AVL Tree

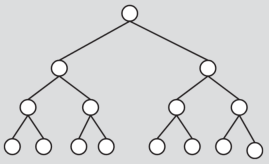
When we use a binary search tree to look up data, the speed of our search can greatly depend on the shape of the tree. This shape, in turn, is determined by the data we put into it. For example, if our data is already sorted, our tree ends up being a straight line, which isn't efficient for searching. However, if our tree is well-structured, our search can be much quicker. In fact, the shorter the tree (or the 'height' of the tree), the faster our search will be. So, we ideally want our tree to be as short as possible.

To achieve this Some might think: "why not make the tree perfectly balanced?". This type of trees is called **A Perfectly Balanced Binary Tree**

A perfectly balanced binary tree

It is a binary tree such that

1. The heights of the left and right subtrees of the root are equal.
2. The left and right subtrees of the root are perfectly balanced binary trees.



> ⚠ **The problem with Perfectly balanced Trees**
> A perfectly balanced binary tree is a special kind of tree where every node, let's call it 'x', has its left and right subtrees of the same height. This means if you look at any node in the tree, the number of levels below it on the left side is the same as on the right side.
>
> Now, if a perfectly balanced binary tree has a height of 'h' (height is the number of levels in the tree), then the total number of nodes in the tree is $2^h - 1$. This is a mathematical property that comes from the way the tree is structured.
>
> But here's the catch: if you have a set of data and the number of items in it is not equal to $2^h - 1$ (where 'h' is a nonnegative integer), then you can't arrange that data into a perfectly balanced binary tree. This is because of the strict structure of perfectly balanced binary trees.

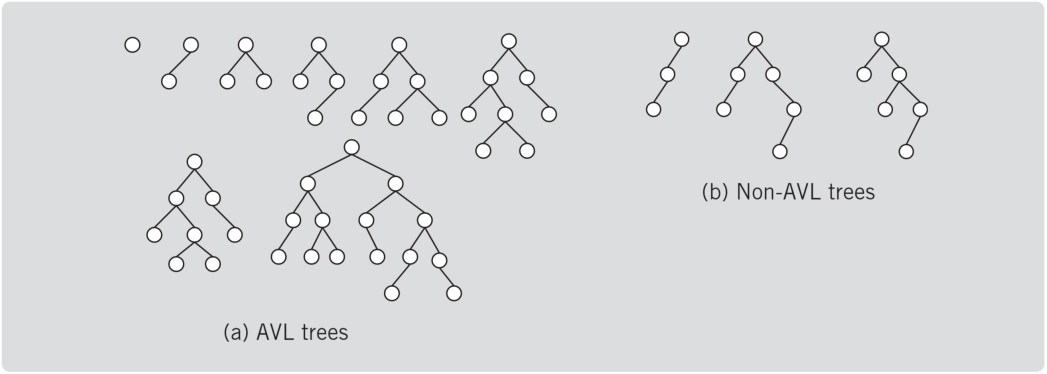
> In other words, perfectly balanced binary trees are very specific and strict in their structure, which can make them difficult to work with in some cases. That's why they're not always practical or possible to use.

A less strict Data structure would be an **AVL Tree**

AVL Tree (Height-Balanced Tree)

It is a binary search tree such that:

1. The heights of the left and right subtrees of the root differ by at most 1.
2. The left and right subtrees of the root are AVL trees.



Because an AVL tree is a binary search tree, the search algorithm for an AVL tree is the same as the search algorithm for a binary search tree. Other operations, such as finding the height, determining the number of nodes, checking whether the tree is empty, tree traversal, and so on, on AVL trees can be implemented exactly the same way they are implemented on binary trees. However, item insertion and deletion operations on AVL trees are somewhat different from the ones discussed for binary search trees. This is because after inserting (or deleting) a node from an AVL tree, the resulting binary tree must be an AVL tree.

