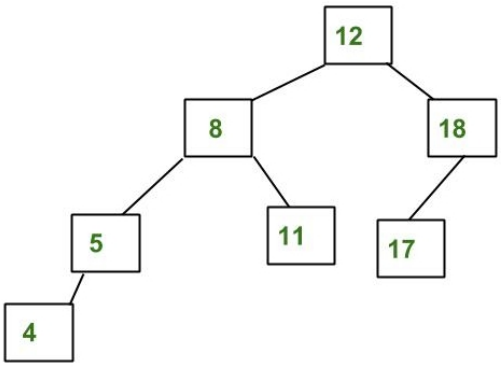
## AVL Trees

AVL trees (after authors: Adelson-Velsky and Landis) are balanced BST, in wchich the height of the left and right sub-tree of each node differ by at most 1.

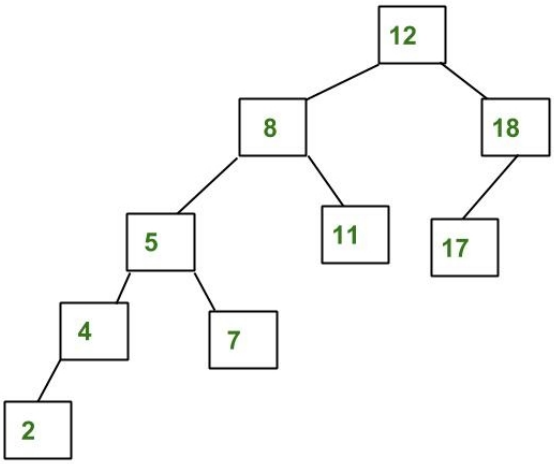
Animation: `https://visualgo.net/en/bst`.

We balance the tree by assigning to each node a *balance factor* which is equal to the difference between the height of the left and right sub-tree. It can be 0, -1 or +1. When inserting or deleting a node from the AVL tree, we also modify the balance factor, and when it gets a forbidden value (outside the set {-1, 0, +1}), we perform special rotations to bring back the balance.

An example AVL tree:




A BST tree that is not a correct AVL tree:



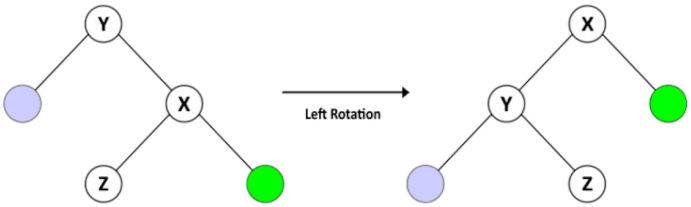
### Why do we want the balance?

Most of the operations on BST trees (e. searching, min, max, inserting, etc.) has complexity $O(h)$, where $h$ is the height of the tree. If a tree is highly unbalanced (more like a list than a tree...), the cost of these operations goes up to $O(n)$. If the height will be kept balanced, we are sure that all these operations will be in $O(log\, n)$, where $n$ is the number of nodes in the tree.

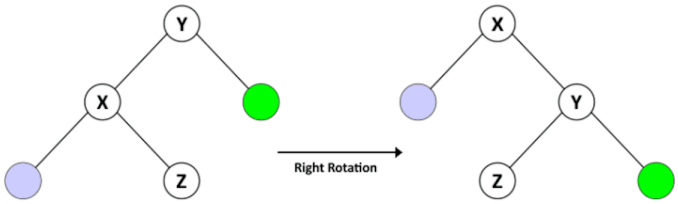
## Basic rotations in AVL tree

To make sure the tree is balanced after each insertion, we extend the standard insert operation by performing additional balancing of the tree. Below are the two basic operations that should be executed in order to balance the tree without changing is BST features:

- left rotate




- right rotate



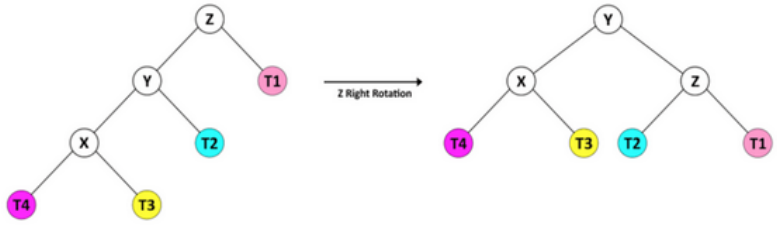

## Problem 1 (1 point)

Write down the pseudocode for LEFT-ROTATE(y) and RIGHT-ROTATE(y) operations, where y is a root of a subtree.

