Adelson-Velskii and Landis (AVL) Tree

AVL tree is a height-balanced binary tree with a $relaxed$ balance condition. A balance condition that requires left and right sub-tree of each node to have same height, is too rigid to be useful. AVL tree enforces a more relaxed balance condition. For example: difference in height of left and right sub-tree is at max 1. Height information is stored in each node and the balancing condition is enforced through variable h_factor.
h_factor = height(left sub-tree) - height(right sub-tree)

Height of an AVL tree is always $O(logN)$

Let A be an AVL tree with height h, containing minimum number of nodes $N(h)$ required for it to be an AVL. If the height of such worst-case AVL tree is bounded, then all AVL trees are bounded.

Without loss of generality, let's assume that the left sub-tree is larger than the right sub-tree. It follows, the height of left sub-tree is $h-1$ and the height of right sub-tree is $h-2$.
Thus,

$$N(0) = 1$$

$$N(1) = 2$$

$$N(h) = N(h-1) + N(h-2) + 1$$

Generating a lower bound on $N(h)$,

$$N(h) > N(h-2) + N(h-2) + 1 > 2N(h-2) + 1 > 2N(h-2)$$

$$N(h) > 2N(h-2)$$

Using recurrence,

$$N(h) > 2 \times 2 \times N(h-4) > 2 \times 2 \times 2 \times N(h-6)$$

$$N(h) > 2^{\frac{h}{2}}$$

Taking log,

$$logN(h) > \frac{h}{2}$$

$$h < 2logN(h)$$

Thus,

$$h = O(logN)$$

Hence, all operations on AVL tree are $O(logN)$.

Insertion in AVL tree

Inserting a new element in an AVL tree might change the h_factor to 2 or -2 and make it unbalanced. In order to re-balance the AVL, left and right rotations are performed. Here is an example showing the right rotation along with the strategy to update h_factor

Consider an AVL right rotation,

```
          A                   B
        /   \               /   \
      B       C    ===>   D       A
    /   \                       /   \
  D       E                   E       C
```

First thing to note is that only A and B's h_factor will change after rotation. Recalculating the heights of left and right sub-trees for A and B node would be too expensive. A faster method for updating h_factor is developed below.

Let,
old_A = h_factor of A before rotation
new_A = h_factor of A after rotation
h_B = height of sub-tree with root node B before rotation

$old_A = h_B - h_C$
$new_A = h_E - h_C$

$new_A - old_A = h_E - h_B$
$h_B = 1 + max(h_D, h_E)$

$new_A - old_A = h_E - 1 - max(h_D, h_E)$
$new_A - old_A = -1 - (max(h_D - h_E, h_E - h_E) = -1 - max(h_D - h_E, 0)$

Thus the updated h_factor of node A is given by,

$$new_A = old_A - 1 - max(old_B, 0)$$