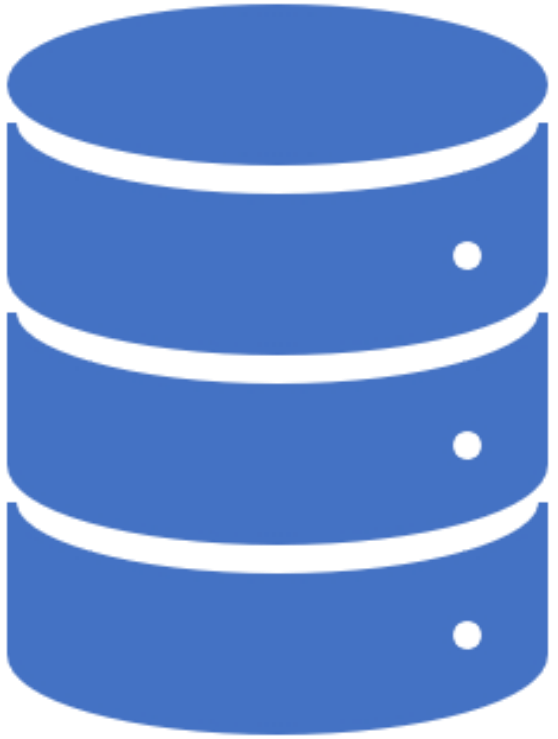
Data Structure with Python Tree



Mr. V.M. Vasava

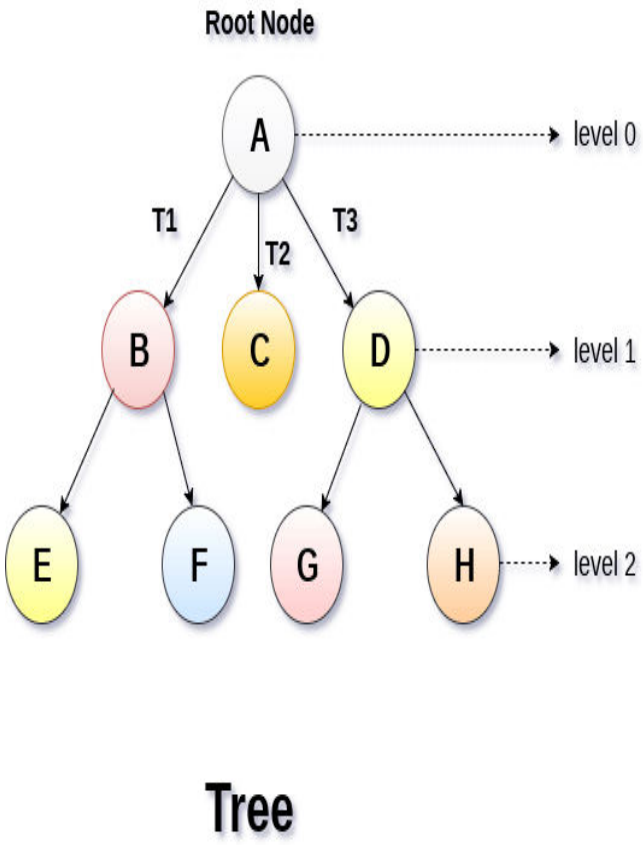
GPG,IT Dept. Surat

Agenda

- Introduction about Tree
- Basic Terminology of Tree
- Types of Tree
- Operations of Tree

Tree

- Def: A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.
- A Tree is a recursive data structure containing the set of one or more data nodes where one node is designated as the root of the tree while the remaining nodes are called as the children of the root.
- The nodes other than the root node are partitioned into the non empty sets where each one of them is to be called sub-tree.
- Nodes of a tree either maintain a parent-child relationship between them or they are sister nodes.



Tree

Basic Terminology of Tree

1. **Root Node** :- The root node is the topmost node in the tree hierarchy.

2. **Node**

   A node is an entity that contains a key or value and pointers to its child nodes.

   The last nodes of each path are called **leaf nodes or external nodes** that do not contain a link/pointer to child nodes.

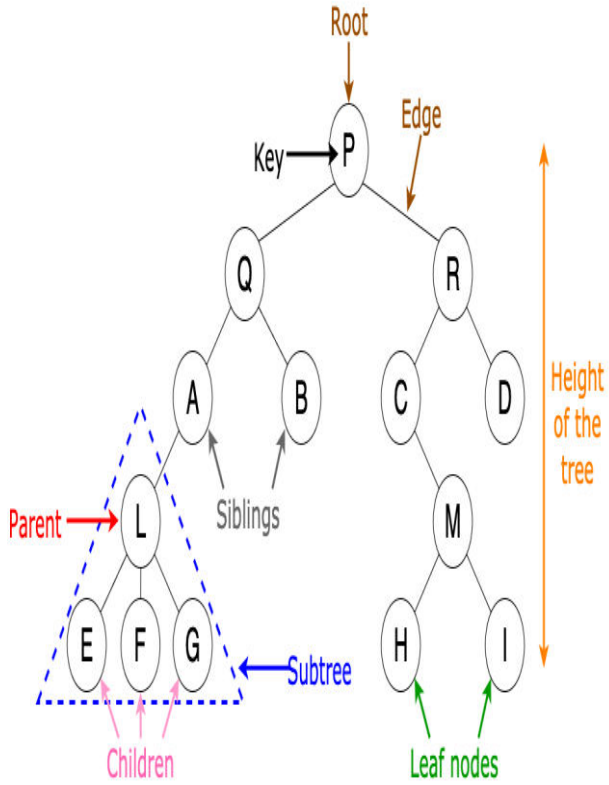
   The node having at least a **child node** is called an **internal node**.

Parent: Node which connects to the child.