Balance Factor

$$\begin{aligned}&\text{let x be a node in a binary tree.}\\&\text{let } x_l \text{ be the height of the left subtree of } x\\&\text{let } x_r\text{be the height of the right subtree of } x\end{aligned}$$

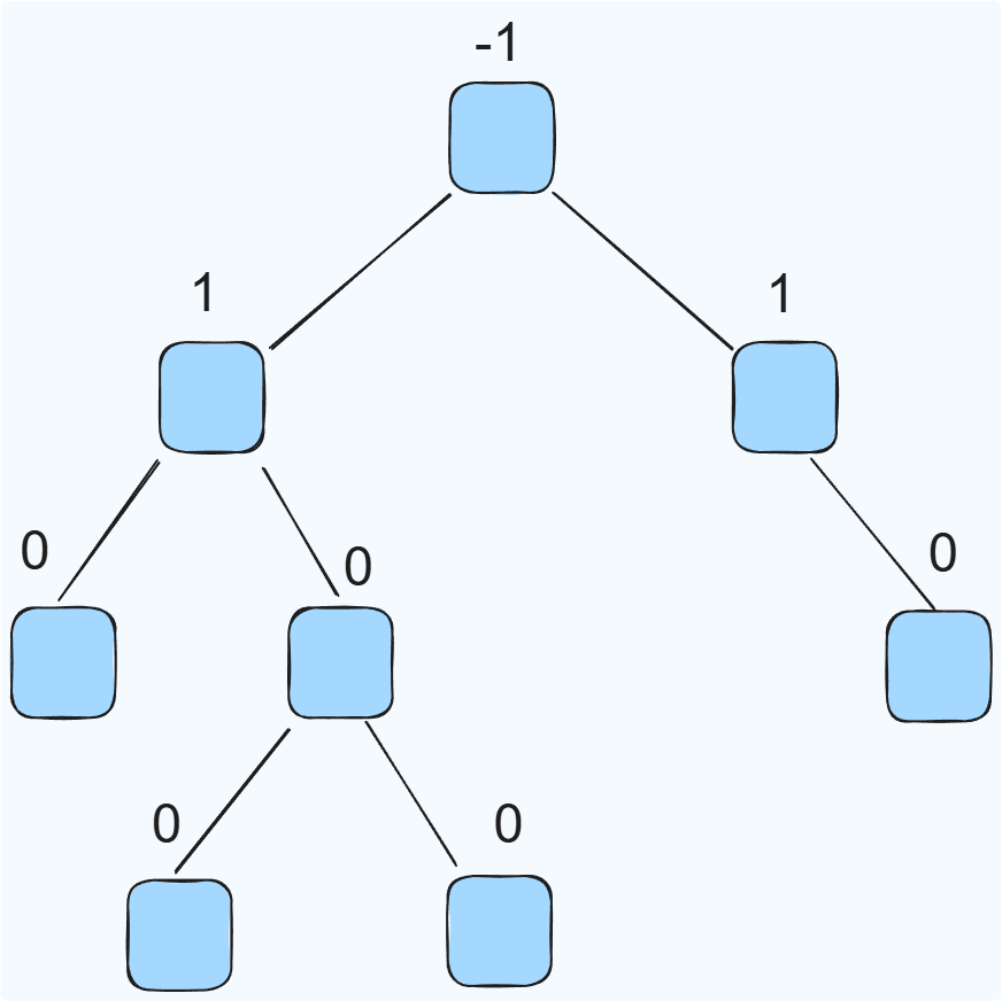
The balance factor equation

$$bf(x) = x_r - x_l$$

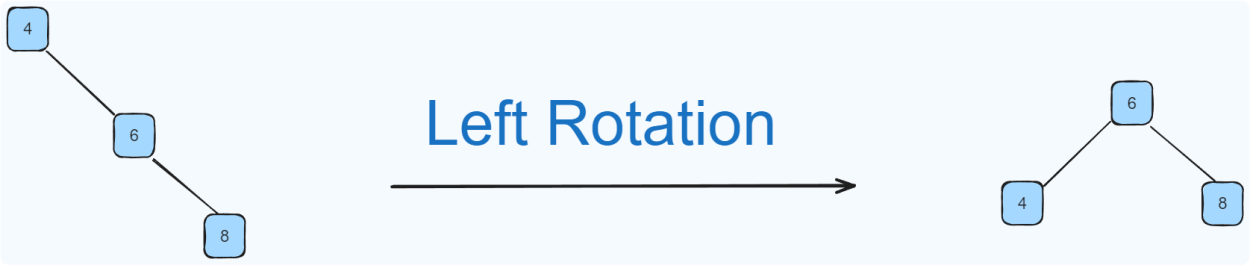
1. If x is left high, bf(x) = -1

2. If x is equal high, bf(x) = 0
3. If x is right high, bf(x) = 1

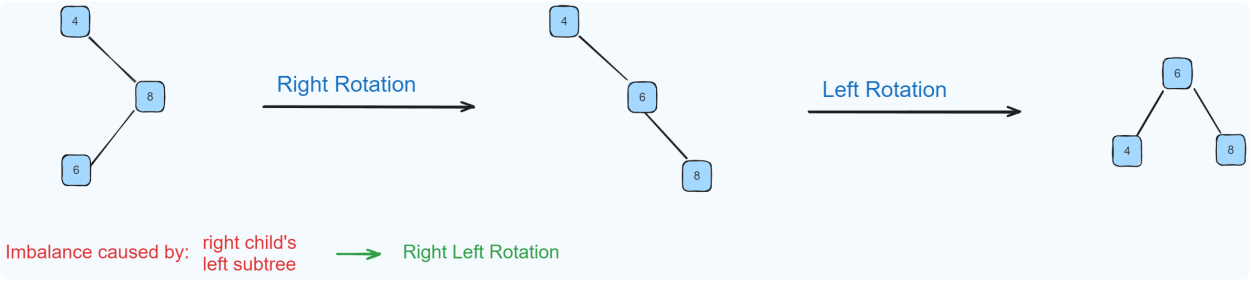
> **Note**
>
> The above values "left high, equal high, and, right high" are a result of our equation with subtracts the height of the right subtree from the height of the left subtree of x. **Others might do the opposite and subtract $x_l - x_r$ instead of $x_r - x_l$ which is totally valid. In this case the values of "left high, equal high, and, right high" would be:**
>
> 1. If x is left high, bf(x) = 1
> 2. If x is equal high, bf(x) = 0
> 3. If x is right high, bf(x) = -1



