SEMESTER-IV

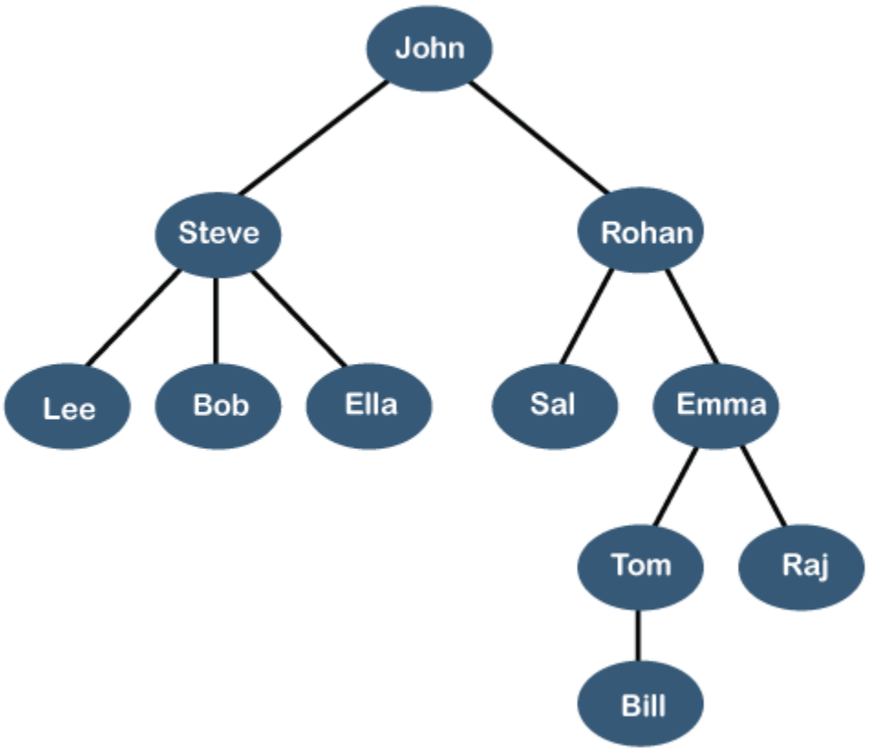
BSc Second YEAR, 2021

UNIT III

Syllabus :

Trees: Binary Tree, Definition, Properties, ADT, Array and Linked representations, Implementations and Applications. Binary Search Trees (BST) – Definition, ADT, Operations and Implementations, BST Applications. Introduction to Threaded Binary Trees, Heap trees.

Tree: A Tree is a Non-linear data structure. It stores the data in hierarchical manner. Suppose we want to show the employees of an organization in the hierarchical form as shown below.



Characteristics of a Tree:

- A tree data structure is defined as a collection of objects or entities known as nodes that are linked together to represent or simulate hierarchy.
- A tree data structure is a non-linear data structure because it does not store in a sequential manner. It is a hierarchical structure as elements in a Tree are arranged in multiple levels.
- In the Tree data structure, the topmost node is known as a root node. Each node contains some data, and data can be of any type. In the above tree structure, the node contains the name of the employee, so the type of data would be a string.
- Each node contains some data and the link or reference of other nodes that can be called children.

Basic Tree Terminology:

Root: The root node is the topmost node in the tree hierarchy. In other words, the root node is the one that doesn't have any parent. If a node is directly linked to some other node, it would be called a parent-child relationship.

Parent: If the node contains any sub-node, other than the root node then that node is called the parent of that sub-node.

Child node: If the node is a descendant of any node, then the node is a child node.

Leaf Node: The node which doesn't have any child node is called a leaf node. A leaf node is the bottom-most node of the tree. There can be any number of leaf nodes present in a general tree.

Sibling: The nodes that have the same parent are known as siblings.

Subtree: Subtree represents the descendants of a node.

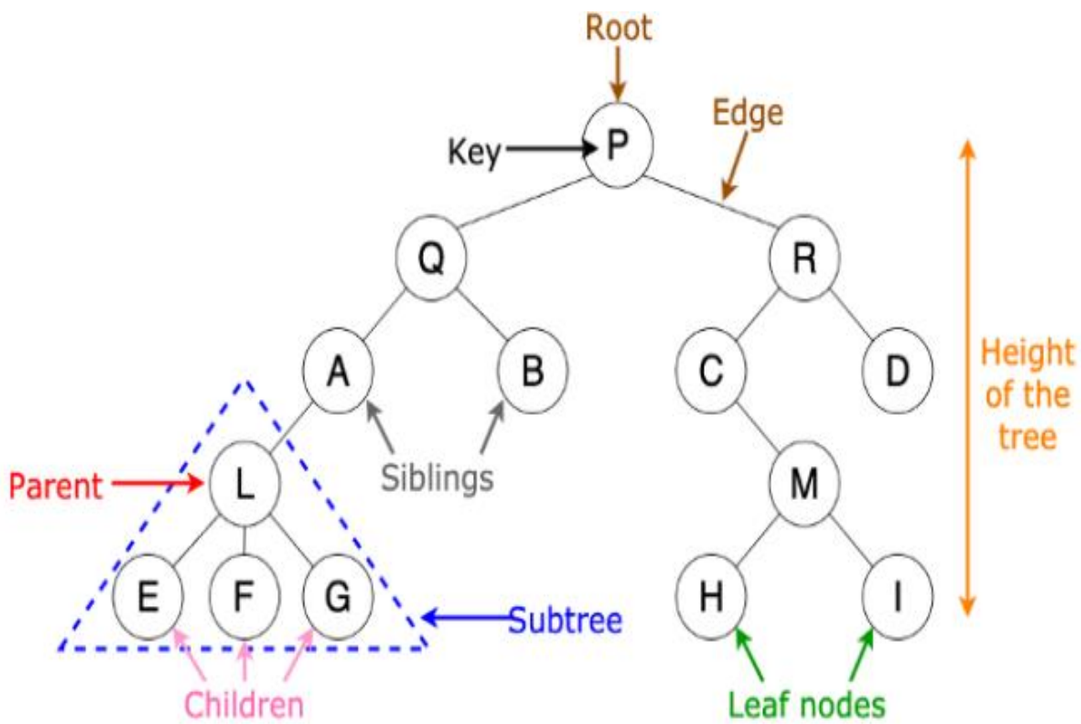
Traversing: Visiting every node in a specific order.

Levels: Level of a node represents the generation of a node. If the root node is at level 0, then its next child nodes are at level 1 and grand child nodes are at level 2 and so on.

Degree: Maximum number of children that is possible for a node is called the degree of a node.

Height: The height of node x can be defined as the longest path from the node x to the leaf node.

Link: Link is used to pointer to the other nodes in a tree. Example is Left child and Right Child are the links to the root or parent.



Q) Binary Tree

A Binary tree is a data structure that is defined as collection of elements. In Binary tree every node contains maximum of two child nodes that means it may contain either 0,1 or 2 nodes only. A node with zero children's is called leaf nodes.

