Show Course Contents

Overview Q&A Downloads Announcements

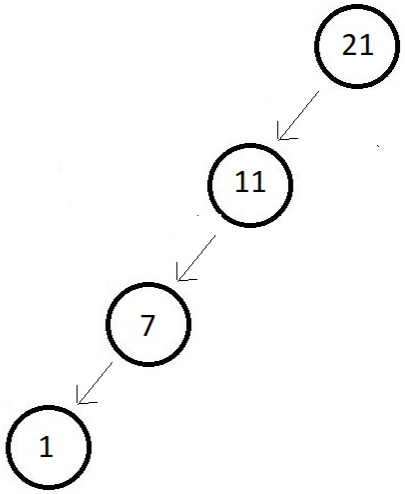
AVL Trees - Introduction

In the last lecture, we finished learning our operations on a binary search tree. We learned to search, insert, and all the cases of deletion in a binary search tree. Today, we'll start learning about the AVL trees.

Well, the operations we have discussed lately have been observed to work faster when the tree is highly distributed, or where the height is actually lose to logn and the complexity tends to O(logn), but they come close to O(n) when our tree becomes sparse, and looks unnecessarily lengthened. This is where AVL trees come to the rescue. I'll explain more on why AVL trees are considered a different topic worth teaching and why we even need AVL trees.

Why AVL trees?

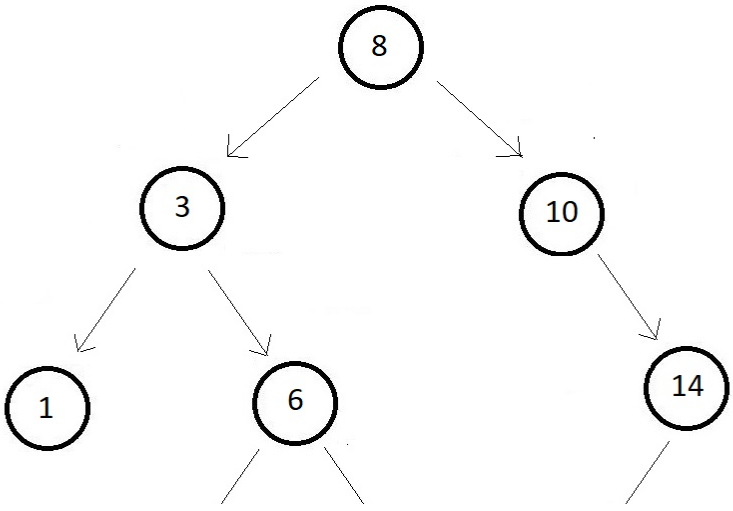
I'll take an example to help you all understand the concept. Suppose you are told to create a Binary Search tree out of some elements given to you. Suppose the numbers were {1, 11, 7, 21}. And to avoid any labour, you simply sort the elements and write them as shown in the figure below.

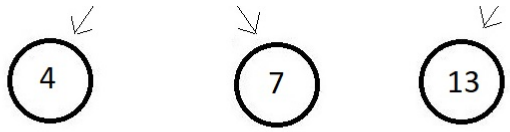


And to be honest, you cannot deny the fact that this is indeed a binary search tree, though skewed. There is no violation of any of the rules of a binary search tree. But any operation here has a complexity O(n). But I can guarantee, you expected something more balanced, something more dense. Well, that is what an AVL tree is.
An AVL tree is needed because:

1. Almost all the operations in a binary search tree are of order O(h) where h is the height of the tree.
2. If we don't plan our trees properly, this height can get as high as n where n is the number of nodes in the Binary Search Tree (Skewed tree).
3. So, to guarantee an upper bound of O(logn) for all these operations we use balanced trees.

This is actually very practical. Because when a binary search tree takes the form of a list, our operations, say searching, starts taking more time. Consider the Binary search tree below.

