Date : 20th - Oct- 2020
Morning Session : 9am – 11.00 PM
By ~ Rohan Kumar

Topics: AVL Tree & Dynamic Programing

AVL tree is a self-balancing binary search tree in which each node maintains extra information called a balance factor whose value is either -1, 0 or +1.

AVL tree got its name after its inventor Georgy Adelson-Velsky and Landis.

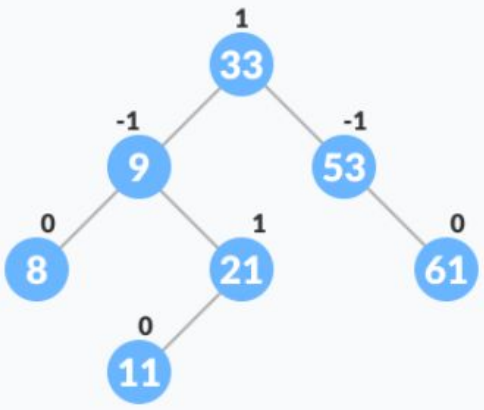
Balance Factor

Balance factor of a node in an AVL tree is the difference between the height of the left subtree and that of the right subtree of that node.

Balance Factor = (Height of Left Subtree - Height of Right Subtree) or (Height of Right Subtree - Height of Left Subtree)

The self balancing property of an avl tree is maintained by the balance factor. The value of balance factor should always be -1, 0 or +1.

An example of a balanced avl tree is:



Avl tree

AVL Tree Rotations 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 operations are used to make a tree balanced.There are four rotations and they are classified into two types:

Single Left Rotation (LL Rotation)

In LL Rotation every node moves one position to left from the current position.

Single Right Rotation (RR Rotation)

In RR Rotation every node moves one position to right from the current position.

Left Right Rotation (LR Rotation)

The LR Rotation is a 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.

Right Left Rotation (RL Rotation).

The RL Rotation is a 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.

Insertion in AVL Trees

Insert operation is almost the same as in simple binary search trees. After every insertion, we balance the height of the tree. Insert operation takes O(log n) worst time complexity.

Step 1:Insert the node in the AVL tree using the same insertion algorithm of BST.

Step 2:Once the node is added, the balance factor of each node is updated.

Step 3:Now check if any node violates the range of the balance factor if the balance factor is violated, then perform rotations using the below case.

Deletion in AVL Trees

Deletion is also very straight forward. We delete using the same logic as in simple binary search trees. After deletion, we restructure the tree, if needed, to maintain its balanced height.

Step 1: Find the element in the tree.

Step 2: Delete the node, as per the BST Deletion.

Step 3: Two cases are possible:-

Resource:

https://www.programiz.com/dsa/avl-tree

https://www.freecodecamp.org/news/avl-tree-insertion-rotation-and-balance-factor/

Dynamic Programing :

Please go through below pdf link for Dynamic Programing

Intro to Dynamic programing by Rohan Kumar

Other Resource:

https://www.freecodecamp.org/news/demystifying-dynamic-programming-3efafb8d4296/

MCQ's:

1. What is an AVL tree?
a) a tree which is balanced and is a height balanced tree
b) a tree which is unbalanced and is a height balanced tree
c) a tree with three children
d) a tree with at most 3 children

Answer: A

2. Why do we need a binary tree which is height balanced?
a) to avoid formation of skew trees
b) to save memory
c) to attain faster memory access
d) to simplify storing

Answer: A

3. What is the maximum height of an AVL tree with p nodes?

a) p

b) log(p)

c) log(p)/2

d) p/2

Answer: B

4. Which of the following is/are property/properties of a dynamic programming problem?

a) Optimal substructure

b) Overlapping subproblems

c) Greedy approach

d) Both optimal substructure and overlapping subproblems

Answer: D