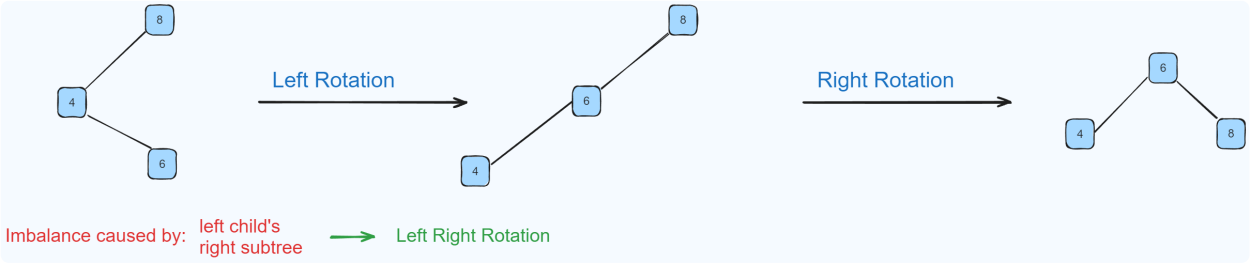
Rotations



Right Left Rotation

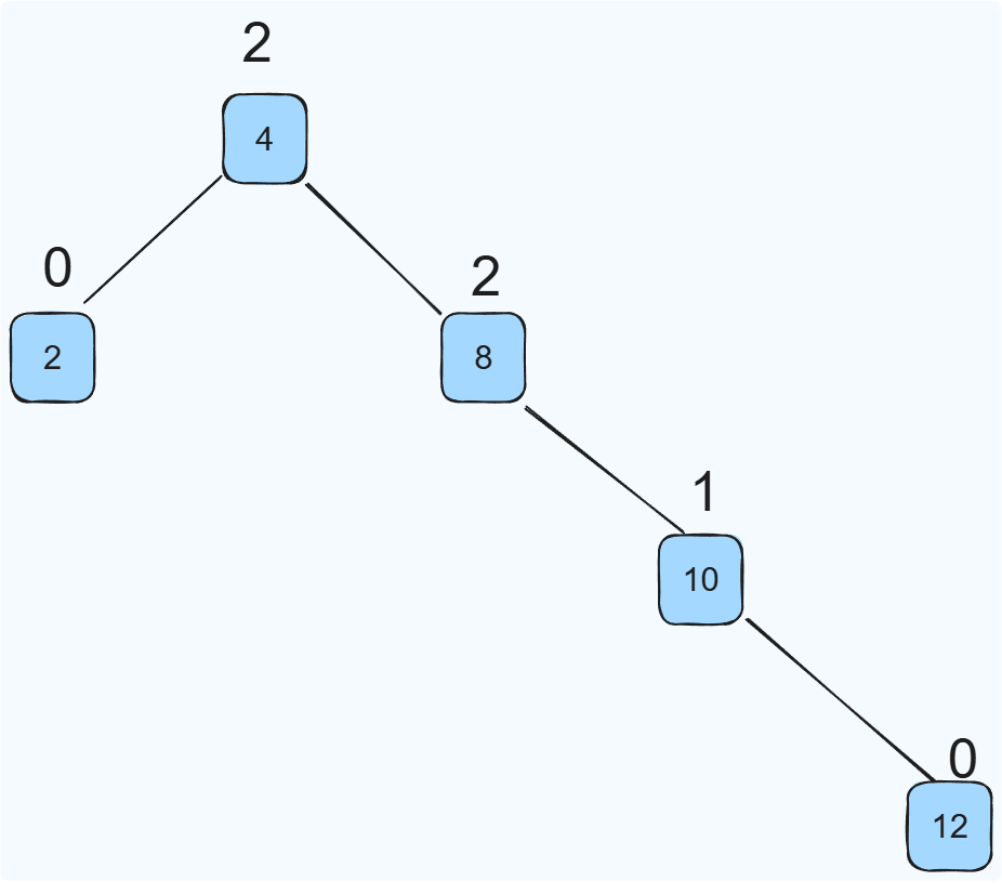


Left Right Rotation