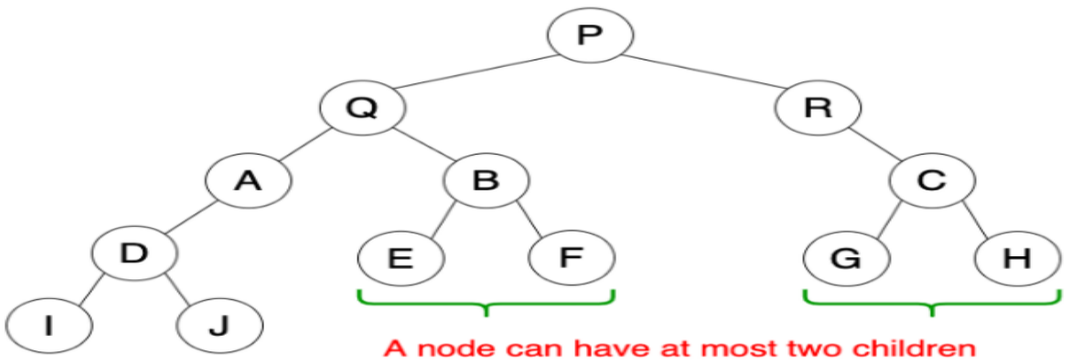
Every node contains a data element, a left pointer which points to the left child and a right pointer which points to the right child.

The root element is pointed by the root element. If root=NULL then the tree is empty.

Example for the Binary tree

In below example P is root node and Q, R are the left and right child nodes along with the sub trees.

The left sub-tree of the root node consists of the nodes: Q,A,B,D,E,F,I and J. The right sub-tree of the root node consists of the nodes: R,C,G and H.



In the above tree, root node P has two successors: Q and R. Node Q has two successors: A and B. Node A has one successor: D. Node B has two successors: E and F. Node D has two successors: I and J. Node R has one successor: C. Node C has two successors: G and H.

Even the leaf nodes may contain the empty left sub trees and empty right sub trees. The nodes I,J,E,F,G and H has no successors then these are empty sub trees.

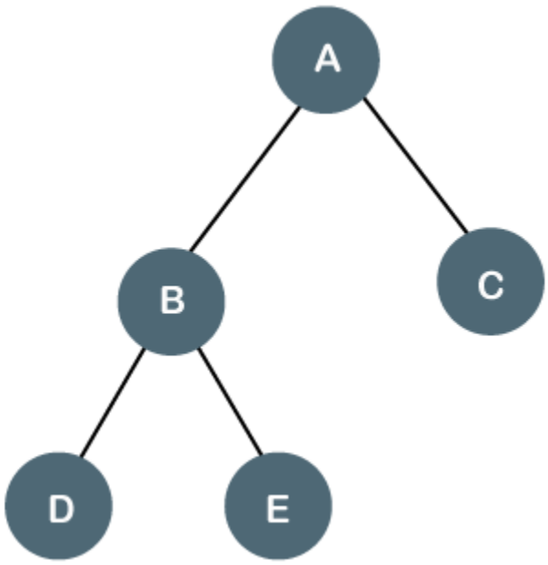
Q) Properties of Binary Tree

- At each level of **i,** the maximum number of nodes is 2^i .
- The height of the tree is defined as the longest path from the root node to the leaf node.
- The maximum number of nodes possible at height h is $(2^0 + 2^1 + 2^2 + ...2^h) = 2^{h+1} - 1$.
- The minimum number of nodes possible at height **h** is equal to **h+1**.
- If the number of nodes is minimum, then the height of the tree would be maximum. Conversely, if the number of nodes is maximum, then the height of the tree would be minimum.

Types of Binary Trees: There are four types of Binary trees. They are,

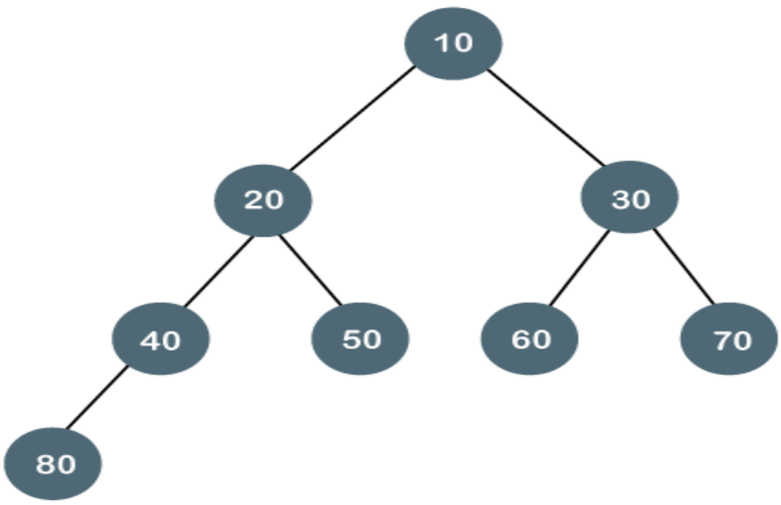
1. Full Binary tree
2. Complete Binary tree
3. Perfect Binary tree
4. Degenerate Binary tree

1. **Full Binary Tree:** The full binary tree is also called as a strict binary tree. The tree can only be considered as the full binary tree if each node must contain either 0 or 2 children. The full binary tree can also be defined as the tree in which each node must contain 2 children except the leaf nodes.



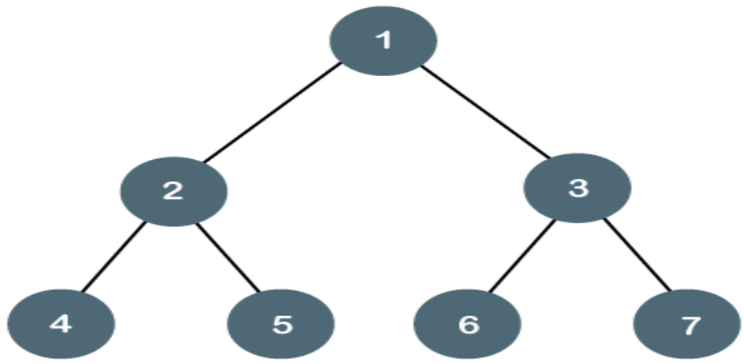
In the above tree, we can observe that each node is either containing zero or two children; therefore, it is a Full Binary tree.

2. **Complete Binary Tree:** The complete binary tree is a tree in which all the nodes are completely filled except the last level. In the last level, all the nodes must be as left as possible. In a complete binary tree, the nodes should be added from the left.