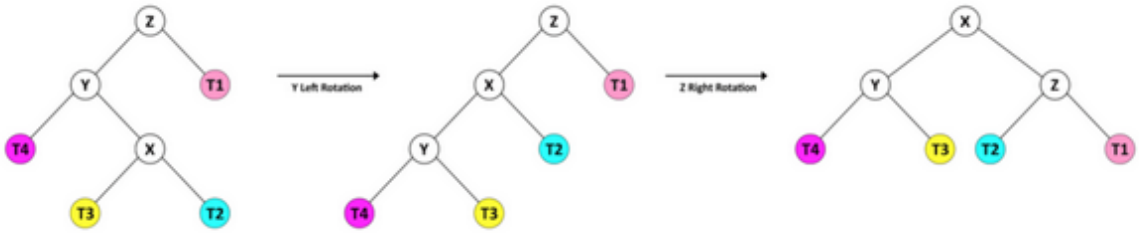
## Advance rotations in AVL trees

We perform rotations when we encounter in and AVL tree a node with the balance factor equal to +2 or -2. Such a value indicates that the tree is inbalanced, i.e., its subtrees' heights differ by 2. The (sequence of) rotations brings back the balance to the tree.

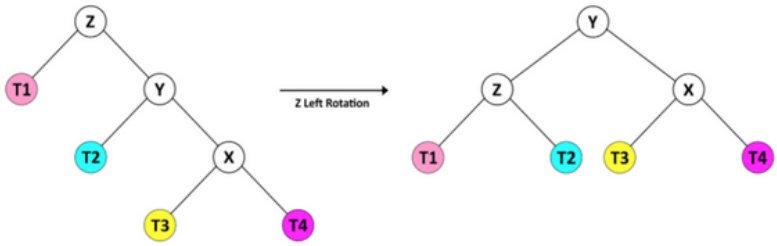
- Single right rotation (case LL):



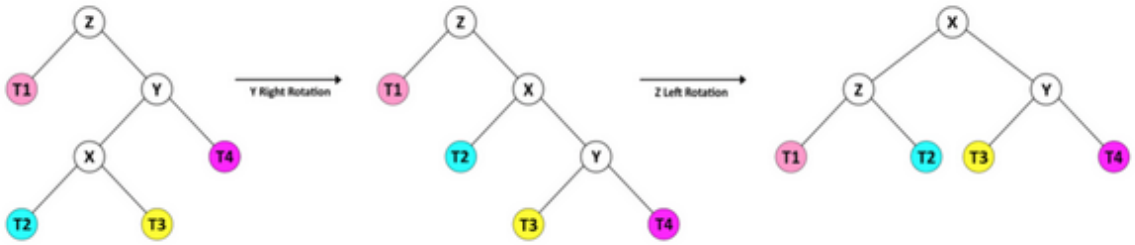

- Double rotation left-right (case LR):




- Single left rotation (case RR):




- Double rotation right-left (case RL):

