What is Tree Data Structure?

Tree data structure is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. **It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.**

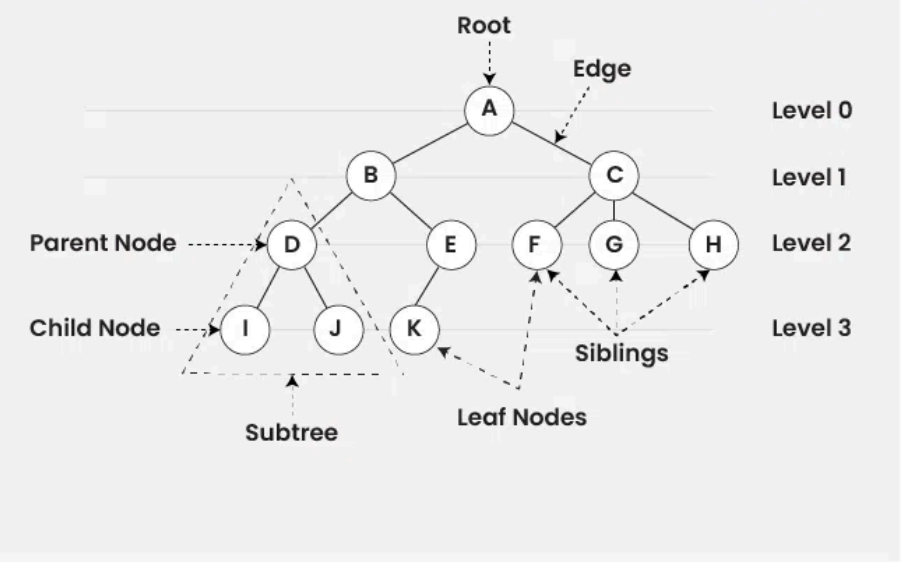
The topmost node of the tree is called the **root**, and the nodes below it are called the child nodes. Each node can have multiple child nodes, and these child nodes can also have their own child nodes, forming a recursive structure.

Why is Tree considered a non-linear data structure?

The data in a tree are not stored in a sequential manner i.e., they are not stored linearly. Instead, they are arranged on multiple levels or we can say it is a hierarchical structure. For this reason, the tree is considered to be a non-linear data structure.

Basic Terminologies In Tree Data Structure:

- **Parent Node:** The node which is a predecessor of a node is called the parent node of that node. **{B}** is the parent node of **{D, E}**.
- **Child Node:** The node which is the immediate successor of a node is called the child node of that node. Examples: **{D, E}** are the child nodes of **{B}.**
- **Root Node:** The topmost node of a tree or the node which does not have any parent node is called the root node. {A**}** is the root node of the tree. A non-empty tree must contain exactly one root node and exactly one path from the root to all other nodes of the tree.
- **Leaf Node or External Node:** The nodes which do not have any child nodes are called leaf nodes. **{K, L, M, N, O, P, G}** are the leaf nodes of the tree.
- **Ancestor of a Node:** Any predecessor nodes on the path of the root to that node are called Ancestors of that node. **{A,B}** are the ancestor nodes of the node **{E}**
- **Descendant:** A node x is a descendant of another node y if and only if y is an ancestor of x.
- **Sibling:** Children of the same parent node are called siblings. **{D,E}** are called siblings.
- **Level of a node:** The count of edges on the path from the root node to that node. The root node has level **0**.
- **Internal node:** A node with at least one child is called Internal Node.
- **Neighbor of a Node:** Parent or child nodes of that node are called neighbors of that node.
- **Subtree**: Any node of the tree along with its descendant.



Properties of Trees

- Trees do not have any cycles or loops.
- There is exactly one path between any two nodes in a tree.
- The number of edges in a tree with "n" nodes is always "n-1".

Importance for Tree Data Structure:

1. One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a computer:

2. Trees (with some ordering e.g., BST) provide moderate access/search (quicker than Linked List and slower than arrays).
3. Trees provide moderate insertion/deletion (quicker than Arrays and slower than Unordered Linked Lists).
4. Like Linked Lists and unlike Arrays, Trees don’t have an upper limit on the number of nodes as nodes are linked using pointers.