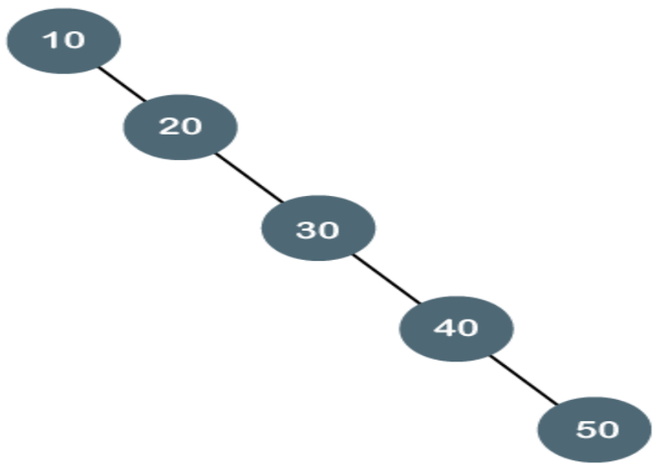
The above tree is a complete binary tree because all the nodes are completely filled, and all the nodes in the last level are added at the left first.

3. **Perfect Binary Tree:** A tree is a perfect binary tree if all the internal nodes have 2 children, and all the leaf nodes are at the same level. All the perfect binary trees are the complete binary trees as well as the full binary tree, but vice versa is not true, i.e., all complete binary trees and full binary trees are the perfect binary trees.

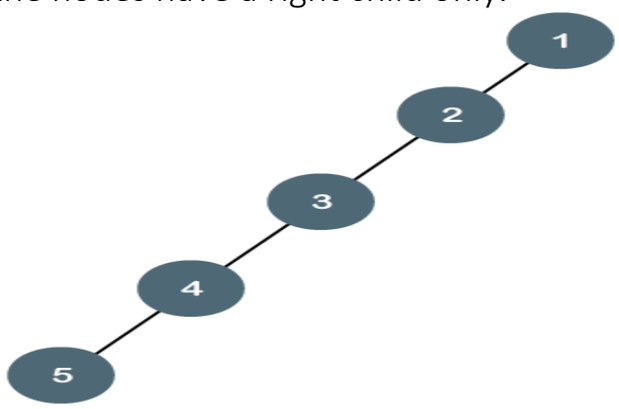


The above tree is a perfect binary tree because all nodes having 2 children except the leaf nodes.

4. **Degenerate Binary Tree:** The degenerate binary tree is a tree in which all the internal nodes have only one children. There are two types of degenerate binary trees. They are **Left-degenerate binary tree** and **Right-degenerate binary tree.**



The above tree is a degenerate binary tree because all the nodes have only one child. It is also known as a right-degenerate tree as all the nodes have a right child only.



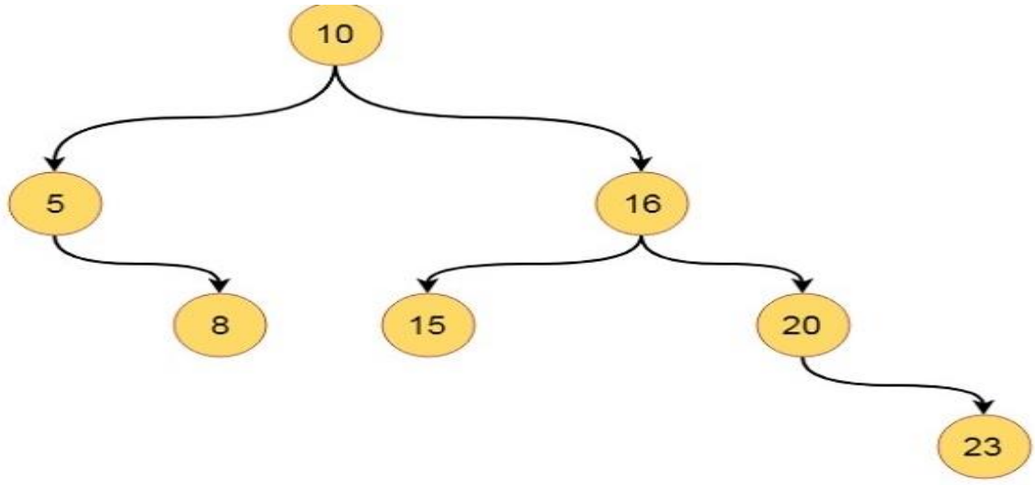
The above tree is also a degenerate binary tree because all the nodes have only one child. It is also known as a left-skewed tree as all the nodes have a left child only.

Q) Representation of Binary Trees?

A binary tree can be represented in a hierarchical relationship between a parent node and child node. A binary tree data structure is represented using two methods. Those methods are as follows...

1. Array Representation
2. Linked List Representation

1. **Array Representation:** An array representation of a binary tree, we use one-dimensional array (1-D Array) to represent a binary tree.
The array representation stores the tree data by scanning elements using level order fashion. So, it stores nodes level by level. Starting from the zero level where only one root node is present. The root node stored in the first memory location. If some element is missing, it left blank spaces for it.
For example, Consider the below tree:



Here the Array representation of the above Tree

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	5	16	-	8	15	20	-	-	-	-	-	-	-	23

The index 1 is holding the root, it has two children 5 and 16, they are placed at location 2 and 3. Some children are missing, so their place is left as blank.
In Array representation we can easily get the position of two children of one node by using this formula –

child1=2∗parent
child2=(2∗parent⍰+1

To get parent index from child we can use below formula

parent=child/2

The height of a binary tree denotes the maximum number of nodes in the longest path of the tree. A binary tree of height 'h' can have at most 2^h -1 nodes.

This approach is good, and easily we can find the index of parent and child, but it is not memory efficient. It will occupy many spaces that has no use. This representation is good for complete binary tree or full binary tree.

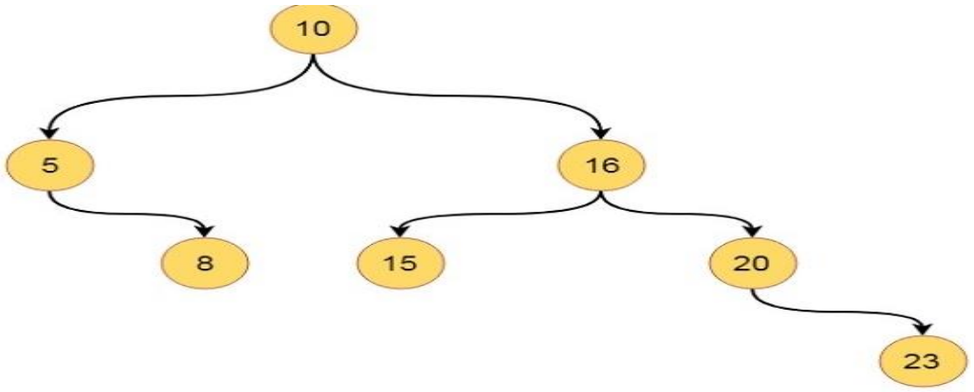
2. Linked List Representation:

To avoid the memory usage in Array representation, we can represent tree using Linked list. Here, 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.

