

AVL Trees

In order to have a worst case running time for insert and delete operations to be $O(\log n)$, we must make it impossible for there to be a very long path in the binary search tree. The first balanced binary tree is the AVL tree, named after its inventors, Adelson-Velskii and Landis. A binary search tree is an AVL tree iff each node in the tree satisfies the following property:

The height of the left subtree can differ from the height of the right subtree by at most 1.

Based on this property, we can show that the height of an AVL tree is logarithmic with respect to the number of nodes stored in the tree.

In particular, for an AVL tree of height H , we find that it must contain at least $F_{H+3} - 1$ nodes. (F_i is the i th Fibonacci number.) To prove this, notice that the number of nodes in an AVL tree is the 1 plus the number of nodes in the left subtree plus the number of nodes in the right subtree. If we let S_H represent the minimum number of nodes in an AVL tree with height H , we get the following recurrence relation:

$$S_H = S_{H-1} + S_{H-2} + 1$$

We also know that $S_0=1$ and $S_1=2$. Now we can prove the assertion above through induction.

Problem: Prove that $S_H = F_{H+3} - 1$.

We will use induction on H, the height of the AVL tree.

Base Cases H=0: LHS = 1, RHS = $F_3 - 1 = 2 - 1 = 1$

H=1: LHS = 2, RHS = $F_4 - 1 = 3 - 1 = 2$

Inductive hypothesis: For an arbitrary integer $k \leq H$, assume that $S_k = F_{k+3} - 1$.

Inductive step: Under the assumption above, prove for $H=k+1$ that $S_{k+1} = F_{k+1+3} - 1$.

$$\begin{aligned} S_{k+1} &= S_k + S_{k-1} + 1 \\ &= (F_{k+3} - 1) + (F_{k+2} - 1) + 1, \text{ using the I.H. twice} \\ &= (F_{k+3} + F_{k+2}) - 1 \\ &= F_{k+4} - 1, \text{ using the defn. of Fibonacci numbers, to} \\ &\text{complete} \\ &\text{proof.} \end{aligned}$$

It can be shown through recurrence relations, that

$$F_n \approx 1/\sqrt{5} [(1 + \sqrt{5})/2]^n$$

So now, we have the following:

$$S_n \approx 1/\sqrt{5} [(1 + \sqrt{5})/2]^{n+3}$$

This says that when the height of an AVL tree is n, the minimum number of nodes it contains is $1/\sqrt{5} [(1 + \sqrt{5})/2]^{n+3}$.

So, in order to find the height of a tree with n nodes, we must replace S_n with n and replace n with h? Why is this the case?

$$n \approx 1/\sqrt{5} [(1 + \sqrt{5})/2]^{h+3}$$

$$n \approx (1.618)^h$$

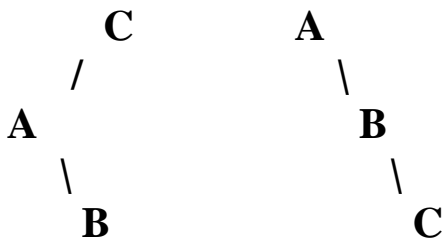
$$h \approx \log_{1.618} n$$

$$h = O(\log_2 n)$$

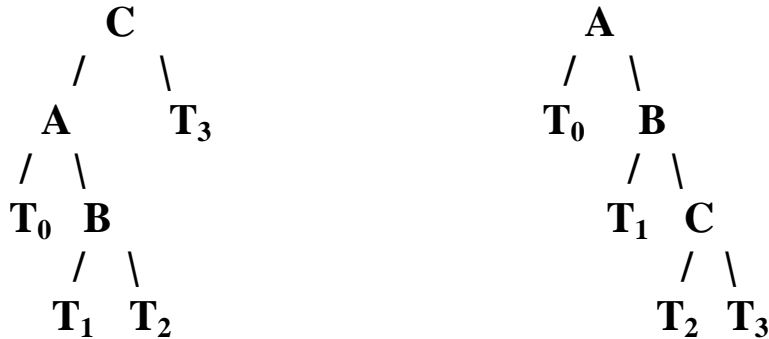
Now the question remains, how do we maintain an AVL tree? What extra work do we have to do to make sure that the AVL property is maintained?