Types of Tree data structures:

Tree data structure can be classified into three types based upon the number of children each node of the tree can have. The types are:

- **Binary tree:** In a binary tree, each node can have a maximum of two children linked to it. Some common types of binary trees include full binary trees, complete binary trees, balanced binary trees, and degenerate or pathological binary trees.
- **Ternary Tree:** A Ternary Tree is a tree data structure in which each node has at most three child nodes, usually distinguished as “left”, “mid” and “right”.

- **N-ary Tree or Generic Tree:** Generic trees are a collection of nodes where each node is a data structure that consists of records and a list of references to its children(duplicate references are not allowed). Unlike the linked list, each node stores the address of multiple nodes.

Binary Tree Data Structure

A **Binary Tree Data Structure** is a hierarchical data structure in which each node has at most two children, referred to as the left child and the right child. It is commonly used in computer science for efficient storage and retrieval of data, with various operations such as insertion, deletion, and traversal.

Binary Tree (Array implementation)

Given an array that represents a tree in such a way that array indexes are values in tree nodes and array values give the parent node of that particular index (or node). The value of the root node index would always be -1 as there is no parent for root. Construct the standard linked representation of a given Binary Tree from this given representation.

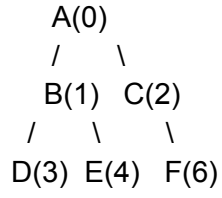
Ways to represent:

Trees can be represented in two ways as listed below:

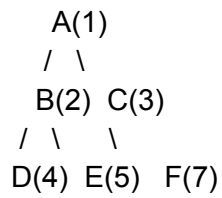
1. Dynamic Node Representation (Linked Representation).
2. Array Representation (Sequential Representation).

Now, we are going to talk about the sequential representation of the trees. In order to represent a tree using an array, the numbering of nodes can start either from 0–(n-1) or 1– n, consider the below illustration as follows:

Illustration:



OR,



Procedure:

Note: parent, left_child and right_child are the values of indices of the array.

Case 1: (0—n-1)

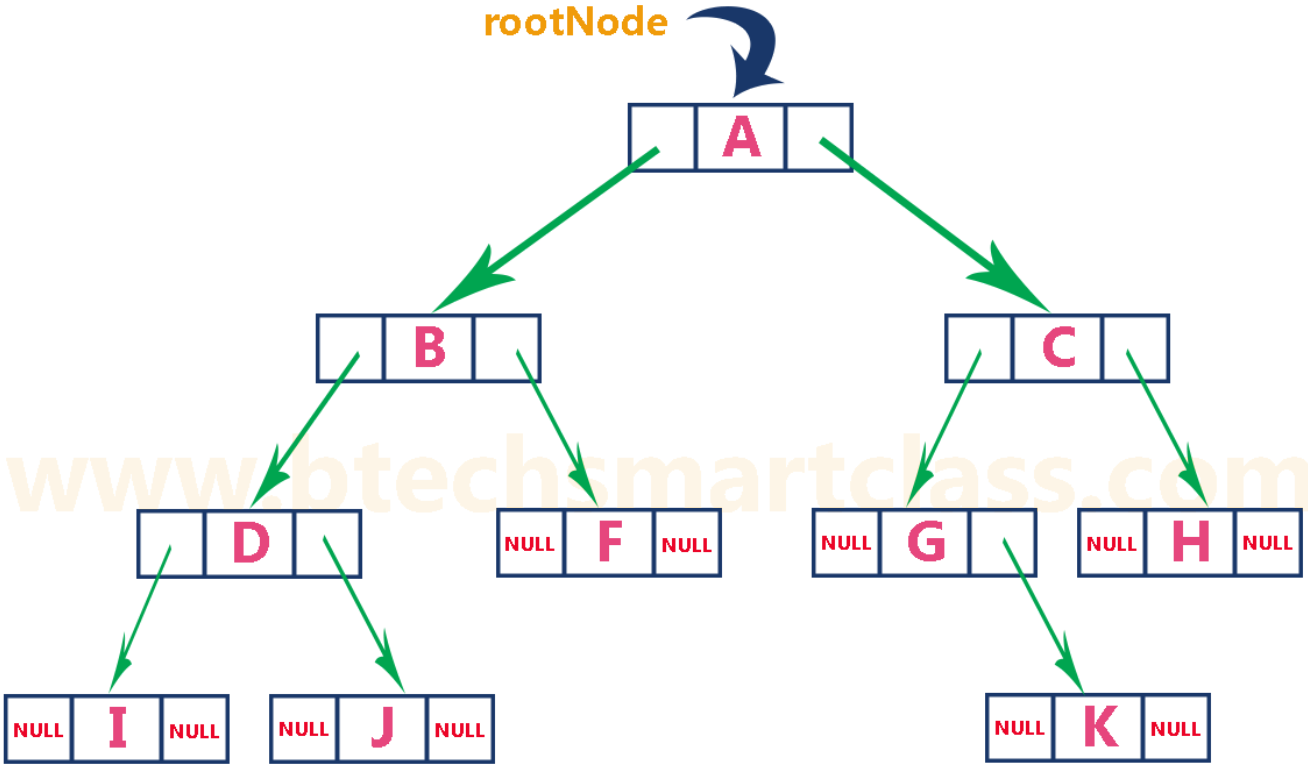
if (say)parent=p;
then left_child=(2*p)+1;
and right_child=(2*p)+2;