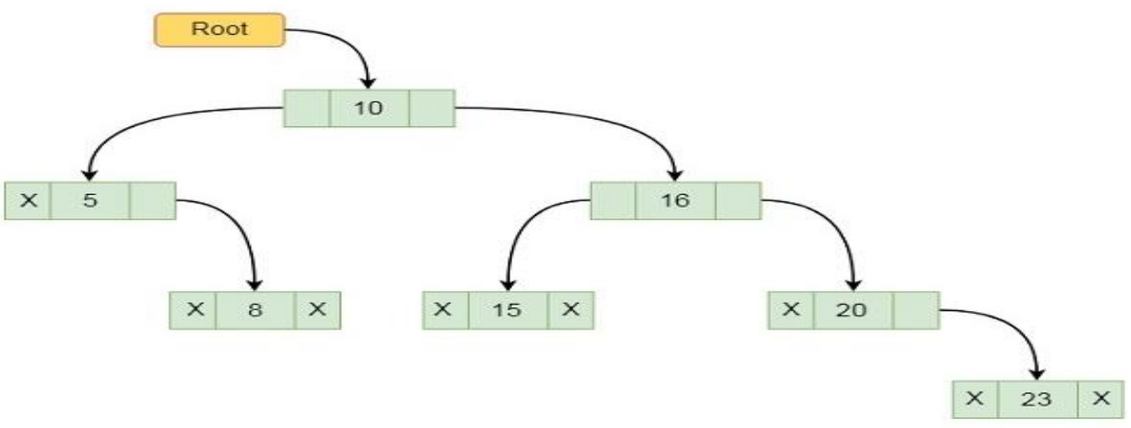
In this linked list representation, a node has the following structure...

In this representation, if one node knows the address of the any other node then from there, we can access any node.

For example, Consider the below tree:



Here the Linked representation of the above Tree



Q) Operations on Binary Tree

We can perform many operations on Binary tree. Here are the some of them.

1. Insertion
2. Deletion
3. Merge
4. Traversals

1. **Insertion:** Insertion operation is used to insert an element in the tree.
2. **Deletion:** Deletion operation is used to delete an element from tree.
3. **Merge:** Merge operation is used to merge two trees into one tree.
4. **Traversals:** Traversal is nothing but going through each and every node in tree.

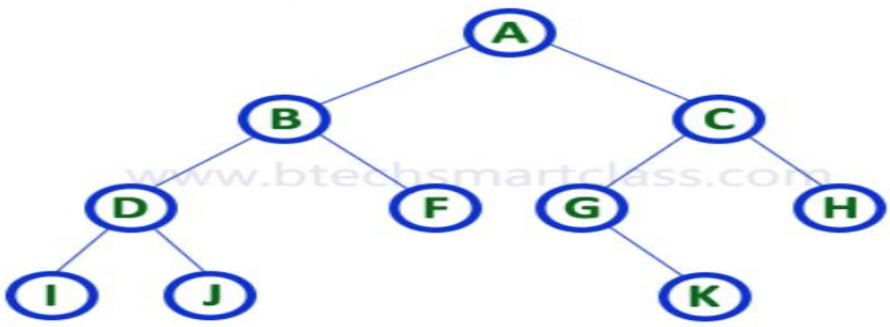
Q) Traversals of a Binary Tree:

Traversal operation is frequently used operation in binary tree. This operation is used to visit each node in the tree exactly once.
A full traversal on a binary tree gives a linear ordering of the data in the tree. We have 3 types of Traversals in binary tree. They are

- Pre-Order Traversals
- In-Order Traversals
- Post-Order Traversals

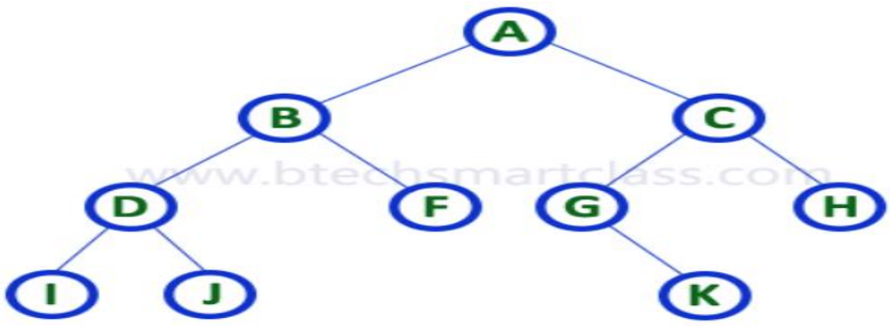
Pre-Order Traversals: In Pre-Order traversal, the root node is visited before the left child and right child nodes. Pre-order traversal is applicable for every root node of all subtrees in the tree. It can be defined as

- Visit root node first
- Visit the left sub-tree of the root
- Visit the right sub-tree of the root



Pre-Order Traversal for the example of the above given binary tree is
A - B - D - I - J - F - C - G - K – H

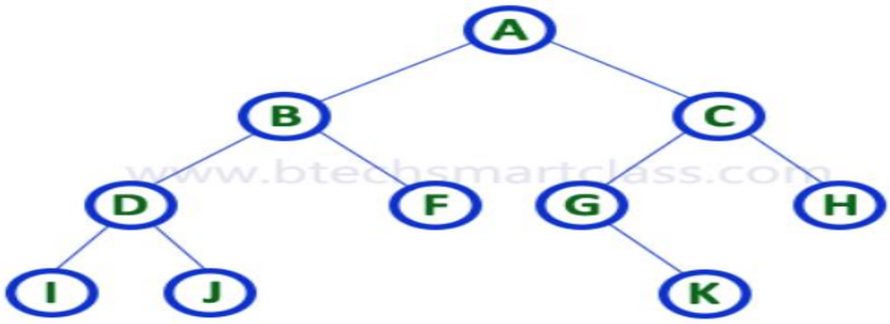
In-Order Traversal: In In-Order traversal, the root node is visited between the left child and right child. In this traversal, the left child node is visited first, then the root node is visited and later we go for visiting the right child node. This in-order traversal is applicable for every root node of all subtrees in the tree. This is performed recursively for all nodes in the tree. Consider the below binary tree



In-Order Traversal for the example of binary tree is

I - D - J - B - F - A - G - K - C – H

Post-Order Traversals: In Pre-Order traversal, the root node is visited before the left child and right child nodes. In this traversal, the root node is visited first, then its left child and later its right child. This pre-order traversal is applicable for every root node of all subtrees in the tree.



Pre-Order Traversal for the example of binary tree is

A - B - D - I - J - F - C - G - K - H

Q) Applications of Binary Trees:

Here is the list of Applications of Binary Trees

Binary Search Tree - Used in many search applications where data is constantly entering/leaving

Binary Space Partition - Used in almost every 3D video game to determine what objects need to be rendered.

Binary Tries - Used in almost every high-bandwidth router for storing router-tables.

Hash Trees - used in p2p programs and specialized image-signatures in which a hash needs to be verified, but the whole file is not available.

