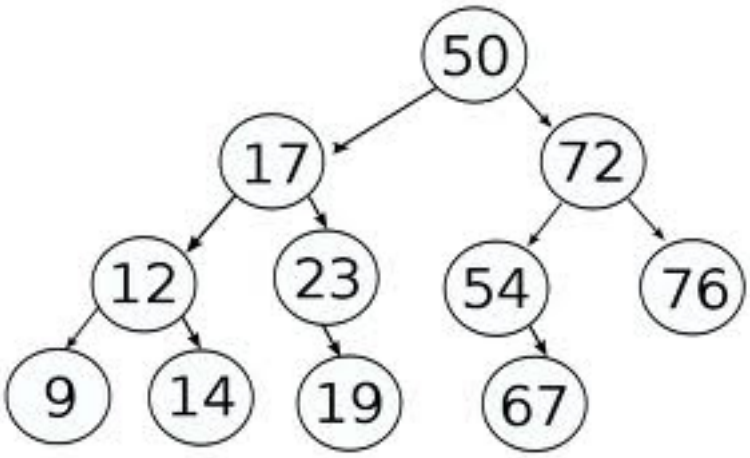
TREE:

Tree is a non linear (or two dimensional) data structure that contains data items or elements arranged in hierarchical format. Trees are used to store data that needs to be represented in a hierarchical format. A tree is a finite non empty set of elements. One of these elements is called the root, and the remaining elements, if any, are partitioned into trees, which are called the sub trees of the tree T. A tree structure looks like as– Child nodes of Root node



TREE TERMINOLOGY

- **Root:** - Root node is the first node of the tree from which all other nodes branch out
- **Parent Node:** - It is the node directly above of any other node in hierarchy
- **Child Node:** - It is the nodes directly blow of any other node in hierarchy
- **Sibling Nodes:** - Sibling nodes are the child nodes of the same parents
- **Leaf Node:** - Leaf node is a node which does not have any child node
- **Level of an Element:** - If the root of the tree T is at level 1 then its children, if any, are at level
- **Degree of an Element:** - The degree of an element is the number of children it has. The degree of a leaf is zero

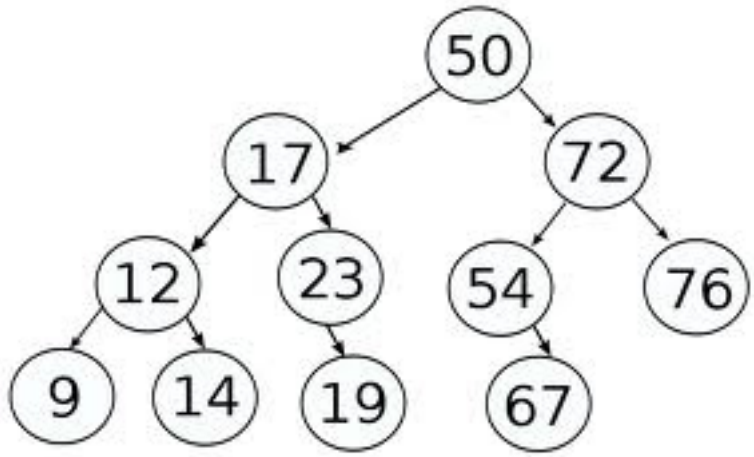
- **Degree of a Tree:** - The degree of a tree is the maximum of its elements degrees
- **Height/ Depth of a Tree:** - If level of the root is denoted by 1, then the maximum level number of the tree is known as its height or depth

BINARY TREE

A binary tree T is either empty or has a finite collections of elements. When the binary tree is not empty, one of its elements is called the root and the remaining elements, if any, are partitioned into two sub trees, which are called the left sub trees and right sub trees of tree T. The essential differences between a binary tree and tree are as–

1. A binary tree can be empty whereas a tree can not
2. Each element in binary tree has at most two sub trees (one or both of these sub trees may be empty). Each element in a tree can have any number of sub trees.

The sub trees of each element in a binary tree are ordered. That is, we distinguish between the left and right sub trees. The sub trees in a tree are unordered



Properties of Binary Tree

1. A binary tree T with n elements, n>0, has exactly n-1 edges.
2. A binary tree T of height h, h>0, has at least h and at most 2^h-1 elements
3. The height of the binary tree T that contains n elements, n>=0, at most n and at least $\lceil \log_a (n+1) \rceil$ elements.
4. If i, 1<=i<=n, be the number assigned to an elements of a complete binary tree. The following statements are true
    - If i = 1, then this element is the root of the binary tree.
    - If 2i > n, then this element has no left child otherwise, its left child has been assigned the number 2i.
    - If 2i + 1 > n, then this element has no right child, otherwise its right child has been assigned the number 2i+1

Types of Binary Tree

Many types of Binary Tree are there some them are given as-

1. Rooted Binary Tree
2. A Full Binary Tree
3. Perfect Binary Tree
4. Complete Binary Tree
5. Extended Binary Tree
6. Strictly Binary Tree

Operations:

Create Root

create a Node class and add assign a value to the node. This becomes tree with only a root node

```
class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data

    def PrintTree(self):
        print(self.data)
root = Node(10)
root.PrintTree()
```

Inserting into a Tree:

Steps:

create a Node and add a insert class to it. The insert class compares the value of the node to the parent node and decides to add it as a left node or a right node. Finally the printTree class is used to print the tree.

```
class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data
    def insert(self, data):
# Compare the new value with the parent node
        if self.data:
            if data < self.data:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.data:
                if self.right is None:
                    self.right = Node(data)
                else:
                    self.right.insert(data)
        else:
            self.data = data
# Print the tree
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print( self.data),
        if self.right:
            self.right.PrintTree()
# Use the insert method to add nodes
root = Node(12)
root.insert(6)
root.insert(14)
root.PrintTree()
```

Traversing a Tree:

The tree can be traversed by deciding on a sequence to visit each node. As we can clearly see we can start at a node then visit the left sub-tree first and right sub-tree next. Or we can also visit the right sub-tree first and left sub-tree next. Accordingly there are different names for these tree traversal methods.

Search Tree:

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties – The left sub-tree of a node has a key less than or equal to its parent node's key. The right sub-tree of a node has a key greater than to its parent node's key. Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as –

Search for a value in a B-tree:

Searching for a value in a tree involves comparing the incoming value with the value exiting nodes. Here also we traverse the nodes from left to right and then finally with the parent. If the searched for value does not match any of the exiting value, then we return not found message else the found message is returned.

```
class Node:
    def __init__(self, data):

        self.left = None
        self.right = None
        self.data = data
# Insert method to create nodes
    def insert(self, data):

        if self.data:
            if data < self.data:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.data:
                if self.right is None:
                    self.right = Node(data)
                else:
                    self.right.insert(data)
        else:
            self.data = data
# findval method to compare the value with nodes
    def findval(self, lkpval):
        if lkpval < self.data:
            if self.left is None:
                return str(lkpval)+" Not Found"
            return self.left.findval(lkpval)
        elif lkpval > self.data:
            if self.right is None:
                return str(lkpval)+" Not Found"
            return self.right.findval(lkpval)
        else:
            print(str(self.data) + ' is found')
# Print the tree
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print( self.data),
        if self.right:
            self.right.PrintTree()
```

```
root = Node(12)
root.insert(6)
root.insert(14)
root.insert(3)
print(root.findval(7))
print(root.findval(14))
```