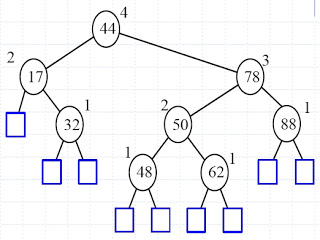
[](http://2.bp.blogspot.com/-ORwQVmYDupQ/UMNWoukI3wI/AAAAAAAAAG8/rvmy0NYF94Q/s1600/avl01.jpg)

<http://en.wikipedia.org/wiki/AVL_tree>

# AVL tree

In [computer science](http://en.wikipedia.org/wiki/Computer_science), an **AVL tree** is a [self-balancing binary search tree](http://en.wikipedia.org/wiki/Self-balancing_binary_search_tree), and it was the first such [data structure](http://en.wikipedia.org/wiki/Data_structure) to be invented.[[1]](http://en.wikipedia.org/wiki/AVL_tree#cite_note-1) In an AVL tree, the[heights](http://en.wikipedia.org/wiki/Tree_height) of the two [child](http://en.wikipedia.org/wiki/Child_node) subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Lookup, insertion, and deletion all take [O](http://en.wikipedia.org/wiki/Big_O_notation)(log *n*) time in both the average and worst cases, where n is the number of nodes in the tree prior to the operation. Insertions and deletions may require the tree to be rebalanced by one or more [tree rotations](http://en.wikipedia.org/wiki/Tree_rotation).

The AVL tree is named after its two [Soviet](http://en.wikipedia.org/wiki/Soviet_Union) inventors, [G. M. Adelson-Velskii](http://en.wikipedia.org/wiki/Georgii_Adelson-Velskii) and [E. M. Landis](http://en.wikipedia.org/wiki/Yevgeniy_Landis), who published it in their 1962 paper "An algorithm for the organization of information".[[2]](http://en.wikipedia.org/wiki/AVL_tree#cite_note-2)

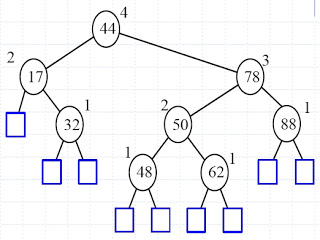
AVL trees are often compared with [red-black trees](http://en.wikipedia.org/wiki/Red-black_tree) because they support the same set of operations and because red-black trees also take [O](http://en.wikipedia.org/wiki/Big_O_notation)(log *n*) time for the basic operations. Because AVL trees are more rigidly balanced, they are faster than red-black trees for lookup-intensive applications.[[3]](http://en.wikipedia.org/wiki/AVL_tree#cite_note-3)[[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed)] Similar to red-black trees, AVL trees are height-balanced, but in general not [weight-balanced](http://en.wikipedia.org/wiki/Weight-balanced_tree) nor μ-balanced;[[4]](http://en.wikipedia.org/wiki/AVL_tree#cite_note-4) that is, sibling nodes can have hugely differing numbers of descendants.

<http://csgate2013.blogspot.com/2012/12/avl-trees.html>

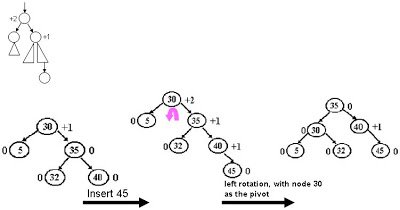
**SATURDAY, 8 DECEMBER 2012**

AVL Trees

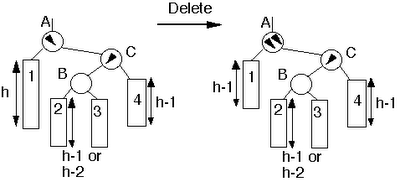
AVL Trees  
===============  
- Also called height balanced tree  
-Height of children present at same level may not have same height. There height can differ at most by 1.  
-The height of an AVL tree T storing n keys is O(log n)  
  
Structure of AVL Tree  
========================  
- Consoder an AVL tree of n nodes  
-Cosider a leaf which is closest to the root  
- Suppose this leaf is at level k.  
-Then height of a tree is at most 2k - 1.

[](http://2.bp.blogspot.com/-ORwQVmYDupQ/UMNWoukI3wI/AAAAAAAAAG8/rvmy0NYF94Q/s1600/avl01.jpg)

Summary of AVL Tree  
==========================  
- If height is h , then leaf closest to root is at level  at least (h+1)/2  
-On the first (h-1)/2 levels the AVL tree is a complete binary tree  
- After (h-1)/2 level the AVL tree may start "thinning out"  
-Number of nodes in AVL tree is at least 2 to raise (h-1)/2 and at most 2 raise to h  
  
Insertion in AVL Tree  
==========================  
Insertion of node make tree changes height and hence height balance property voilated.  
  
If Insertion causes T to become unbalanced , we travel up the tree from the newly created node until we find first node x such that its grandparent z is unbalanced node.(IMP)  
  
-Rotation if performed to make balanced  
even after rotation height of tree remains same  
-Middle node becomes root after rotation

[](http://1.bp.blogspot.com/-px2v6q9MwuQ/UMNXRxylavI/AAAAAAAAAHE/oCxzgHMO4lE/s1600/insertavl.jpg)

Deletion  
=============================  
-Let w be a node to be deleted  
-Let z be the firt unbalanced node encountered while travelling up the  tree from w . Also let x be the child of y with a larger height .  
-We perform rotation to restore balance at subtree rooted at z  
-As this restructring may upset the balance of another node heigher in the tree , we must continue checking for balance until the root of T is reached.

[](http://4.bp.blogspot.com/-lKbx4Xmbj1o/UMNYIEZtEEI/AAAAAAAAAHM/bt-wOr4_2dQ/s1600/deletionavl.gif)