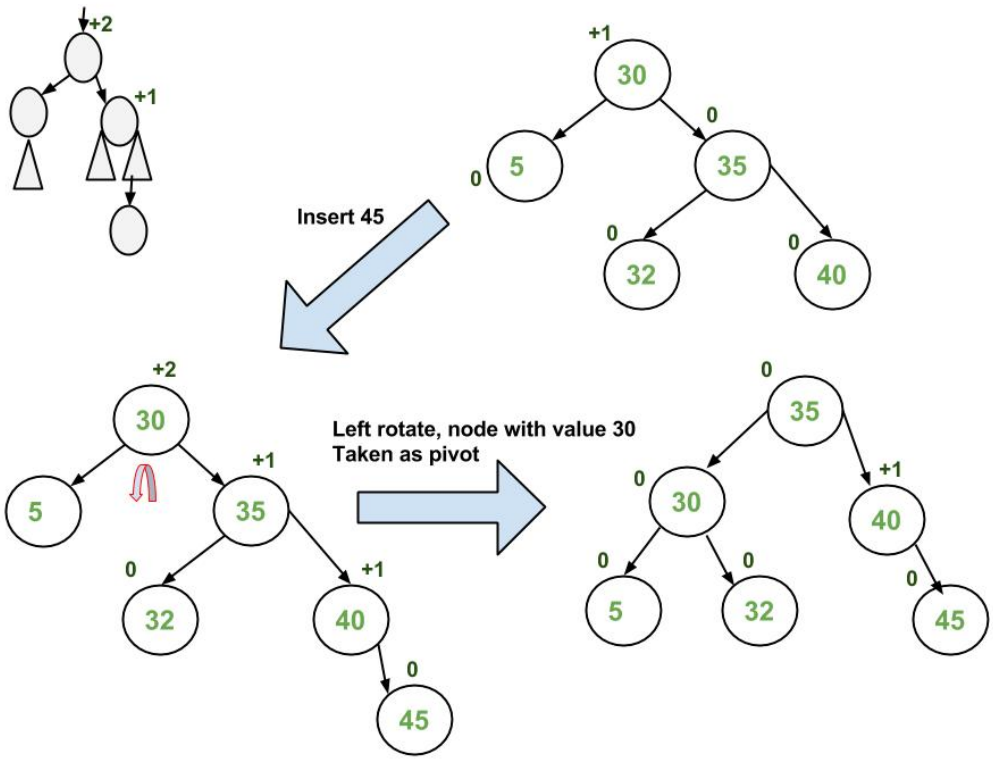



## Problem 2 (2 points)

Write down the pseudocode for REBALANCE(y) operation, where y is a root of a subtree.

## Inserting elements into AVL tree

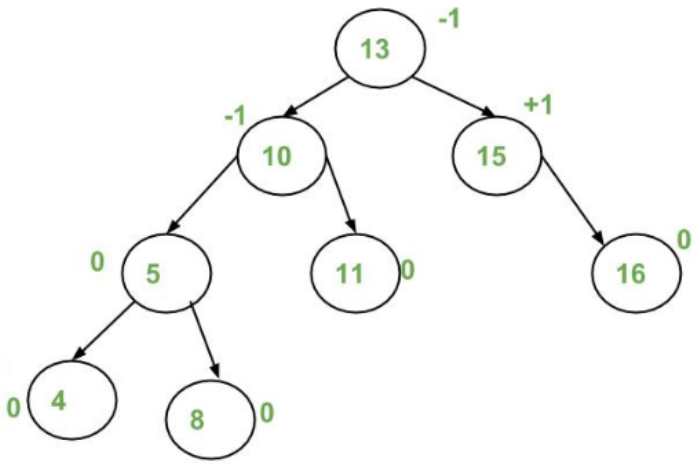
- Insert the element as in BST
- Check the balance factors and balance the tree if needed (going up from the inserted node to the root)



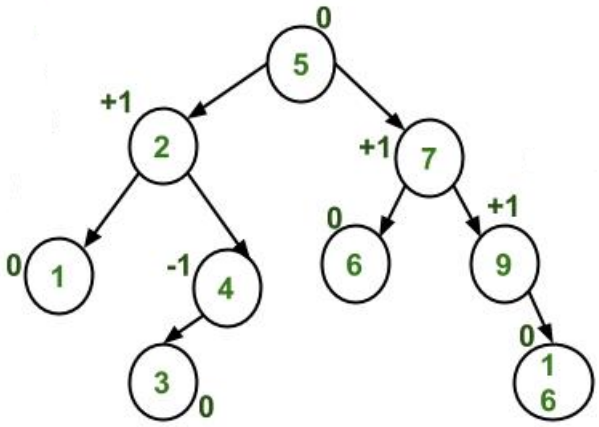

## Problem 3 (1 point)

Show how the elements are inserted in to the AVL tree:

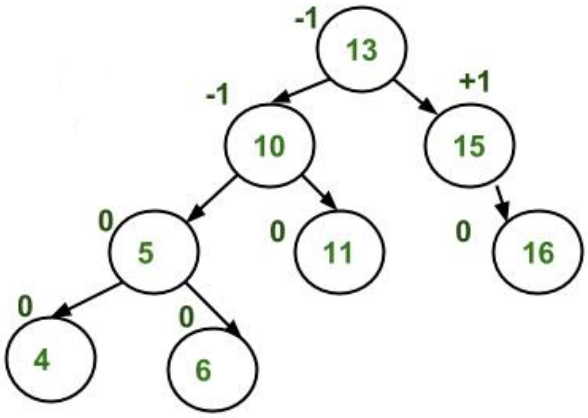
- Insert "3".



- Insert "15".




- Insert "7".




## Problem 4 (2 points)

Write down the pseudocode for INSERT(y, k) operation, where y is a root of a subtree, and k is the new element.

*Hint*: Note that after inserting the element you must check (going up from the inserted element to the root) if/where the tree has been unbalanced. Thus, one can user recursion and analyze only one level in each call.

## Problem 5 (3 points)

Solve the challenge `https://www.hackerrank.com/challenges/self-balancing-tree/problem` on Hackerrank.

## Problem 6 (1 point)

Describe the algorithm of deleting a node from a AVL tree.

## References

- Guide to AVL trees in Java: `https://www.baeldung.com/java-avl-trees`
- AVL Tree: `https://www.geeksforgeeks.org/avl-tree-set-1-insertion/`