Heaps - Used in implementing efficient priority-queues, which in turn are used for scheduling processes in many operating systems, Quality-of-Service in routers, and A* (path-finding algorithm used in AI applications, including robotics and video games). Also used in heapsort.

Huffman Coding Tree (Chip Uni) - used in compression algorithms, such as those used by the .jpeg and .mp3 file-formats.

GGM Trees - Used in cryptographic applications to generate a tree of pseudo-random numbers.

Syntax Tree - Constructed by compilers and (implicitly) calculators to parse expressions.

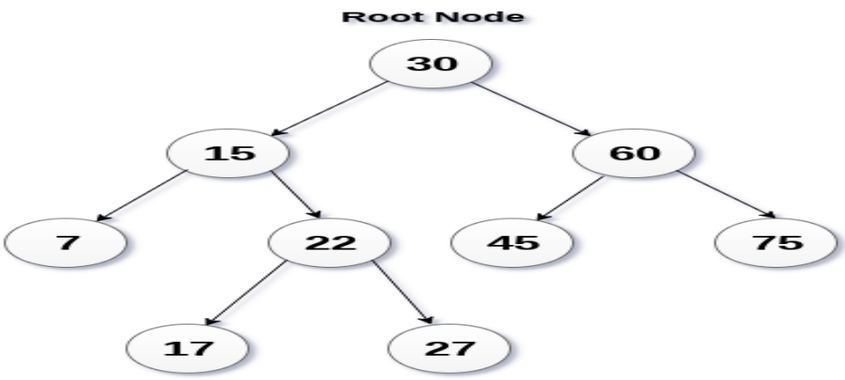
Treap - Randomized data structure used in wireless networking and memory allocation.

T-tree - Though most databases use some form of B-tree to store data on the drive, databases which keep all (most) their data in memory often use T-trees to do so

Q) Binary Search Tree and ADT of Binary Search tree?

A Binary tree is also a binary search tree if it satisfies the following properties:

- Binary Search tree can be defined as a class of binary trees, in which the nodes are arranged in a specific order. This is also called ordered binary tree.
- In a binary search tree, the value of all the nodes in the left sub-tree is less than the value of the root node.
- Similarly, value of all the nodes in the right sub-tree is greater than or equal to the value of the root.
- This rule will be recursively applied to all the left and right sub-trees of the root.



A Binary search tree is shown in the above figure. As the constraint applied on the BST, we can see that the root node 30 doesn't contain any value greater than or equal to 30 in its left sub-tree and it also doesn't contain any value less than 30 in its right sub-tree.

Advantages of using binary search tree

1. Searching become very efficient in a binary search tree since, we get a hint at each step, about which sub-tree contains the desired element.
2. The binary search tree is considered as efficient data structure in compare to arrays and linked lists. In searching process, it removes half sub-tree at every step.
3. It also speed up the insertion and deletion operations as compare to that in array and linked list.

ADT of Binary Search Tree: Abstract Data type of Binary search tree can be represented below

```
Abstract datatype
{
    Instances: * T has a special node called root node
               *  Left sub tree nodes has smaller values then its root
               * right sub tree nodes has lager values then its root
    Methods: put(x) – it is used to insert an element X
             Get(x) – it is used to get an element x
             Remove(x) – it is used to remove an element x
}
```

Q) Operations on Binary Search Tree

There are many operations which can be performed on a binary search tree. Below are the operations which are performed frequently

- Searching in BST
- Insertion in BST
- Deletion in BST

Searching in BST: Searching means finding or locating some specific element or node within a tree data structure. However, searching for some specific node in binary search tree is very easy because the elements or nodes in BST are stored in a specific order.

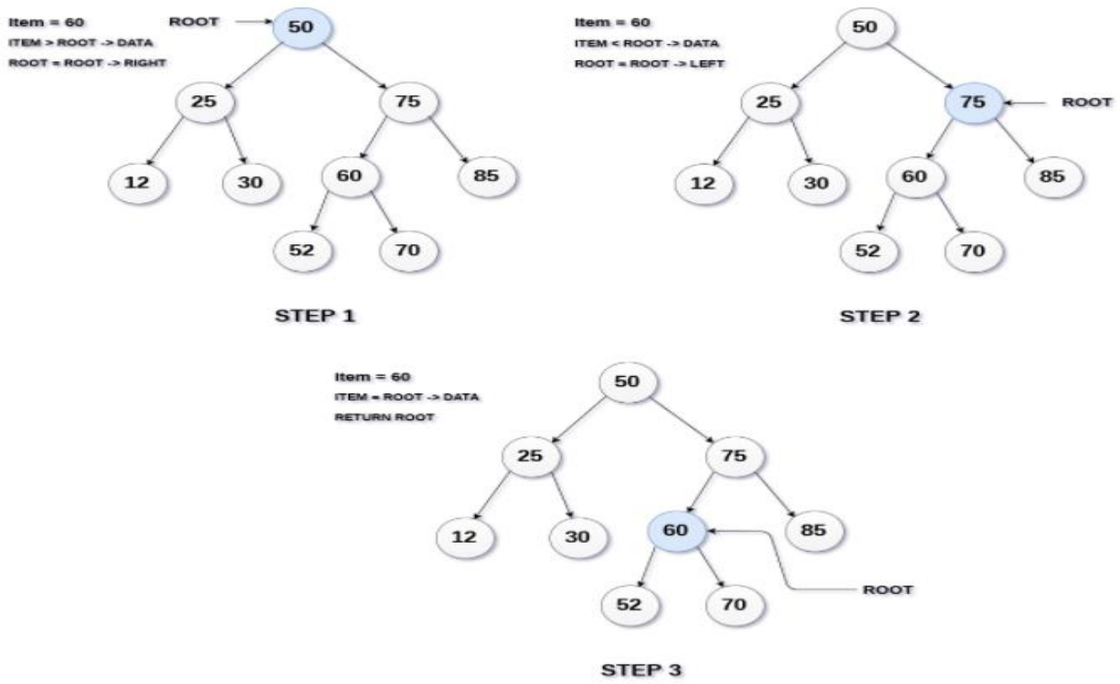
1. Compare the element with the root node of the tree.
2. If the item is matched then return the location of the root node.
3. Otherwise check if item is less than the element present on root, if so then move to the left sub-tree.
4. If not, then move to the right sub-tree.
5. Repeat 3 and 4 recursively until match found.
6. If element is not found then return NULL.

We will use the below algorithm to search an element

```
Step 1: IF ROOT =  NULL then
            write " my tree is empty"
        END IF
        Goto Step 3
Step 2: IF ITEM = ROOT□DATA then
            print "ROOT DATA and address"
        ELSEIF
            ITEM < ROOT □ DATA then
```

Search (LEFT Tree, ITEM)
Repeat this until you find the ITEM
ELSE
Search(RIGHT Tree, ITEM)
Repeat this until you find the ITEM
END IF
ENDIF
Step 3: Exit

Example: Search element/item 60 from the given tree



Insertion in BST: Insert is used to add a new element in a binary search tree at appropriate location. Insert is designed in such a way that; it must not violate the property of binary search tree at each value.