Child: Node which is connected to another node is its child.

3. **Edge**

   It is the link between any two nodes.



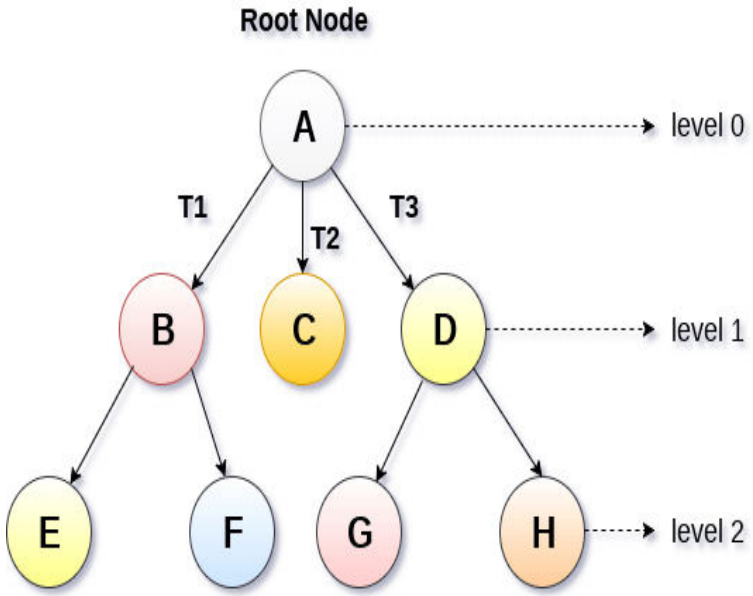
Tree

4.Path :- The sequence of consecutive edges is called path. In the tree shown in the above image, path to the node E is A→ B → E.

5. **Degree** :- Degree of a node is equal to number of children, a node have. In the tree shown in the above image, the degree of node B is 2.

6.**Level Number** :- Each node of the tree is assigned a level number in such a way that each node is present at one level higher than its parent. Root node of the tree is always present at level 0.

7.**Sub Tree** :- If the root node is not null, the tree T1, T2 and T3 is called sub-trees of the root node.



Tree

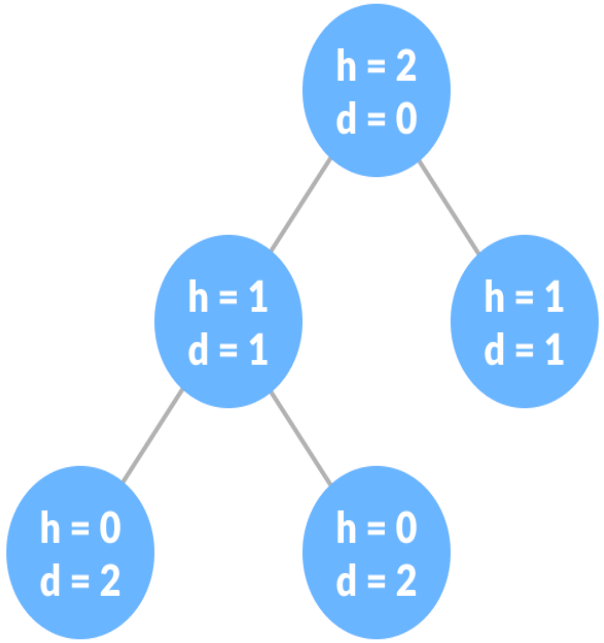
Continued..

9.Depth of a Node

The depth of a node is the number of edges from the root to the node.

10. Degree of Tree

The highest degree of the node among all the nodes in a tree is called the Degree of Tree.



Continued.

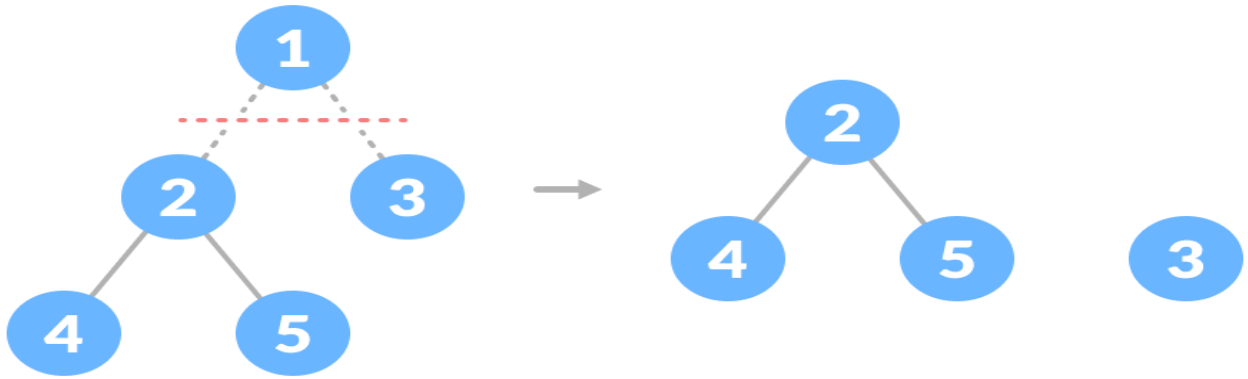
Degree of a Node

The degree of a node is the total number of branches of that node.

Forest

A collection of disjoint trees is called a forest.

Sibling: Node belongs to the same parent.



