DS. Sem 4th 29/4/2020 AVL trees

AVL tree is a self-balanced binary search tree. That means, an AVL tree is a binary search tree but it is a balanced tree. A binary tree is said to be balanced, if the difference between the heights of left and right subtrees of every node in the tree is either -1, 0 or +1. In other words, a binary tree is said to be balanced if for every node, height of its children differ by at most one. In an AVL tree, every node maintains an extra information known as balance factor to take care of the self-balancing nature of the tree.

An AVL tree is defined as follows…

An AVL tree is a balanced binary search tree. In an AVL tree, balance factor of every node is either -1, 0 or +1

Balance factor = height Of Left Subtree — height Of Right Subtree (OR) height Of Right Subtree — height Of Left Subtree

An AVL tree is given in the following figure. We can see that, balance factor associated with each node is in between the range of -1 to +1.

Every AVL Tree is a binary search tree but all the Binary Search Trees need not to be AVL trees.

Why AVL Tree ?

AVL tree controls the height of the binary search tree by not letting it to be skewed. The time taken for all operations in a binary search tree of height h is O(h). However, it can be extended to O(n) if the BST becomes skewed (i.e. worst case). By limiting this height to log n, AVL tree imposes an upper bound on each operation to be O(log n) where n is the number of nodes.

AVL Tree Rotations ?

In AVL tree, after performing every operation like insertion and deletion we need to check the balance factor of every node in the tree. If every node satisfies the balance factor condition then we conclude the operation otherwise we must make it balanced. We use rotation operations to make the tree balanced whenever the tree is becoming imbalanced due to any operation.

Rotation is the process of moving the nodes to either left or right to make tree balanced in terms of its height.

To balance itself, an AVL tree may perform the following four kinds of rotations −

1> Left rotation (Single)

2> Right rotation (Single)

3> Left-Right rotation (Double)

4> Right-Left rotation (Double)

1> Left Rotation (Single LL)

In LL Rotation every node moves one position to left from the current position. To understand LL Rotation, let us consider following insertion operations into an AVL Tree…

2> Right Rotation (Single RR)

In RR Rotation every node moves one position to right from the current position. To understand RR Rotation, let us consider following insertion operations into an AVL Tree…

3> Left Right Rotation (Double — LR Rotation)

The LR Rotation is combination of single left rotation followed by single right rotation. In LR Rotation, first every node moves one position to left then one position to right from the current position. To understand LR Rotation, let us consider following insertion operations into an AVL Tree…

4> Right Left Rotation (Double — RL Rotation)

The RL Rotation is combination of single right rotation followed by single left rotation. In RL Rotation, first every node moves one position to right then one position to left from the current position. To understand RL Rotation, let us consider following insertion operations into an AVL Tree…

Operations on AVL tree

Due to the fact that, AVL tree is also a binary search tree therefore, all the operations are performed in the same way as they are performed in a binary search tree. Searching and traversing do not lead to the violation in property of AVL tree. However, insertion and deletion are the operations which can violate this property.

1. Searching

Searching in an AVL Tree is done as in any binary search tree. The Special thing about AVL Tree is that the number of comparison required, i.e. The AVL three’s height, is guaranteed never to exceed log(n).

Step 1: Read the search element from the user.

Step 2: Compare, the search element with the value of root node in the tree.

Step 3: If both are matching, then display “Given node found!!!” and terminate the function.

Step 4: If both are not matching, then check whether search element is smaller or larger than that node value.

Step 5: If search element is smaller, then continue the search process in left subtree.

Step 6: If search element is larger, then continue the search process in right subtree.

Step 7: Repeat the same until we found exact element or we completed with a leaf node.

Step 8: If we reach to the node with search value, then display “Element is found” and terminate the function.

Step 9: If we reach to a leaf node and it is also not matching, then display “Element not found” and terminate the function.

2. Insertion

Insertion in AVL tree is performed in the same way as it is performed in a binary search tree. However, it may lead to violation in the AVL tree property and therefore the tree may need balancing. The tree can be balanced by applying rotations.

Step 1: Insert the new element into the tree using Binary Search Tree insertion logic.

Step 2: After insertion, check the Balance Factor of every node.

Step 3: If the Balance Factor of every node is 0 or 1 or -1 then go for next operation.

Step 4: If the Balance Factor of any node is other than 0 or 1 or -1 then tree is said to be imbalanced. Then perform the suitable Rotation to make it balanced. And go for next operation.

3. Deletion

Deletion can also be performed in the same way as it is performed in a binary search tree. Deletion may also disturb the balance of the tree therefore; various types of rotations are used to rebalance the tree.