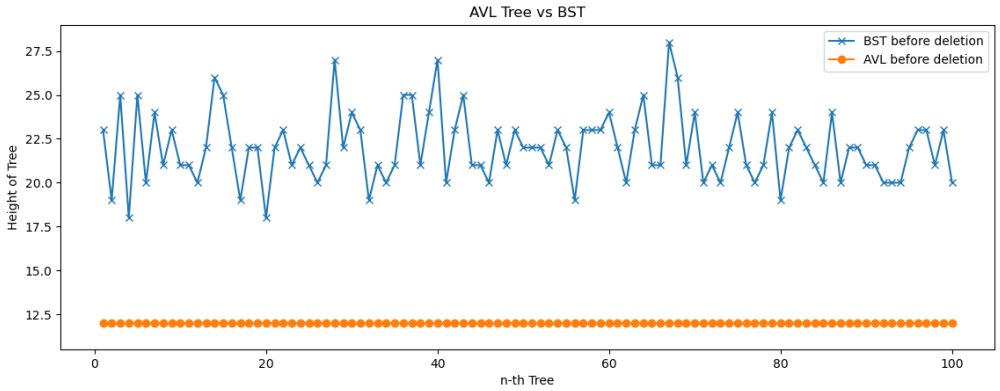
Binary search trees vs AVL trees

Introduction:

This write-up provides a comprehensive comparison between 100 BST trees and AVL trees with 1000 nodes in each tree, each node inserted in a sequential order from a set of 1-1000 numbers generated in random order. And then also compares the trees after deleting 100 nodes at random from all the 100 BST trees and 100 AVL trees.

Height Comparisons:

The first graph shows the heights of each tree in the set of 100 trees with 1000 nodes i.e before deletion of the 100 random nodes.

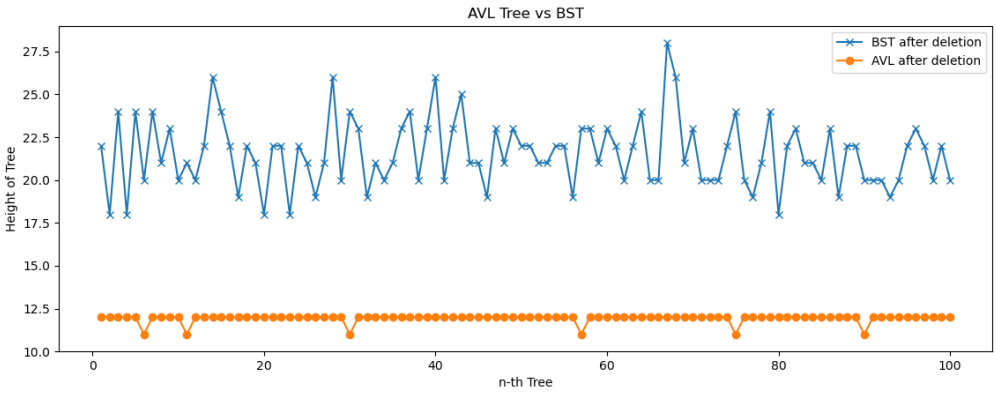


It is clear from the graph that heights of an AVL tree with same nodes have a much lower heights than the corresponding BST tree with same nodes inserted in same order.

It is also apparent from the graph that the likelihood of all AVL trees with same data in the nodes having the same height is very high but not in the case of BST trees as the height varies highly between 2 trees.

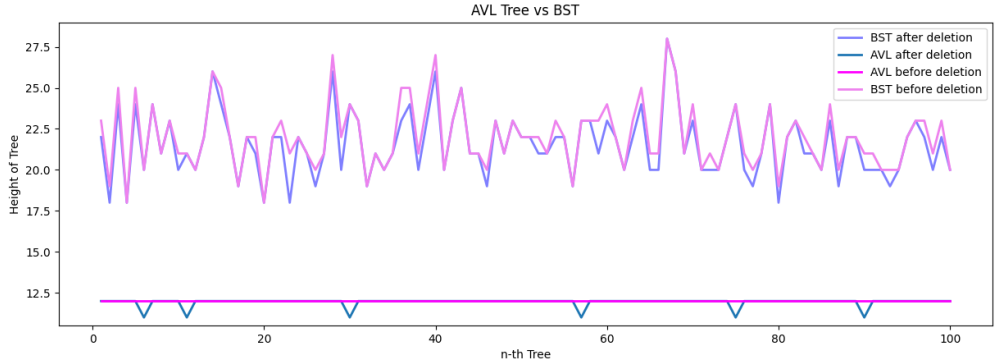
The height required to accommodate 1000 nodes in a perfect binary tree would be 10, here the height of AVL tree is 12 and the BST height varies from 18 to 27 which tells us that an AVL tree in general is much more balanced than a BST tree, meaning searching in an AVL is faster than in BST.

This graph shows the heights of each tree in the set of 100 trees with 900 nodes i.e after deletion of the 100 random nodes.



We can see similar trends as in the case of the first figure and the observations are similar as in the case of before deletion.

The graph shows all the 4 plots for the 200 trees i.e heights of both AVL and BST before and after deletion with the appropriate legend in the graph.



From this graph after deleting 100 nodes, we could see that there is much more volatility in heights of BST trees, but the AVL trees heights remained almost same with only a few trees heights dropping to 11 from 12. This tells that an AVL trees ability balance itself after very insertion or deletion makes its heights optimized so there won't be much change in height after a single operation compared to the 1000 nodes it has.

Operation Comparisons:

The bar graph shows that the number of operations done in the case of BST tree is around 10 lakhs where as in the case of AVL it is aroung 14 lakhs. This small difference when compared with 100 trees each with 1000

nodes is inconsiderable and when checking the work done for particular inputs, sometimes the BST tree had to do more work than AVL tree so making AVL tree computationally slightly slower but overall searching in an AVL is much faster than in a BST as the height difference will be much higher meaning in the worst case scenario the algorithm has to traverse to a much smaller height in an AVL tree, which in return means that deletion is slightly faster in AVL tree than in a BST.
Meanwhile when it comes to insertion as in the case of AVL it has to check for the balance factor or each parent node in a recursive manner and do necessary rotations the computation is slower, and the operations are more than in the case of BST where it has to just move to the correct position and insert the node. This difference becomes much smaller or even gets reversed if the inputs are in a skewed manner leading to a BST with a very high height.