Indegree and outdegree

- **Indegree**
- The total count of incoming edges to a particular node is known as the indegree of that node.
- Number of branches directed towards the node. Indegree of node B = **1**
- **Outdegree**
- The total number of outgoing edges from a particular node is known as the outdegree of that node.
- The branch is directed away from the node.
- Outdegree of node B = **2**

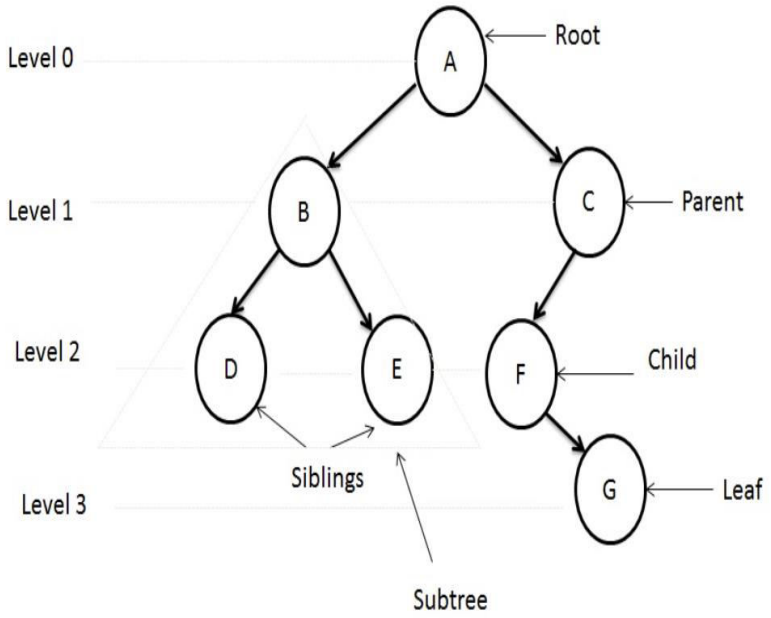


Fig 2. A Tree

Types of Tree

- Binary Tree
- Binary Search Tree
- AVL Tree
- B-Tree

Operations of Tree

1. Creation of Tree

2. Insert the node

3. Delete the node

4. Tree Traversal

Any questions??

Data Structure with Python

Binary Search Tree

Mr. V. M. Vasava

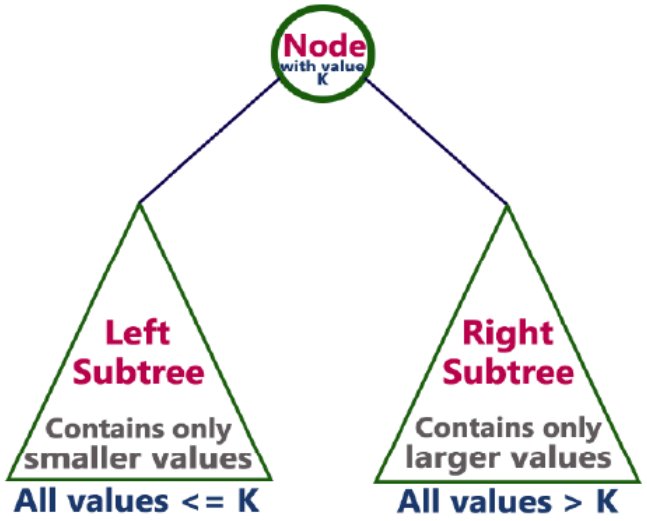
GPG,IT Dept. Surat

Agenda

- Introduction about BT & BST
- Creation of BST
- Traversal – Preorder,Inorder,Postorder
- Application of Binary Tree

BST

- Binary Search Tree is a binary tree in which every node contains only smaller values in its left subtree and only larger values in its right subtree.
- **Binary Search Tree** is a node-based binary tree data structure which has the following properties:
- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.
- BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as −
- left_subtree (keys) < node (key) >= right_subtree (keys)



Operation of BT &BST

- Following are the basic operations of a tree −
- **Search** − Searches an element in a tree.
- **Insert** − Inserts an element in a tree.
- **Pre-order Traversal** − Traverses a tree in a pre-order manner.

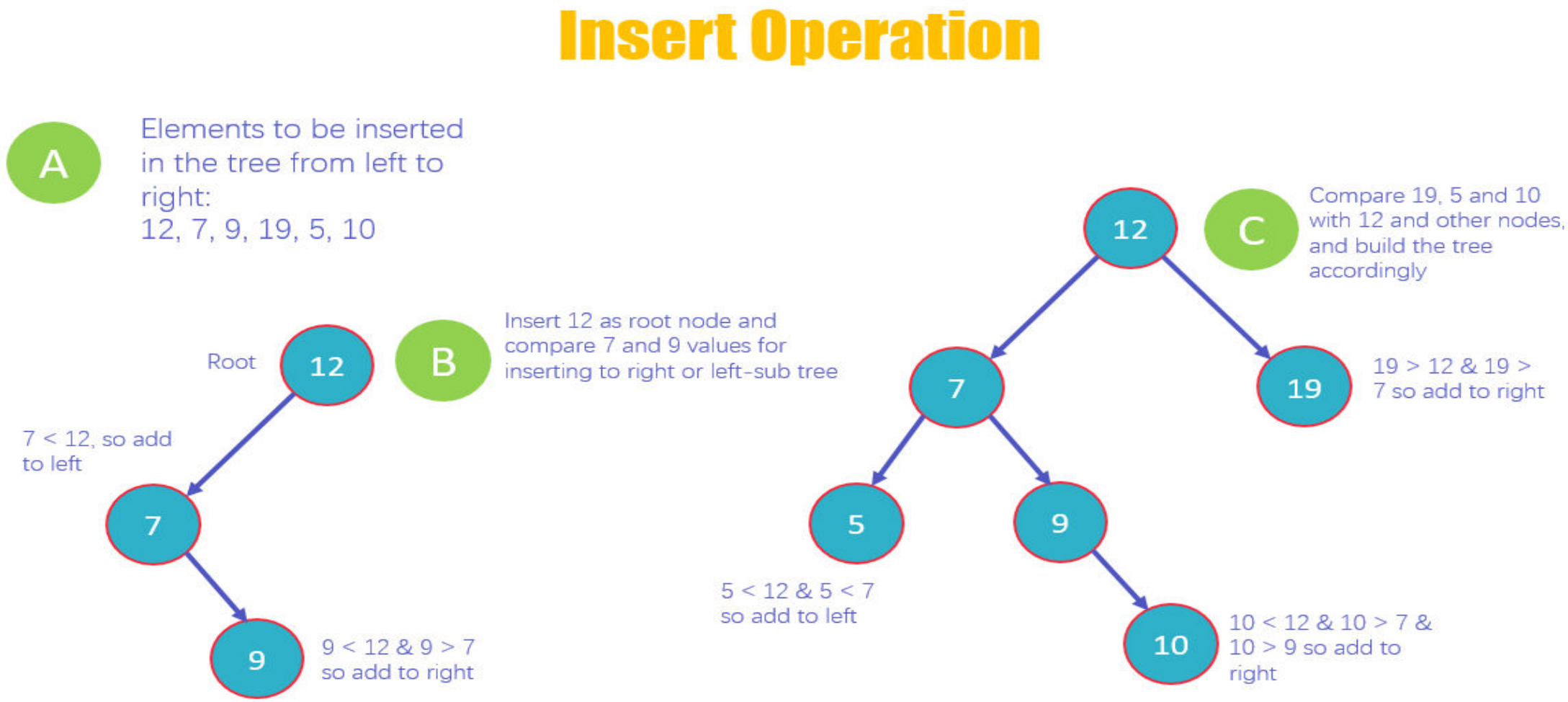
  (Root,Left,Right)
- **In-order Traversal** − Traverses a tree in an in-order manner.