Basically whenever an insertion or deletion is done, it is possible that the new node added or taken away destroys the AVL property. In these situations, we have to "rework" the tree so that the binary search tree and AVL properties are satisfied.

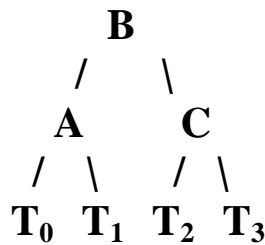
When an imbalance is introduced to a tree, it is localized to three nodes and their four subtrees. Denote these three nodes as A, B, and C, in their inorder listing. Structurally, they may appear in various configurations. A couple of these are listed below:



Denote the four subtrees as T_0 , T_1 , T_2 , and T_3 , also listed in their inorder listing. Here is where these would lie in the trees drawn above:



No matter which of these structural imbalances exist, they can all be fixed the same way:



Another way we can view these transformations is through two separate types or restructuring operations: a single rotation and a double rotation.

Let's look at how both of these work.

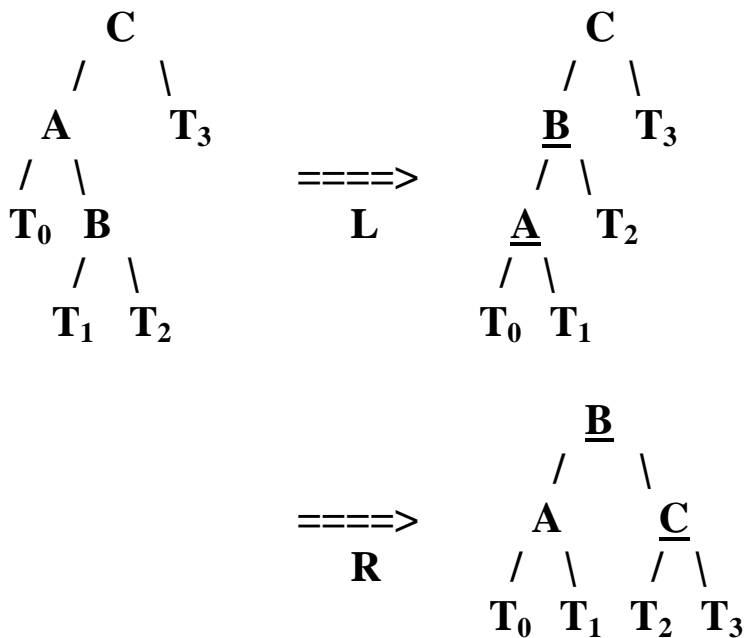
Here are the four cases we will look at:

- 1) insertion into the left subtree of the left child of the root.
- 2) insertion into the right subtree of the left child of the root.
- 3) insertion into the left subtree of the right child of the root.
- 4) insertion into the right subtree of the right child of the root.

Technically speaking, cases 1 and 4 are symmetric as are 2 and 3.

For cases 1 and 4, we will perform a single rotation, and for 2 and 3 we will do a double rotation.

In the pictures I have above, the left picture is case 2 of this description, and the right picture is case 4. Why is the case on the left called a double rotation? Because we can achieve it by performing two rotations on the root node:



Insertion into an AVL Tree

So, now the question is, how can we use these rotations to actually perform an insert on an AVL tree?

Here are the basic steps involved:

- 1) Do a normal binary tree insert.**
- 2) Restoring the tree based on this leaf node.**

This restoration is more difficult than just following the steps above. Here are the steps involved in the restoration of a node:

- 1) Calculate the heights of the left and right subtrees, use this to set the potentially new height of the node.**
- 2) If they are within one of each other, recursively restore the parent node.**
- 3) If not, then perform the appropriate restructuring described above on that particular node, THEN recursively call the method on the appropriate parent node.**

Note: No recursive call is made if the node in question is the root node and has no parents.

You'll notice that the code in the book has no recursion in the rebalance method. None is needed, but I felt the explanation of the algorithm was more straightforward with recursion.