**AVL TREES (a self-balancing tree)**

AVL 🡪 founders initials

We keep height difference in left subtree and right subtree 🡪 rightHeight - leftHeight

This difference can be:

* 1
* 0
* -1

Difference cannot be 2.

If balance gets ot of the range -1 to +1, the tree is rotated to bring it back into balance

Violation of balance can happen on the path root to the insertion point.  
Violation of balance can happen on the path root to the deletion point.

As items are added to or removed from a tree, the balance of each subtree from the insertion or removal point up to the root is updated.

Balancing a Left-Left Tree

Shape

Description automatically generated

Let’s assume we made an insertion in subtree a and assume that height of this subtree is increased h to h+1. Before insertion, height of a is h, height of b is h and balance of node 25 is 0.

Before insertion, height of left subtree of 50 is h+1 (a and b are h, 1 more for 25) and height of c (right subtree of 50) is h. Balance of 50 is 1. It is proper.

After the insertion, height of left subtree of 50 becomes h+2 (height of all tree becomes h+3). Balance of 50 becomes 2. It is not proper. We have to perform a rotation to the right.

The heights of the left and right subtrees are unimportant; only the relative difference matters when balancing.

When the root and left subtree are both left-heavy (as above), the tree is called a left-left tree  
Left-left tree can be balanced by a rotation right

A picture containing text, balalaika, psaltery, envelope

Description automatically generated

The overall height (h+2) has not increased.

Yani 50 bir root değil de daha büyük ağacın parçası olsaydı, 50’nin üzerindeki (insert edilen node’dan roota giden path üzerindeki nodelar) balance’lara bakmaya gerek kalmayacaktı. Tek bir rotation yeterli oldu.

How to pick the rotation?

Find the first node starting from inserted node to root node whose balance is violated.

Balancing a Left-Right Tree

Shape

Description automatically generated

Height = h+2

a, b, c all have height h.

A single rotation cannot be enough.

Subtree b needs to be expanded into its subtrees bL and bR.

Diagram

Description automatically generated with medium confidence

bL and bR are both h-1. After insertion, left subtree of 40 becomes h.

Diagram

Description automatically generated

The overall tree is now left-left and a rotation right will balance it.

Shape

Description automatically generated with medium confidence

Height = h+2

Insertion may be on bR instead of bL:

A picture containing text, clock

Description automatically generated

Shape

Description automatically generated

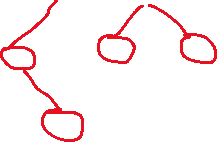
Shape

Description automatically generated with low confidence



Balance property has to be satisfied for each node.

So this tree is not balanced bc blue node has 2 as balance value.

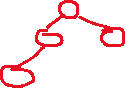


Best case for AVL tree is perfect binary tree

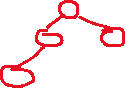
If height is h but number of nodes is minimum 🡪 worst case for AVL tree. It requires “h” amount of time to search but n (minimum number of elements) elements are kept.

Examples:

h = 4



h = 3



h = 2

h = 1



As you see for worst case for height = 4, we use h=3 worst case and h=2 worst case as left and right subtrees.

We can do same for h=5 (h=4 worst case as left subtree, h=3 worst case as right subtree).

Height is still logn for the worst case AVL trees.

Four Kinds of Critically Unbalanced Trees

1. Left-Left (parent balance is -2, left child balance is -1)
   1. Rotate right around parent
2. Left-Right (parent balance -2, left child balance +1)
   1. Rotate left around child
   2. Rotate right around parent
3. Right-Right (parent balance +2, right child balance +1)
   1. Rotate left around parent
4. Right-Left (parent balance +2, right child balance -1)
   1. Rotate right around child
   2. Rotate left around parent

**EXAMPLE:**

Build an AVL tree from the words in : “The quick brown fox jumps over the lazy dog”

A picture containing text

Description automatically generatedWe add “The” and “quick” without any problem.

When we add brown, balance property is broken. Starting from inserted node: is there a problem with “brown”? No bc its balance is 0. Is there a problem with “quick”? No bc its balance is -1 (right-left). Is there a problem with “The”? Yes. Its balance is +2 (Look Right-Left case above). We have to rotate right around “quick” and rotate left around “The”.