Here are steps to insert an element in Binary search tree
1. Allocate the memory for tree.
2. Set the data part to the value and set the left and right pointer of tree, point to NULL.
3. If the item to be inserted, will be the first element of the tree, then the left and right of this node will point to NULL.
4. Else, check if the item is less than the root element of the tree, if this is true, then recursively perform this operation with the left of the root.
5. If this is false, then perform this operation recursively with the right sub-tree of the root.

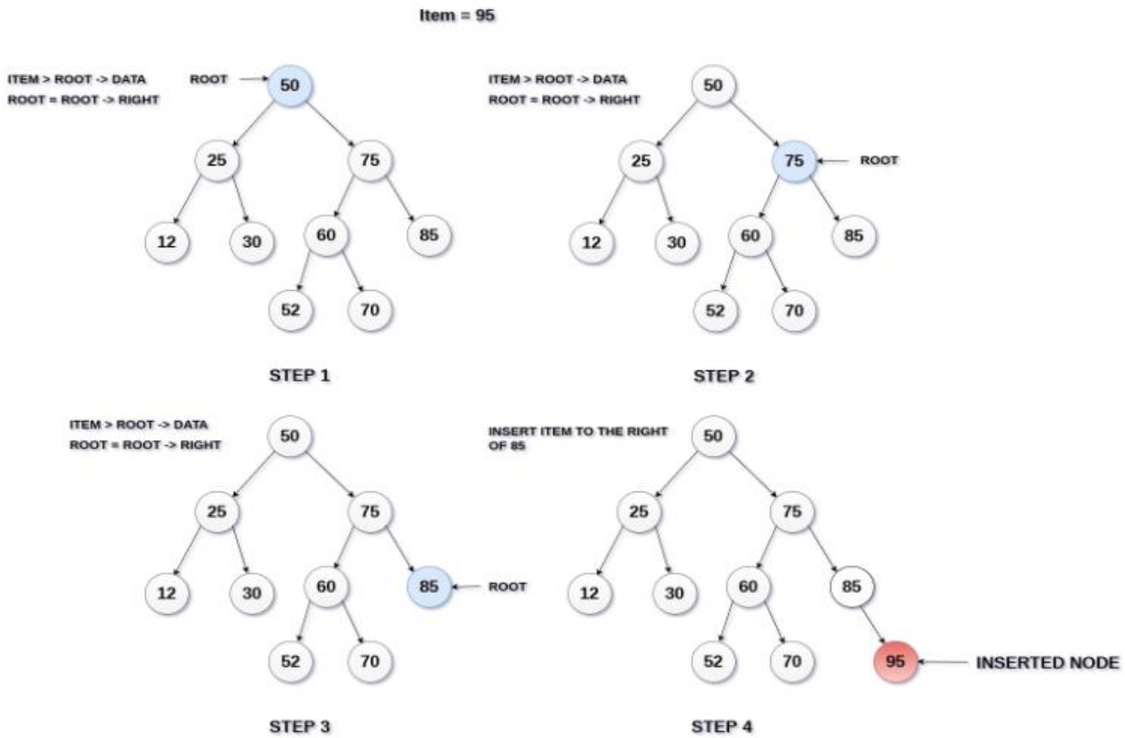
We will use the below algorithm to insert an element

Step 1: IF ROOT = NULL then
ROOT→DATA = ITEM
END IF
Goto Step 3

Step 2: IF ITEM < ROOT →DATA then
Insert (LEFT Tree, ITEM)
Repeat Step 2 until you insert the ITEM
ELSE
Search(RIGHT Tree, ITEM)
Repeat Step 2 until you Insert the ITEM
END IF

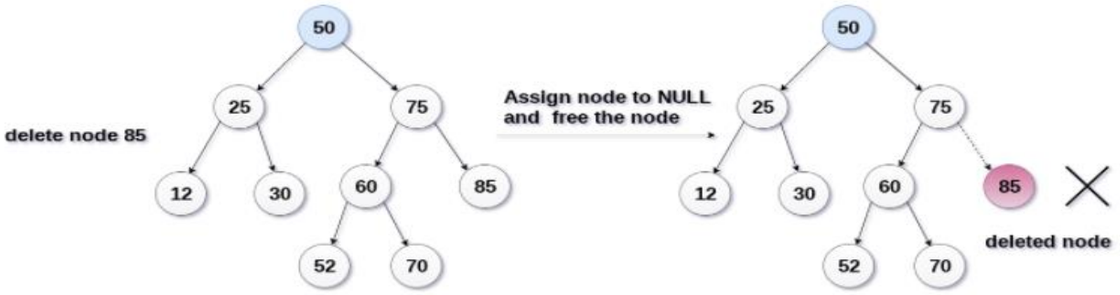
Step 3: Exit

Example: Insert an element 95 to the tree



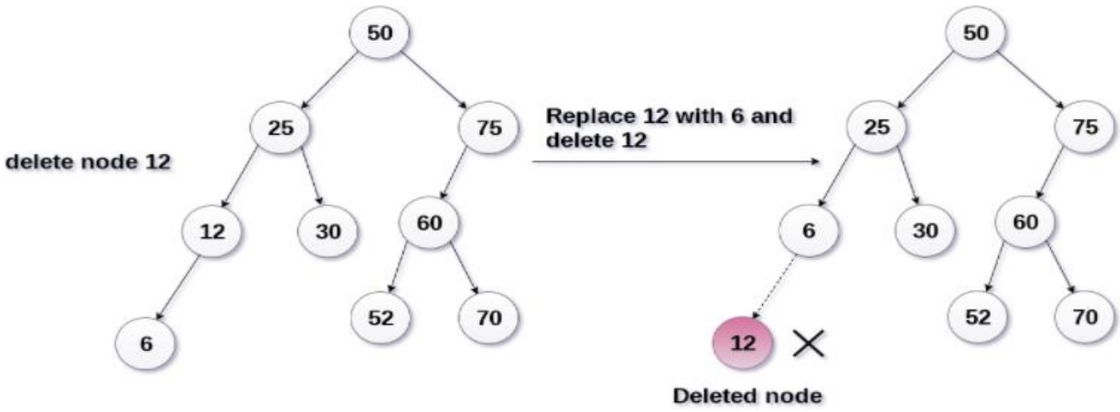
Deletion in BST: Delete operation is used to delete the specified node from a binary search tree. However, we must delete a node from a binary search tree in such a way, that the property of binary search tree doesn't violate. There are three situations of deleting a node from binary search tree.

The node to be deleted is a leaf node: This is very simple, in this just replace the leaf node with the NULL and simple free the allocated space. In the following image, we are deleting the node 85, since the node is a leaf node, therefore the node will be replaced with NULL and allocated space will be freed.