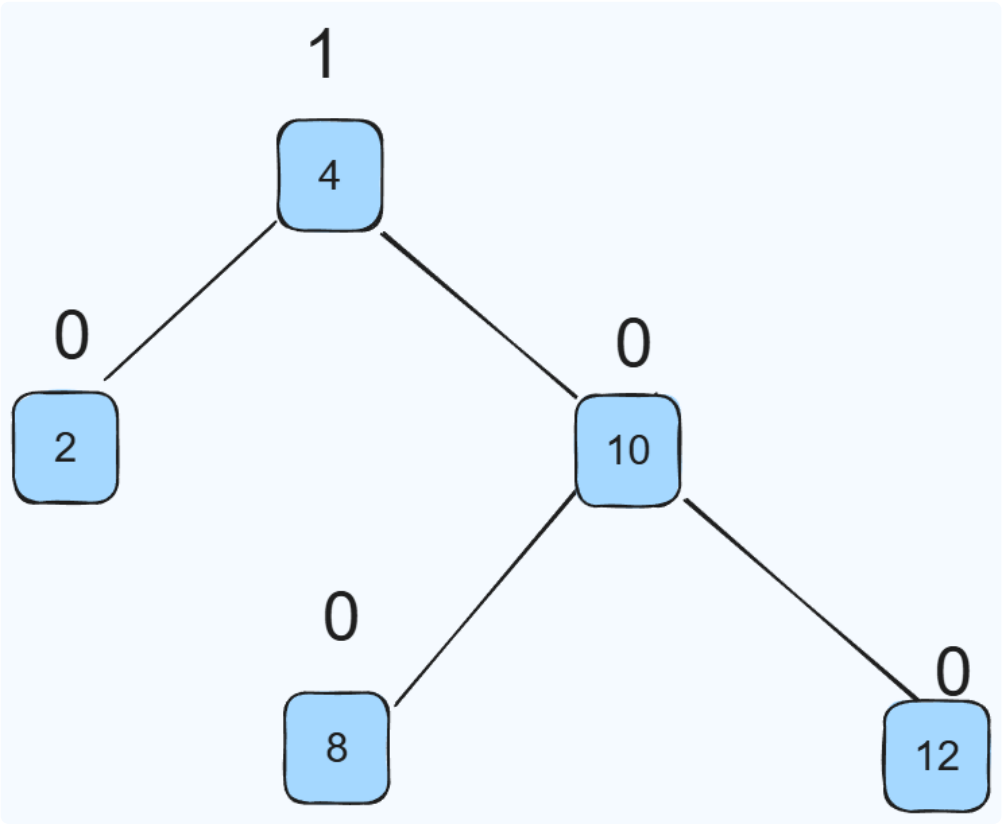


Balancing AVL Trees

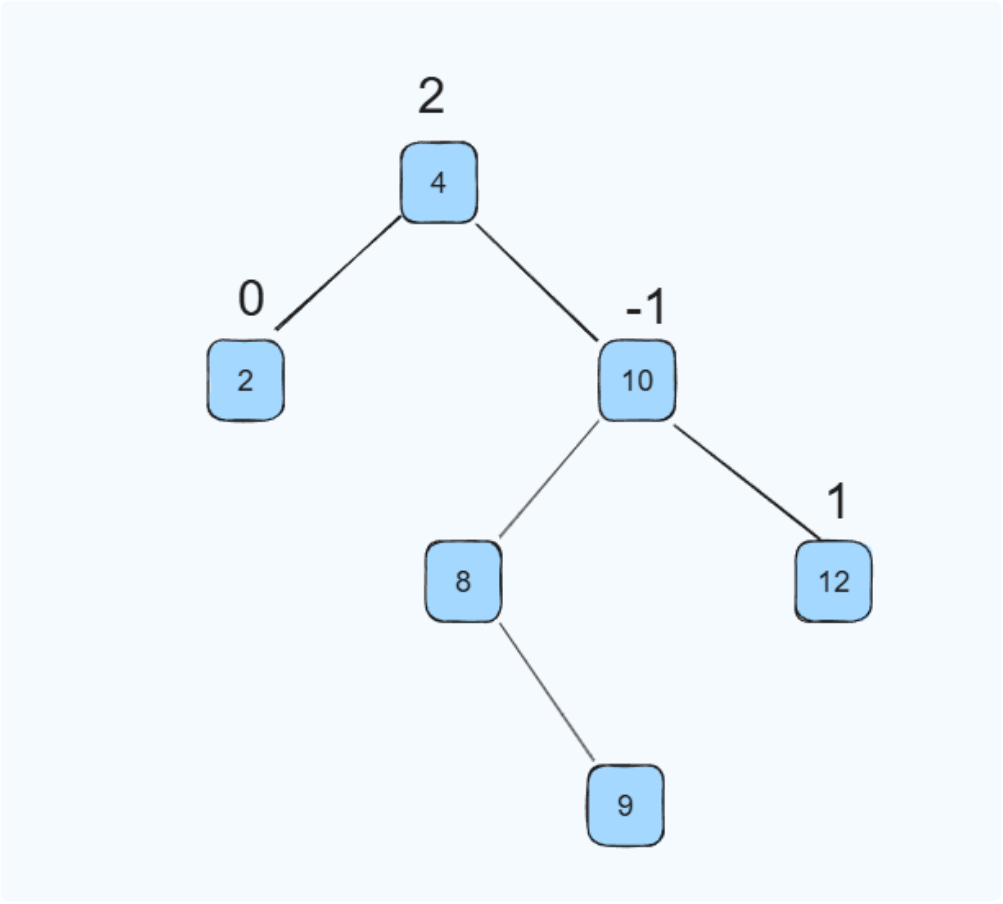
example 1 unbalanced