Rotate right around “quick”: Rotate left around “The”:

Chart, line chart

Description automatically generated Line chart

Description automatically generated

Diagram, radar chart

Description automatically generated with medium confidenceThere is no problem when we insert the fox.

Diagram

Description automatically generatedWe have problem at “quick” (-2) and “brown” (+2). The nearest node to the new inserted node is “quick” so when we fix it, problem at “brown” will also be fixed.



Situation is left-right.



Rotate left around “fox”: Rotate right around “quick”:

Diagram

Description automatically generated Diagram

Description automatically generated



Diagram

Description automatically generatedRight-right case at “brown”.

A picture containing text, watch

Description automatically generated

A picture containing text, watch, gauge

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

**Implementing an AVL Tree**

Diagram

Description automatically generated

“-increase” keeps whether the height is increased or not throughout the recursive calls.

We didn’t override contains and find methods bc AVLTree can be searched as it is bst, actually it is a bst.

**We didn’t override remove bc we call delete method in remove method. This is property of polymorphism. If the tree is AVLTree, AVLTree’s delete method will be called. This is decided during runtime (late-binding).**

Text, letter

Description automatically generated

AVLNode Class

Text, letter

Description automatically generated

Inserting into an AVLTree

The easiest way to keep a tree balanced is never to let it become unbalanced

If any node becomes critical, rebalance immediately

Identify critical nodes by checking the balance at the root node as you return along the insertion path

Text

Description automatically generated

We perform recursive call until insertion position, number of nodes on this path is at most height of the tree and height of the tree is logarithmic. So overall running time is logarithmic.

Text, letter

Description automatically generatedText

Description automatically generated Graphical user interface, text, application, email

Description automatically generated

Symmetric for data > 0 (lefts are right, we incrementBalance, if balance>1, we perform rebalanceRight.

Graphical user interface, text

Description automatically generated

Effect of Rotations on Balance

The rebalance algorithm above is incomplete as the balance of the nodes was not adjusted

For a left-left tree the balances of the new root node and of its right chid are 0 after a right rotation

Left-right is more complicated:

* the balance of the root is 0

If the critically unbalanced situation was due to an insertion into

* subtree bL (left-right-left case), the balance of the root’s left child is 0 and the balance of the root’s right child is +1

A picture containing scissors

Description automatically generated

* subtree bR (left-right-right case), the balance of the root’s left child is -1 and the balance of the root’s right child is 0

Shape, polygon

Description automatically generated

Text, letter

Description automatically generated

A black and white document

Description automatically generated with medium confidence

rebalanceRight is symmetric.

decrementBalance()

As we return from an insertion into a node’s left subtree, we need to decrement the balance of node

Shape

Description automatically generatedWe also need to indicate if the subtree height at that node has not increased (setting increase to false)

2 cases to consider:

* a balanced node --> insertion into its left subtree will make it left-heavy and its height will increase by 1
* a right-heavy node -->insertion into its left subtree will cause it to become balanced and its height will not increase

Text, application

Description automatically generated

Removal from an AVL Tree

Removal:

* from a left subtree, increases the balance of the local root
* from a right subtree, decreases the balance of the local root

The bst removal method can be adapted for removal from an AVL tree

A data field decrease tells the previous level in the recursion that there was a decrease in the height of the subtree from which the return occurred

The local root balance is incremented or decremented based on this field

If the balance is outside the threshold, a rebalance method is called to restore balance

Methods decrementBalance(), incrementBalance(), rebalanceLeft(), and rebalanceRight() need to be modified to set the value of decrease and increase after a node’s balance is decremented

Each recursive return can result in a further need to rebalance

Performance of the AVL Tree

Since each subtree is kept as close to balanced as possible, the AVL has expected O(logn)

Each subtree is allowed to be out of balance 1 so the tree may contain some holes

In the worst case (which is rare) an AVL tree can be 1.44 times the height of a complete binary tree that contains the same number of items

Ignoring constants, this still yields O(logn) performance

Empirical tests show that on average logn+0.25 comparisons are required to insert the nth item into an AVL tree -- close to a corresponding complete bst