative, the imbalance occurs to the right. Hence, we rotate to the left of this.