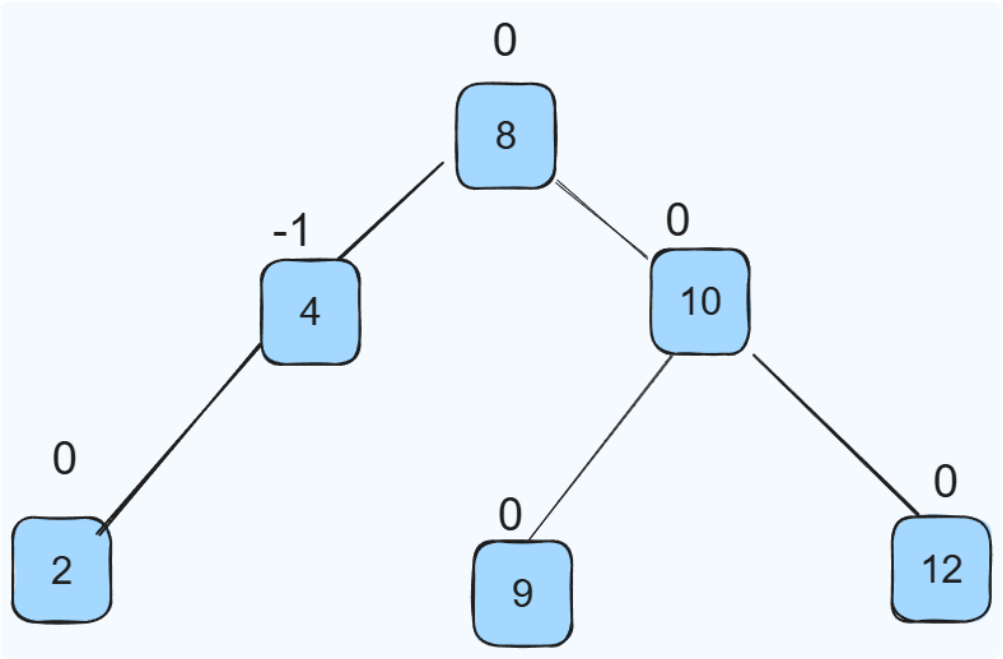
Example 1 Solution



Example 2 unbalanced



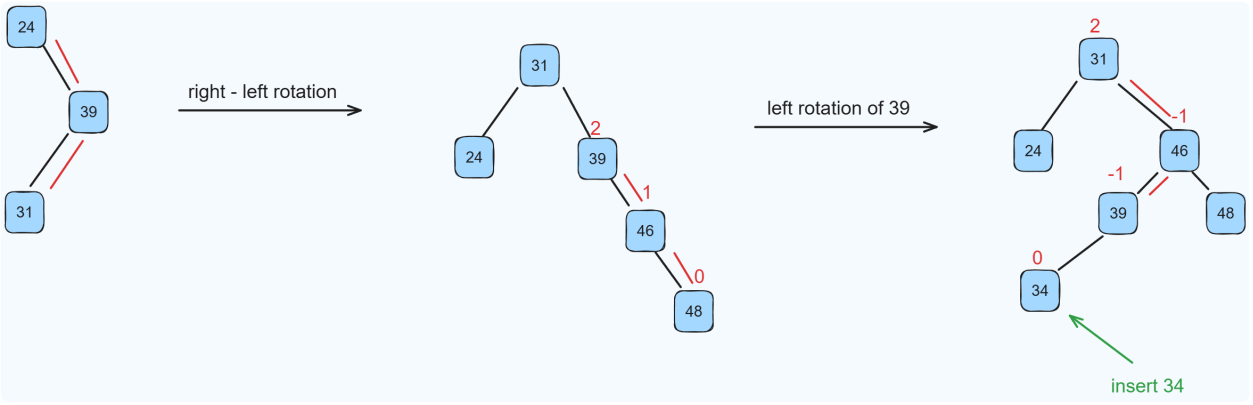
Example 2 Solution:

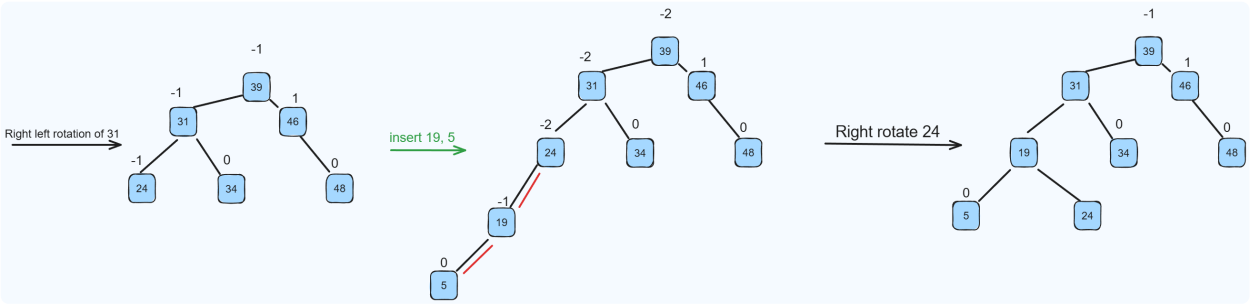


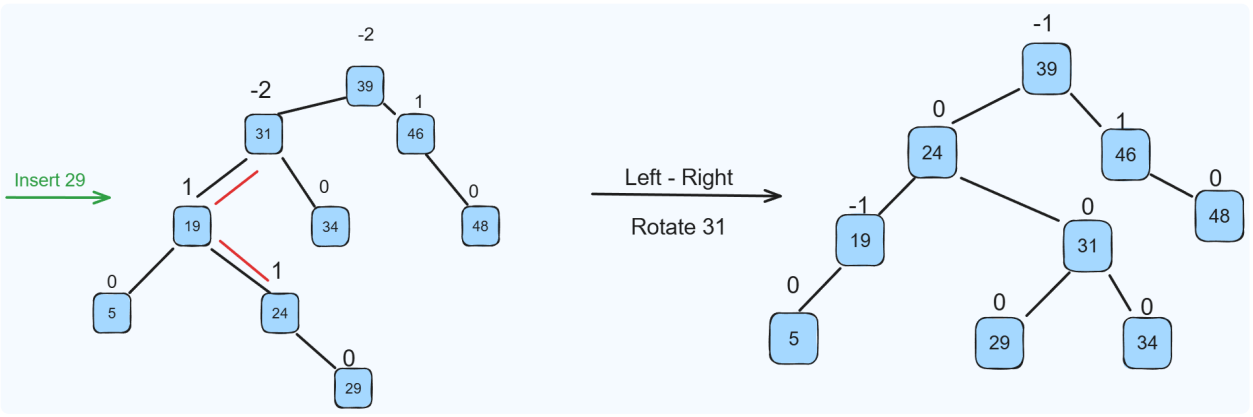
Insertion Example

Insert the following values in order

24, 39, 31, 46, 48, 34, 19, 5, 29







🔗 Resources

- Trees (programiz.com)
- AVL Trees (programiz.com)
- Data Structures by Rob Edwards
  - Intro to rotations
  - Rotations
  - AVL 1 introduction
  - AVL Tree 7 complete example of adding data to an AVL tree
- 10.1 AVL Tree - Insertion and Rotations