  (Left, Root, Right)
- **Post-order Traversal** − Traverses a tree in a post-order manner.

  (Left, Right, Root)

Example

- Construct BST as give data: 12,7,9,19,5,10



Create BST

```
def add_child(self,data):  # Recursive function to insert a data into a BST.
    if self.data==data:
        return #Node already exists
    if data<self.data:
        #insert left subtree
        if self.left:
            self.left.add_child(data)
        else:
            self.left=BST(data)
    else:
        #insert right subtree
        if self.right:
            self.right.add_child(data)
        else:
            self.right=BST(data)
```

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

```
def build_tree(ele):
    print("Building Tree",ele)
    root=BST(ele[0])
    for i in range(1,len(ele)):
        root.add_child(ele[i])
    return root
no=[4,6,2,15,8]
#no=['vmv','pap','rrr','jjd','zvp']
root=build_tree(no)
```

Tree Traversal: preorder(root)

- In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.
- `Algorithm Preorder(root)`

```
1. Visit the root.
2. Recursively Traverse the left subtree
3. Recursively Traverse the right subtree
```

```
def preorder(self):
        print(self.data)
        if self.left:
                self.left.inorder()
        if self.right:
                self.right.inorder()
```

In-order traversal

- In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.
- If a binary tree is traversed **in-order**, the output will produce sorted key values in an ascending order.

Algorithm: **In-order(root)** →Until all nodes are traversed −

- **Step 1** − Recursively traverse left subtree.
  Step 2 − Visit root node.
  Step 3 − Recursively traverse right subtree.

```
def inorder(self):
        if self.left:
                self.left.inorder()
        print(self.data)
        if self.right:
                self.right.inorder()
```

Post order traversal

- In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.

Algorithm: Post-order(root)

- **Step 1** − Recursively traverse left subtree.
  Step 2 − Recursively traverse right subtree.
  Step 3 − Visit root node.

```
def postorder(self):
        if self.left:
                self.left.inorder()
        if self.right:
                self.right.inorder()
        print(self.data)
```