Case 2: 1—n

if (say)parent=p;
then left_child=(2*p);
and right_child=(2*p)+1;

Linked List Representation of Binary Tree

We use a double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for storing right child address.



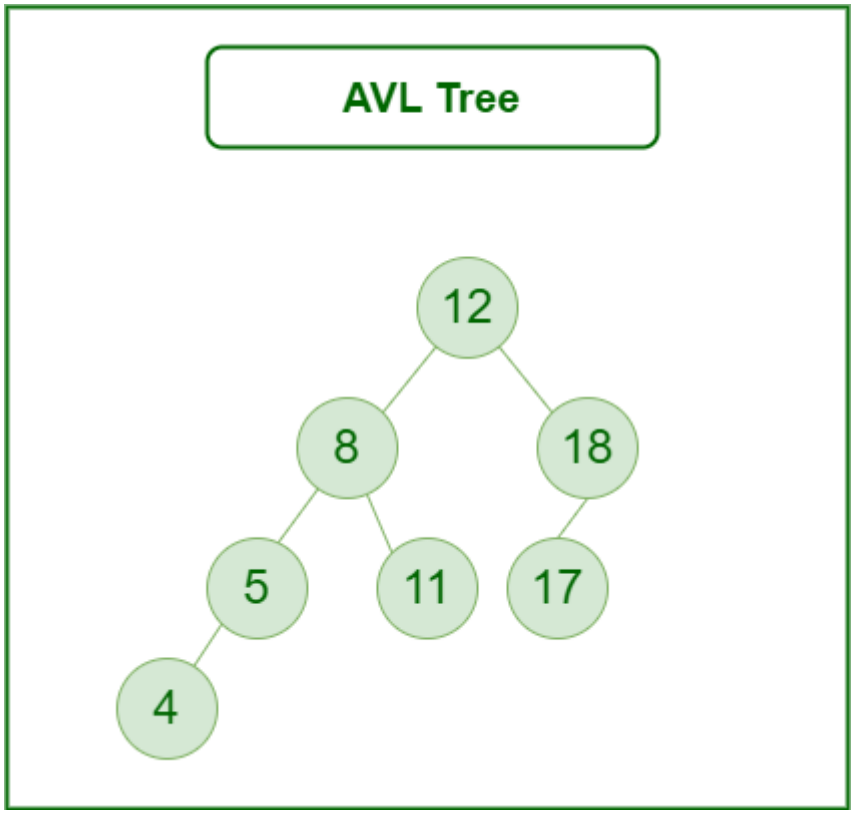
AVL Tree Data Structure

> An **AVL tree** defined as a self-balancing **Binary Search Tree** (BST) where the difference between heights of left and right subtrees for any node cannot be more than one.

The difference between the heights of the left subtree and the right subtree for any node is known as the **balance factor** of the node.

The AVL tree is named after its inventors, Georgy Adelson-Velsky and Evgenii Landis, who published it in their 1962 paper “An algorithm for the organization of information”.

Example of AVL Trees:



AVL tree

The above tree is AVL because the differences between the heights of left and right subtrees for every node are less than or equal to 1.