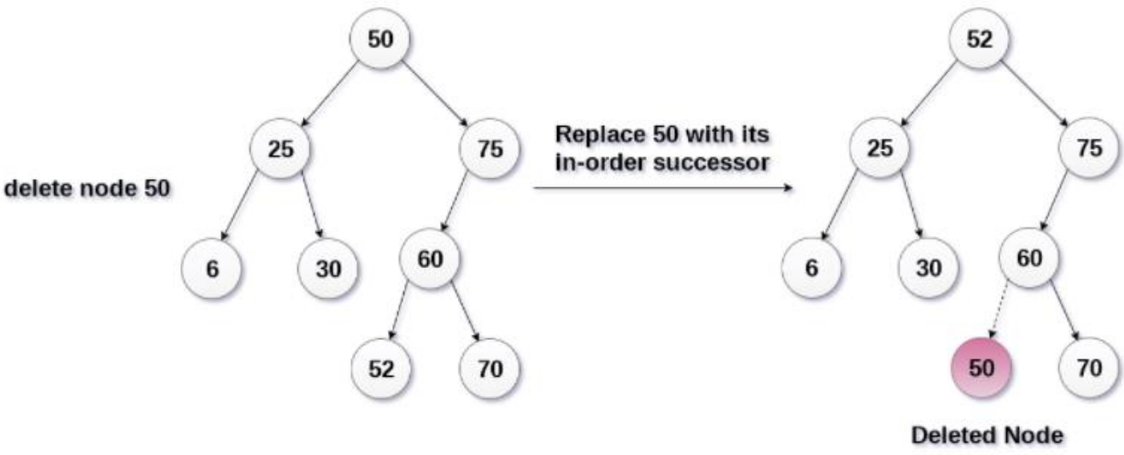
The node to be deleted has only one child: In this , replace the node with its child and delete the child node, which now contains the value which is to be deleted. Simply replace it with the NULL and free the allocated space.

In the following image, the node 12 is to be deleted. It has only one child. The node will be replaced with its child node and the replaced node 12 (which is now leaf node) will simply be deleted.



The node to be deleted has two children: It is a bit complexed case compare to other two cases. However, the node, which is to be deleted, is replaced with its in-order successor or predecessor recursively until the node value (to be deleted) is placed on the leaf of the tree. After the procedure, replace the node with NULL and free the allocated space.

In the following image, the node 50 is to be deleted which is the root node of the tree. The in-order traversal of the tree given below.



We will use the below algorithm to delete an element

Step 1: IF ROOT = NULL then
Write " UNDERFLOW"
END IF
Goto Step 3
Step 2: IF ITEM = ROOT□DATA then
Delete ROOT□DATA
ELSEIF ITEM < ROOT □ DATA then
Delete (LEFT Tree, ITEM)
Repeat Step 2 until you delete the ITEM
ELSE
Delete(RIGHT Tree, ITEM)
Repeat Step 2 until you delete the ITEM
END IF
Step 3: Exit

Q) Applications of Binary Search Tree:

Binary Search Trees are used for a lot of applications due to its ordered structure.

1. Binary search trees can be used to Represent Arithmetic expressions.
2. Binary search trees can be used to Manipulate hierarchical data.
3. Binary search trees can be used to store data efficiently.
4. It stores data efficiently because of this the access is faster and simple
5. Searching is very faster in Binary search trees compared to basic trees
6. Binary Search Trees are used to implement various searching algorithms.
7. TreeMap and TreeSet data structures are internally implemented using self-balancing BSTs.
8. A Binary Search Tree is used to implement doubly ended priority queue.

Q) Threaded Binary Tree:

We know that the binary tree nodes may have at most two children. But if they have only one children, or no children, the link part in the linked list representation remains null. Using threaded binary tree representation, we can reuse that empty links by making some threads. If one node has some vacant left or right child area, that will be used as thread. There are two types of threaded binary tree. They are

1. Single threaded Binary tree
2. Fully threaded binary tree.

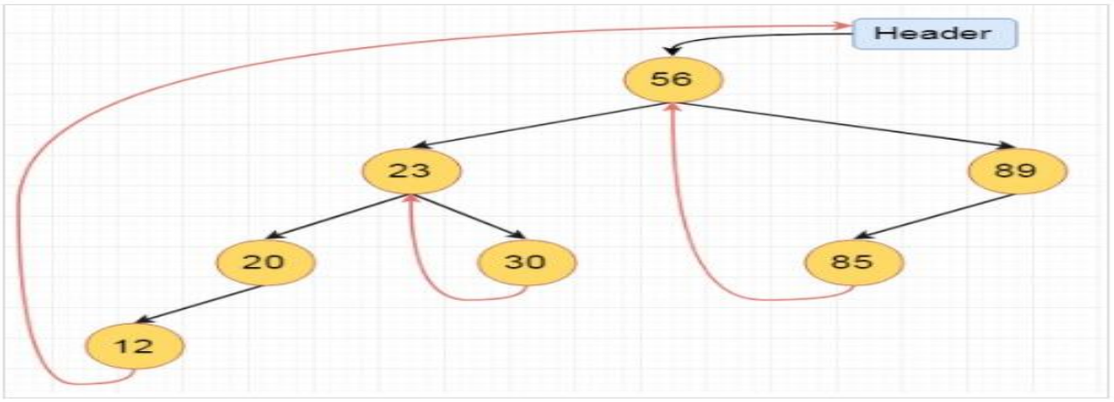
The **Single threaded Binary Tree**, there are another two variations. **Left threaded Binary Tree** and **Right threaded Binary Tree.**

In the **left threaded Binary Tree** if some node has no left child, then the left pointer will point to its inorder predecessor.

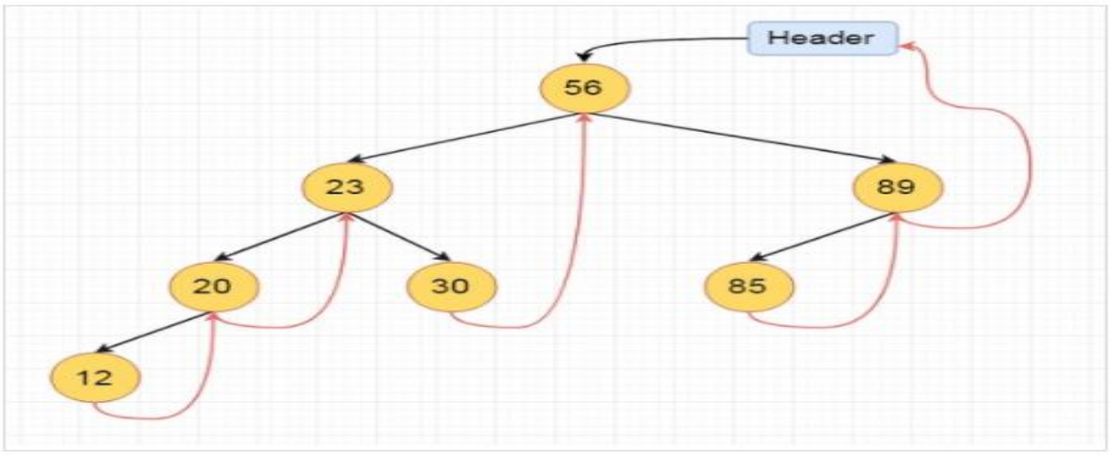
In the **right threaded Binary Tree** if some node has no right child, then the right pointer will point to its inorder successor. In both cases, if no successor or predecessor is present, then it will point to header node.