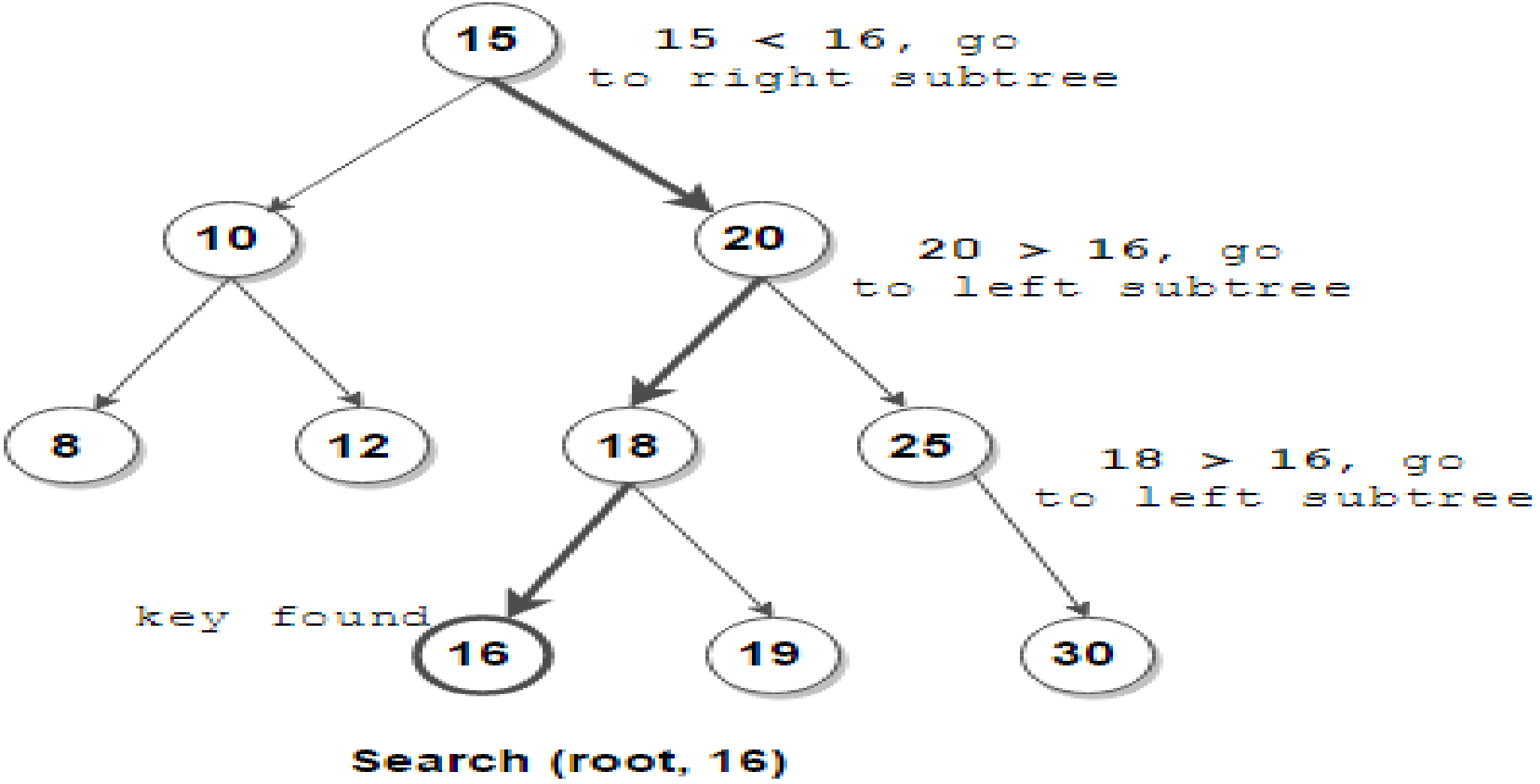
Search :

In a binary search tree, the search operation is performed with **O(log n)** time complexity.
Find a value v in tree.

1. Scan the current node
2. Go left if v is smaller than this node
3. Go right if v is larger than this node
4. Natural generalization of binary search

```
def search(self, val):
        if self.data == val:
            print("Element is Found")
            return
        if val < self.data:  # search in the left subtree
            if self.left:
                self.left.search(val)
            else: print("Element is not found in tree!")
        else: # search in the right subtree
            if self.right:
                self.right.search(val)
            else:  print("Element is not found in tree!")
```

Example: Search(root,16)



Delete Node

In a binary search tree, the deletion operation is performed with **O(log n)** time complexity. Deleting a node from Binary search tree includes following three cases...

- ✓**Case 1: Deleting a Leaf node (A node with no children)**
- ✓**Case 2: Deleting a node with one child**
- ✓**Case 3: Deleting a node with two children**

1. The node to be deleted is a leaf node:

- If the node to be deleted is a leaf node, deleting the node alone is enough and no additional changes are needed.

Delete Node in BST

2. The node to be deleted has one child:

- Copy the contents of the one-child to the current node and delete the child and no other modifications are needed.

3. The node to be deleted has 2 children:

- Find the smallest node in the right subtree of the current node which is also called the inorder **successor** and replace the current node with it which instead can be done with the inorder **predecessor** and the changes would be valid.

Delete v

- If v is present, delete it
- If deleted node is a leaf, done
- If deleted node has only one child,
- “promote” that child
- If deleted node has two children,fill in the hole with

  self.left.maxval() (or self.right.minval())
- Delete self.left.maxval()—

  must be leaf or have only one child

BST Applications

In multilevel indexing in the database

For dynamic sorting

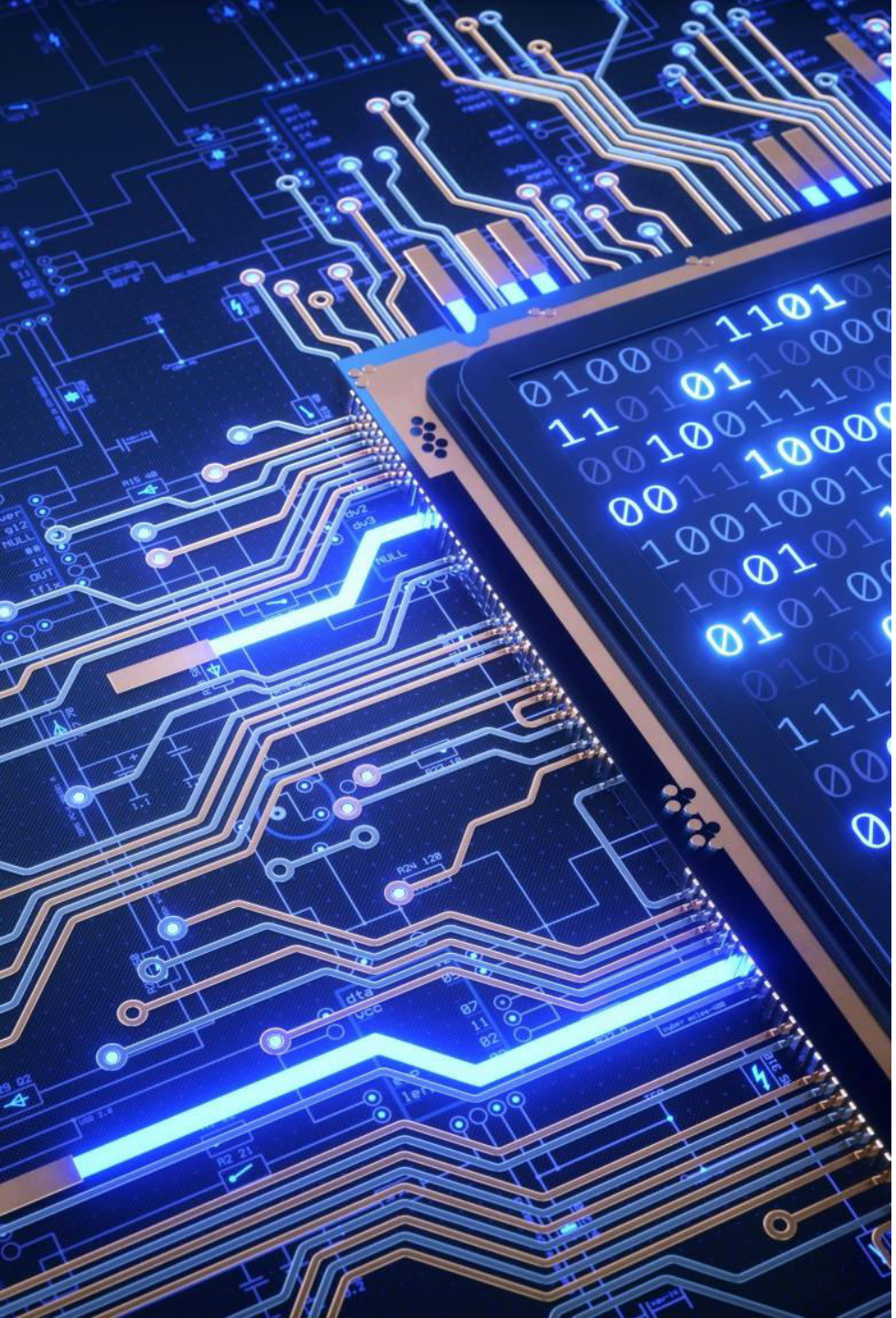
For managing virtual memory areas in Unix kernel