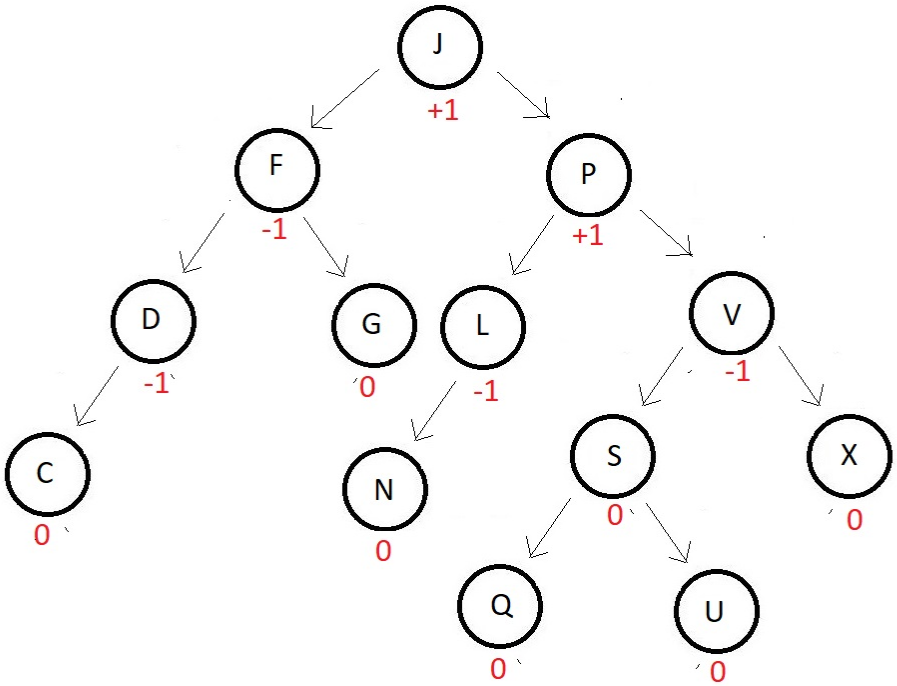




To search 1, in this binary search tree, you would need only 3 operations, while to search in a list type binary search tree having the same elements, this would have taken 9 operations.

What are AVL trees?

1. AVL trees are height balanced binary search trees. Because most of the operations work on O(h), we would want the value of h to be minimum possible, which is log(n).
2. Height difference between the left and the right subtrees is less than 1 or equal in an AVL tree.
3. For AVL trees, there is a balance factor BF, which is equal to the height of the left subtree subtracted from the height of the right subtree. If we consider the below binary search tree, you can see the balance factor mentioned beside each node. Carefully observe each of those.

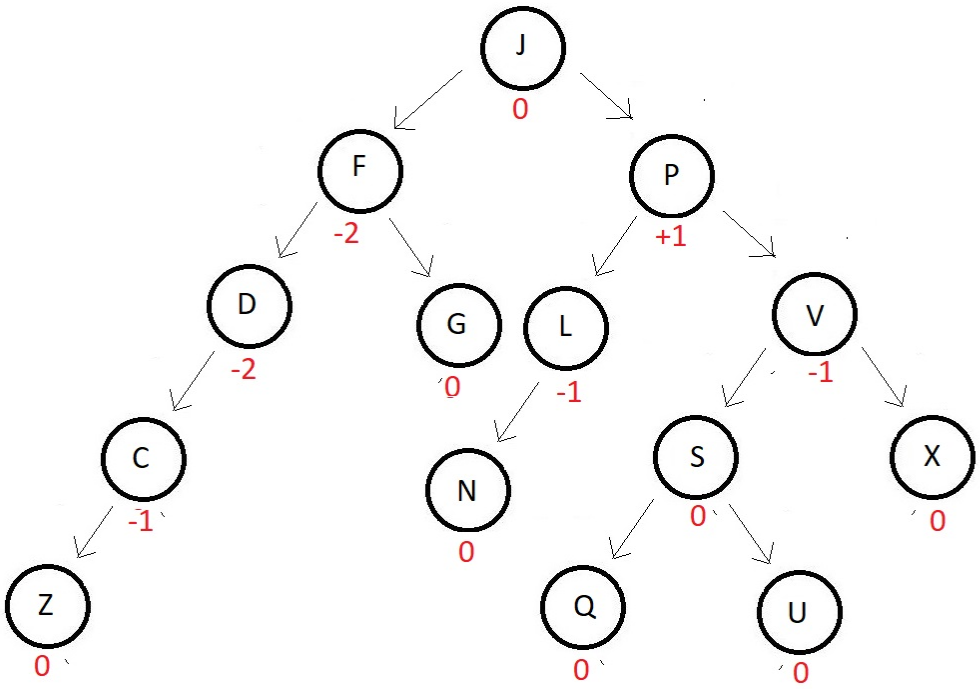


You can see, none of the nodes above have a balance factor more than 1 or less than -1. So, for a balance tree to be considered an AVL tree, the value of |BF| should

be less than or equal to 1 for each of the nodes, i.e., **|BF|≤ 1.**

4. And even if some of the nodes in a binary search tree have a |BF| less than or equal to 1, those nodes are considered balanced. And if all the nodes are balanced, it becomes an AVL.

One thing before we finish. An AVL tree gets disturbed sometime when we try inserting a new element in it. For example, in the above AVL tree, if we try inserting an element Z at the end of the leftmost element, the balanced factor gets updated for each of the nodes following above. And the tree is no more an AVL tree. See the updated tree below.



And to avoid this unbalancing, we have an operation called **rotation** in AVL trees. This helps maintain the balancing of nodes even after a new element gets inserted. I'll show you how in the next video.

So, this was all we had for today. We will see the operations, the insertion, and the rotation in the AVL tree in the next video. Take a list of elements and make a balanced tree out of it yourself, and find the balancing factor of each node. And keep practicing.

I appreciate your support throughout. I hope you enjoyed the tutorial. If you genuinely appreciate my work, please let your friends know about this course too. If

you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial where we'll see a few operations, **rotation, and insertion in AVL trees**. Till then keep coding.

Previous Next

CodeWithHarry