Operations on an AVL Tree:

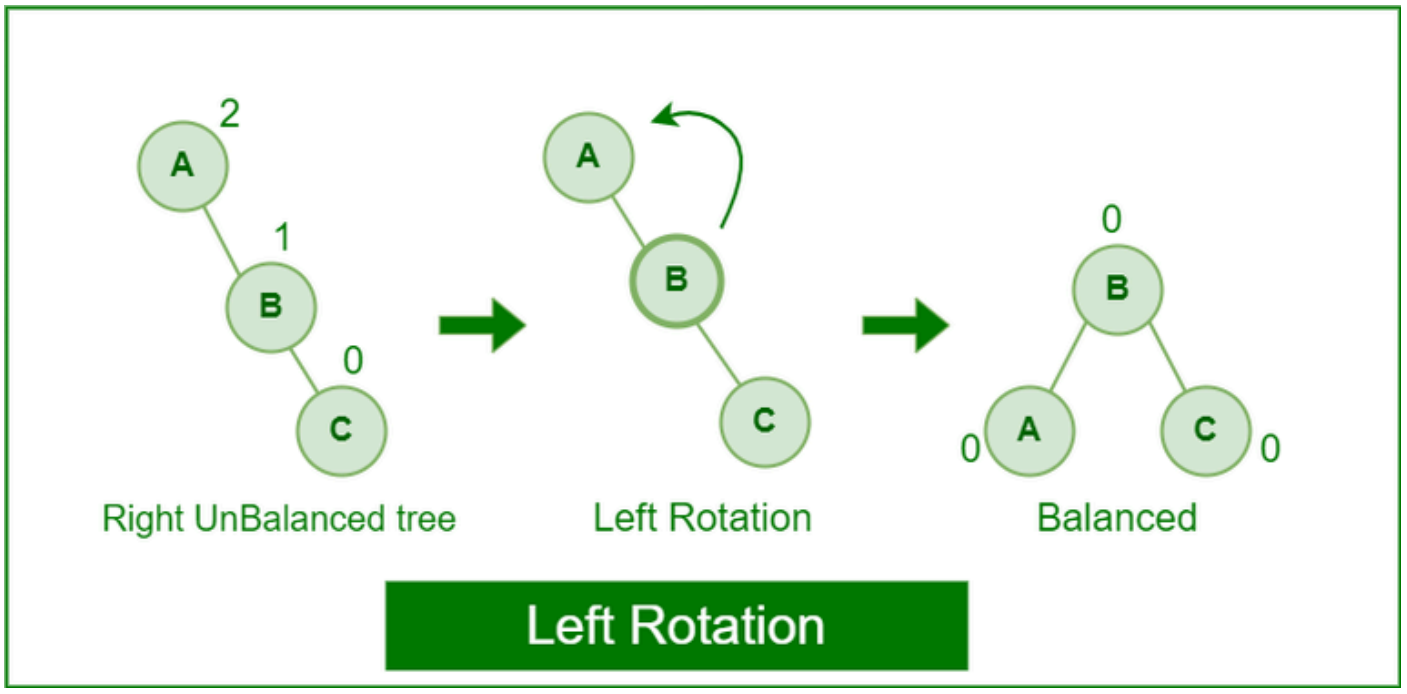
- Insertion
- Deletion
- Searching [It is similar to performing a search in BST]

Rotating the su btrees in an AVL Tree:

An AVL tree may rotate in one of the following four ways to keep itself balanced:

Left Rotation:

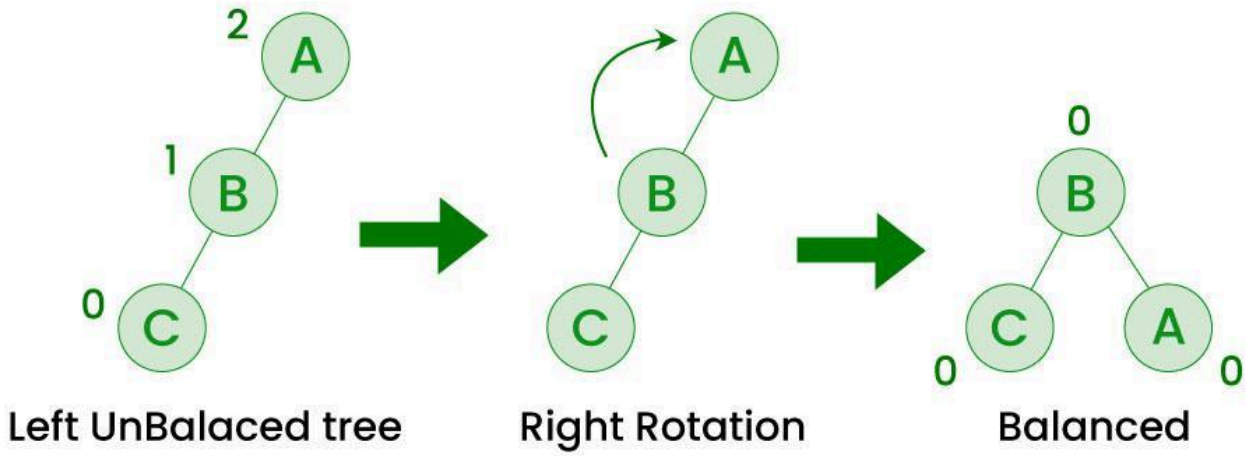
When a node is added into the right subtree of the right subtree, if the tree gets out of balance, we do a single left rotation.



Left-Rotation in AVL tree

Right Rotation:

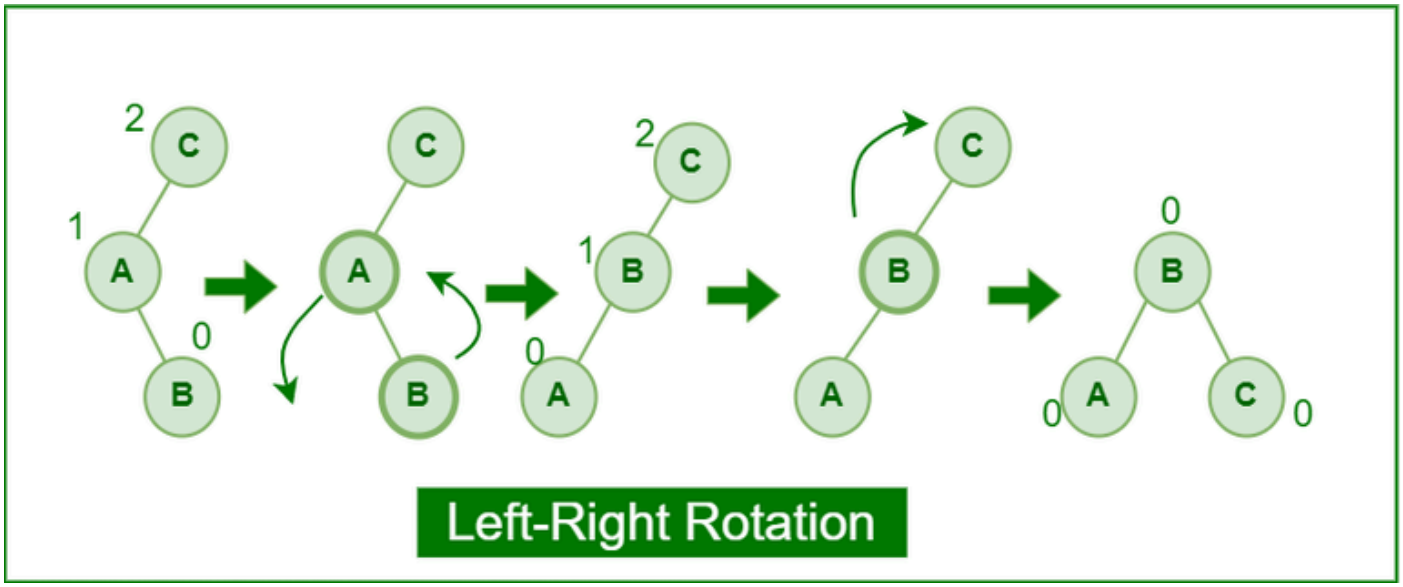
If a node is added to the left subtree of the left subtree, the AVL tree may get out of balance, we do a single right rotation.