Any Questions ??

Data Structure with Python
Binary Tree

Mr. V. M. Vasava

GPG Surat, IT Dept.

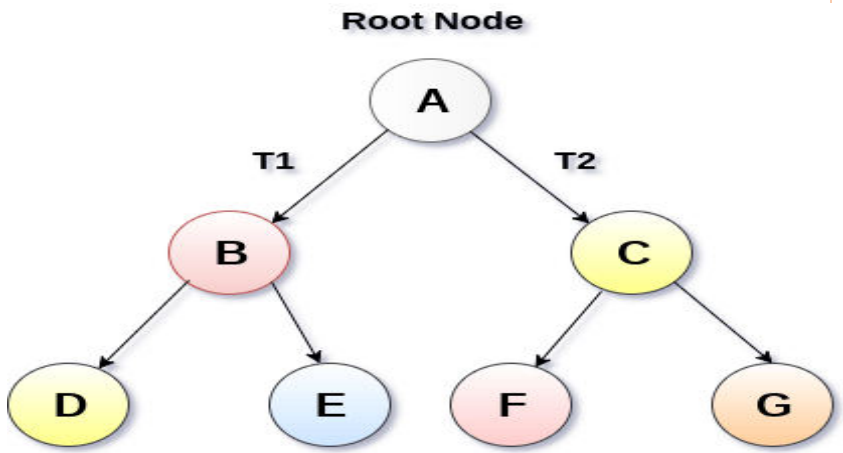


Agenda

- Introduction about BT
- Representation of Binary Tree
- Creation of BT
- Traverse of BT
- Pre-order
- In-Order
- Post-Order

Binary Tree

- A binary tree is a tree which has at most 2 children for all the nodes.
- A **binary tree** is a hierarchical data structure in which each node has **at most two children** generally referred as left child and right child.
- Binary Tree is a special type of generic tree in which, each node can have at most two children. Binary tree is generally partitioned into three disjoint subsets.
- Root of the node
- left sub-tree which is also a binary tree.
- Right binary sub-tree

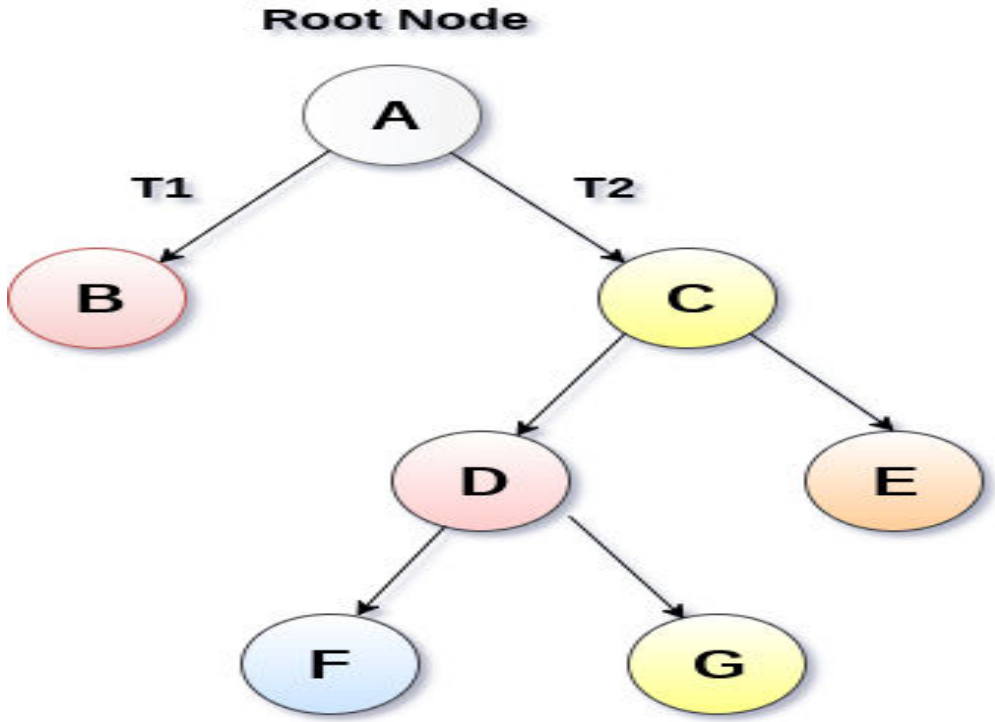


Binary Tree

Types of BT

1. strictly binary tree

- All node have either 0 or 2 children.
- In Strictly Binary Tree, every non-leaf node contain non-empty left and right sub-trees.
- In other words, the degree of every non-leaf node will always be 2.