Right-Rotation in AVL Tree

Left-Right Rotation:

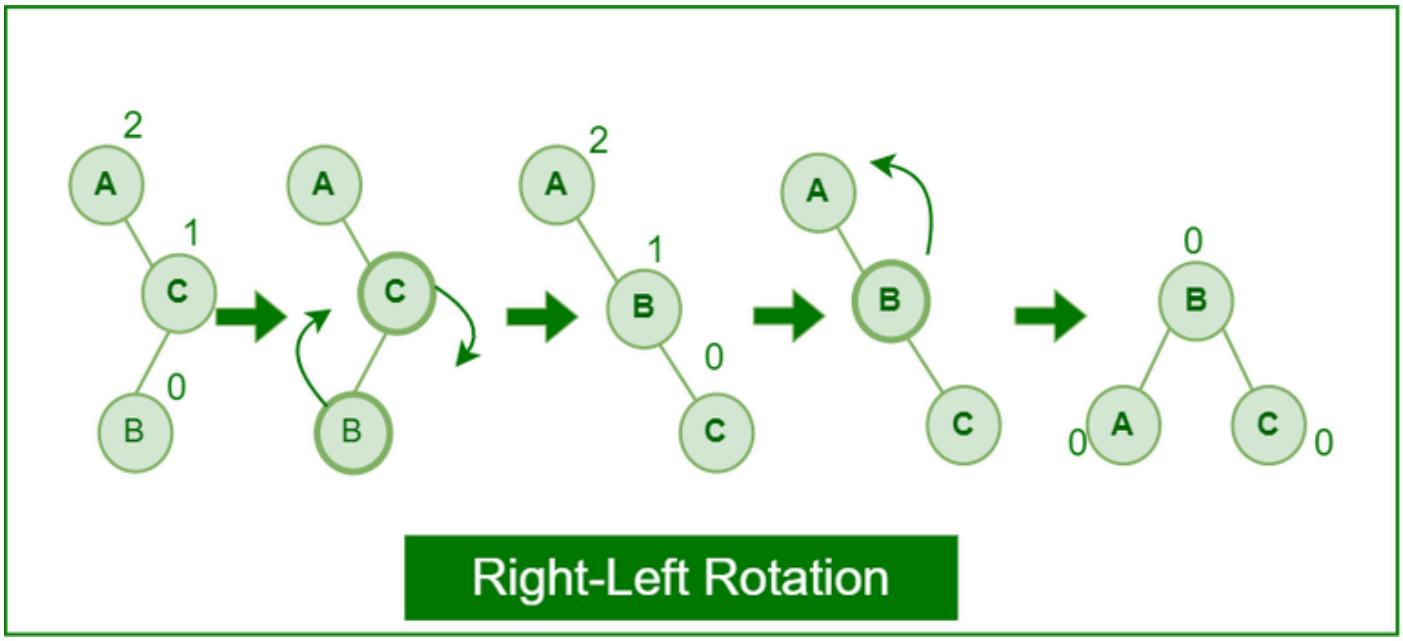
A left-right rotation is a combination in which first left rotation takes place after that right rotation executes.



Left-Right Rotation in AVL tree

Right-Left Rotation:

A right-left rotation is a combination in which first right rotation takes place after that left rotation executes.



Right-Left Rotation in AVL tree

Applications of AVL Tree:

1. It is used to index huge records in a database and also to efficiently search in that.
2. For all types of in-memory collections, including sets and dictionaries, AVL Trees are used.
3. Database applications, where insertions and deletions are less common but frequent data lookups are necessary
4. Software that needs optimized search.
5. It is applied in corporate areas and storyline games.

Advantages of AVL Tree:

1. AVL trees can self-balance themselves.

2. It is surely not skewed.
3. It provides faster lookups than Red-Black Trees
4. Better searching time complexity compared to other trees like binary tree.
5. Height cannot exceed log(N), where, N is the total number of nodes in the tree.

Disadvantages of AVL Tree:

1. It is difficult to implement.
2. It has high constant factors for some of the operations.
3. Less used compared to Red-Black trees.
4. Due to its rather strict balance, AVL trees provide complicated insertion and removal operations as more rotations are performed.
5. Take more processing for balancing.