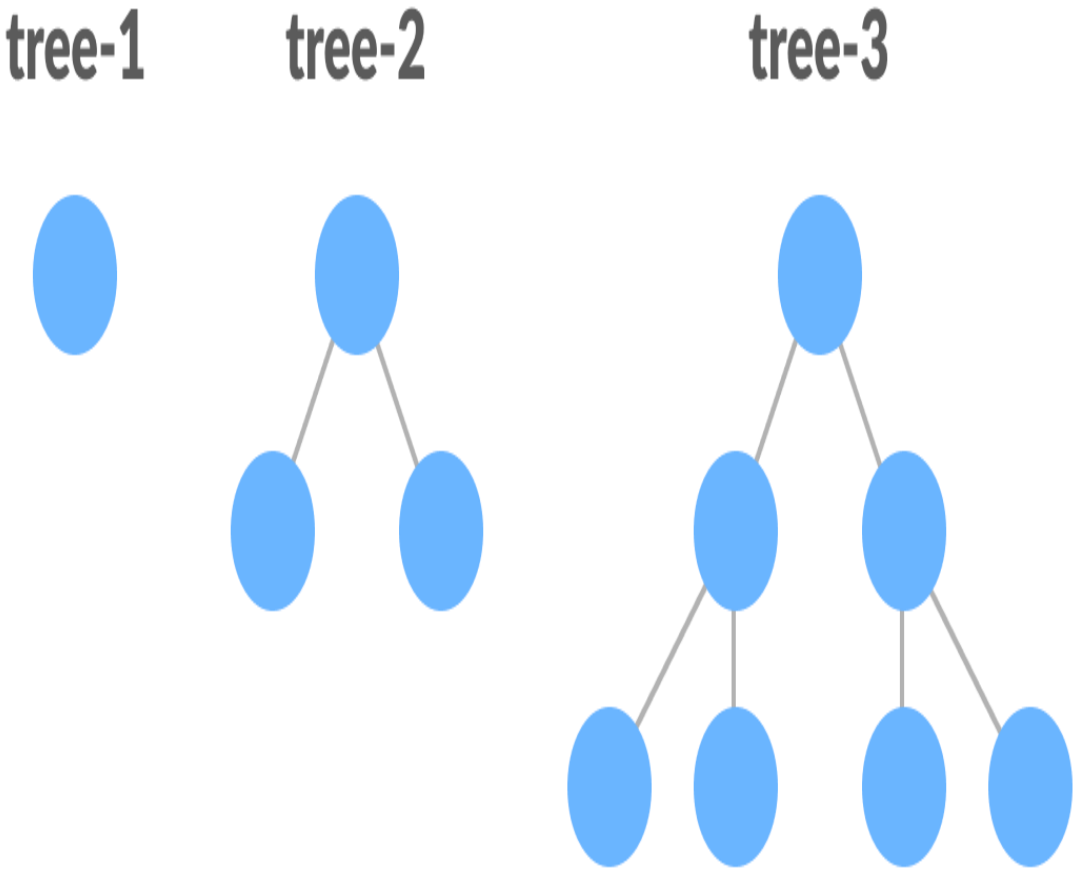


Strictly Binary Tree

2.Perfect Binary Tree

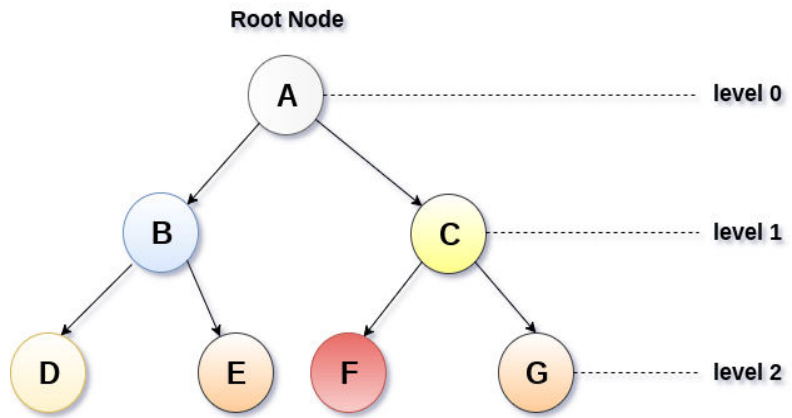
- Internal node have 2 children + all leaf node are on same level

- every internal node has exactly two child nodes and all the leaf nodes are at the same level.



3.Complete Binary Tree

- All level are completely filled except possible the last level + last level must have its keys as left as possible.
- A complete binary tree is said to be a proper binary tree where all leaves have the same depth.
- In a complete binary tree number of nodes at depth **d** is $\mathbf{2^d}$.
- In a complete binary tree with **n** nodes height of the tree is **log(n+1)**.
- All the levels **except the last level** are completely full.

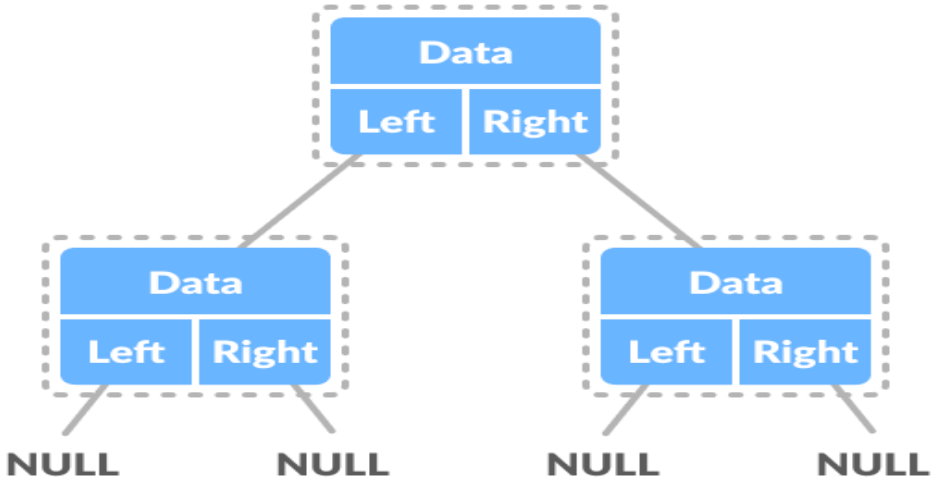


Complete Binary Tree

Implementation of BT

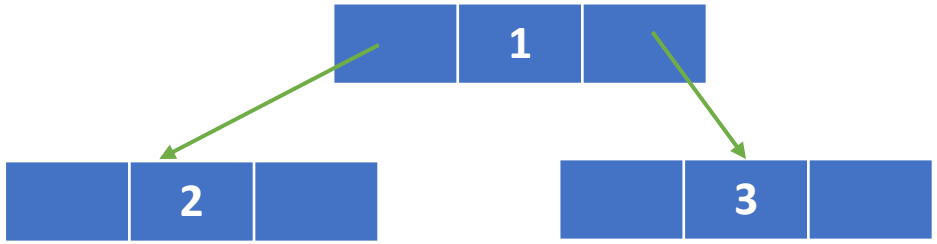
- A node of a binary tree is represented by a structure containing a data part and two pointers to other structures of the same type.

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



Create of BT

```
class BT:
    def __init__(self,Data):
        self.data=Data
        self.left=None
        self.right=None
def printTree(root):
    if root==None:
        return
    print(root.data,end=": ")
    if root.left!=None:
        print("L",root.left.data,end=" ")
    if root.right!=None:
        print("R",root.right.data,end=" ")
    print()
    printTree(root.left)
    printTree(root.right)
root=BT(1)              #create object
bt1=BT(2)
bt2=BT(3)
root.left=bt1           #attach node in left
root.right=bt2          #attach node in right
printTree(root)
```




```
1:  L  2  R  3
2:
3:

In [2]:
```

Tree Traversal

- **Tree traversal** (also known as **tree** search) is a form of graph **traversal** and refers to the process of visiting (checking and/or updating) each node in a **tree data structure**, exactly once.
  Such **traversals** are classified by the order in which the nodes are visited.

(a) Inorder (Left, Root, Right)

(b) Preorder (Root, Left, Right)

(c) Postorder (Left, Right, Root)

Pre-Order(Root)

- In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.
- Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on of an expression tree.

- `Algorithm Preorder(tree)`
- `   1. Visit the root.`
- `   2. Recursively Traverse the left subtree`
- `   3. Recursively Traverse the right subtree`

In-Order(Root)

- In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.
- If a binary tree is traversed **in-order**, the output will produce sorted key values in an ascending order.

- Algorithm: Inorder(tree)
- Until all nodes are traversed −

Step 1 − Recursively traverse left subtree.

Step 2 − Visit root node.

Step 3 − Recursively traverse right subtree.

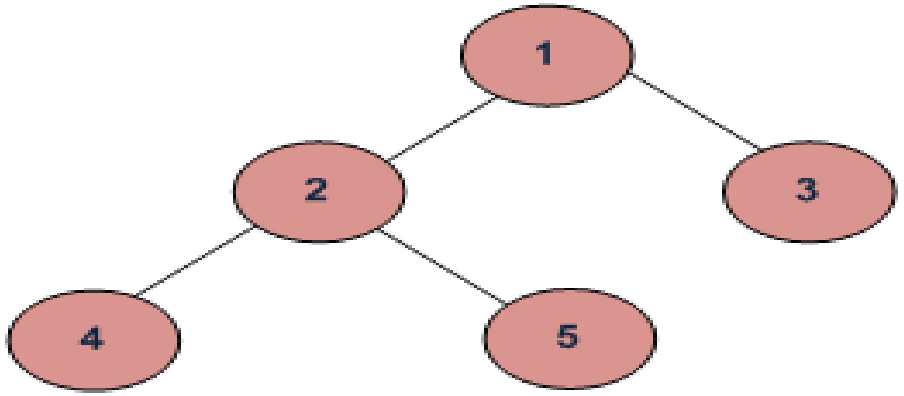
Postorder(root)

- In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.

- Algorithm: Postorder(tree)
- Until all nodes are traversed −
-
- **Step 1** − Recursively traverse left subtree.
- **Step 2** − Recursively traverse right subtree.
- **Step 3** − Visit root node**.**

Example

- Traverse the tree inorder,pre-order,postorder as given tree:



- Output:
- (a) Inorder (Left, Root, Right) : 4 2 5 1 3
  (b) Preorder (Root, Left, Right) : 1 2 4 5 3
  (c) Postorder (Left, Right, Root) : 4 5 2 3 1

Any Questions???