Below are the examples for Left and right threaded binary tree:

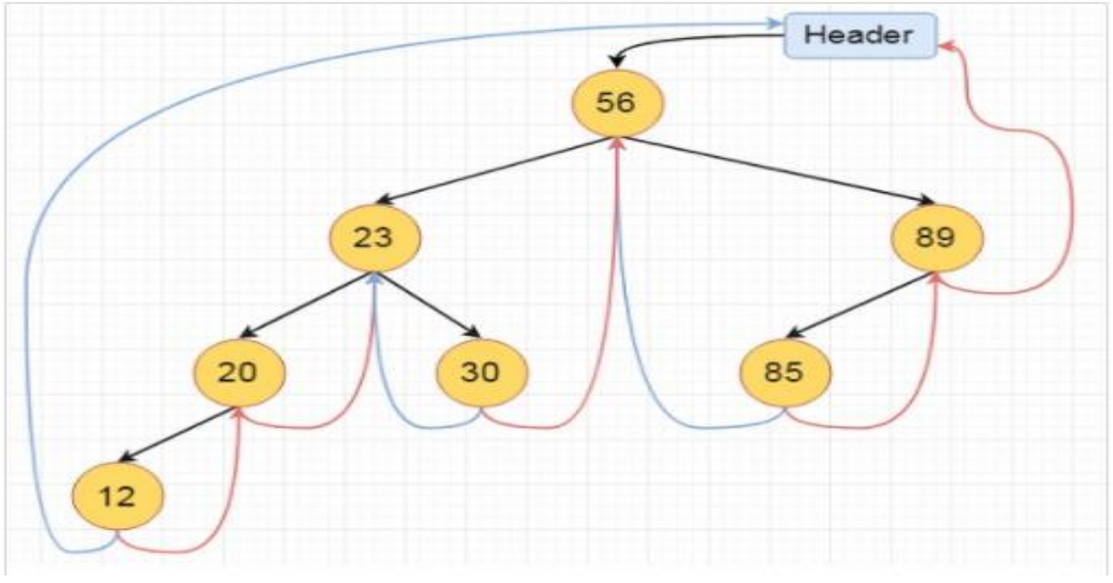
Left Threaded Binary Tree



Right Threaded Binary Tree:



For **fully threaded binary tree**, each node has five fields. Three fields like normal binary tree node, another two fields to store Boolean value to denote whether link of that side is actual link or thread.



Q) Heap Trees:

A heap is a specialized binary tree, and the binary tree is a tree in which the node can have utmost two children. The heap tree is a special balanced binary tree data structure where the root node is compared with its children and arrange accordingly.

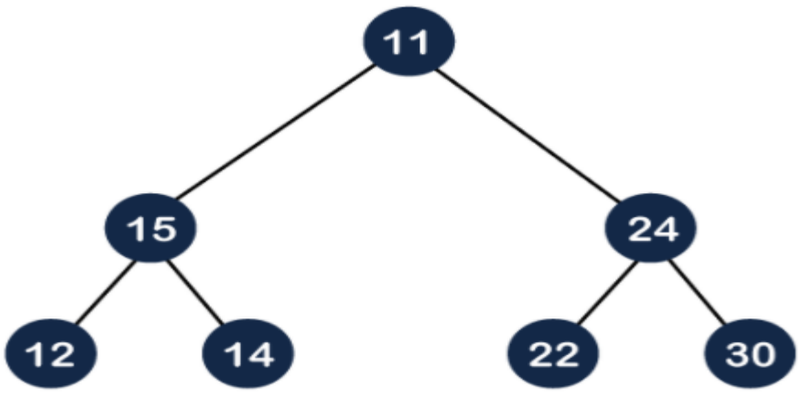
We can represent the Heap trees in two ways

1. Max-Heap tree
2. Min-Heap Tree

1. Min-Heap Trees:

The value of the parent node should be less than or equal to either of its children. In other words, the min-heap can be defined as, for every node i, the value of node i is greater than or equal to its parent value except the root node. Mathematically, it can be defined as:

A[Parent(i)] <= A[i]

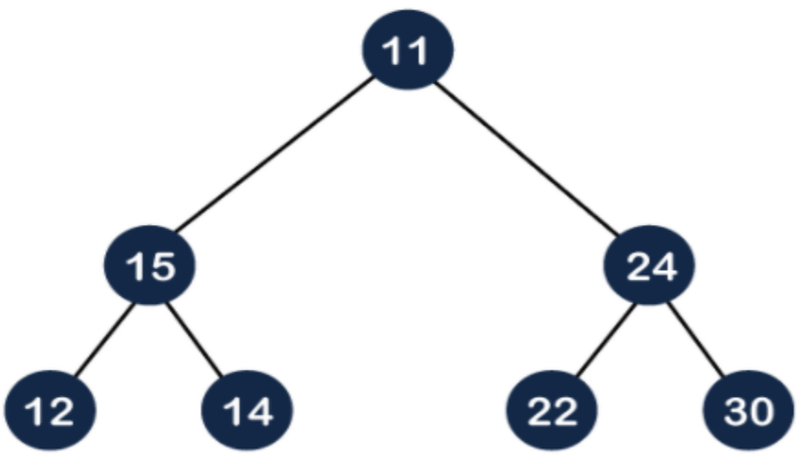


In the above figure, 11 is the root node, and the value of the root node is less than the value of all the other nodes (left child or a right child).

2. Max-Heap Trees:

The value of the parent node is greater than or equal to its children. In other words, the max heap can be defined as for every node i; the value of node i is less than or equal to its parent value except the root node. Mathematically, it can be defined as:

A[Parent(i)] >= A[i]



In the above figure, 44 is the root node, and the value of the root node is greater than the value of all the other nodes (left child or a right child).

Q) Draw the max-Heap and Min-Heap Trees using below values

44, 33, 77, 11, 55, 88, 66

Suppose we want to create the max heap tree. To create the max heap tree, we need to consider the following two properties:

1. we have to insert the element in such a way that the property of the complete binary tree must be maintained.
2. The value of the parent node should be greater than the either of its child.

Here are the steps to create max heap tree

Step 1: First we add the 44 element in the tree

Step 2: The next element is 33. As we know that insertion in the binary tree always starts from the left side so 44 will be added at the left of 33

Step 3: The next element is 77 and it will be added to the right of the 44

As we can observe in the above tree that it does not satisfy the max heap property, i.e., parent node 44 is less than the child 77. So, we will swap these two value

Step 4: The next element is 11. The node 11 is added to the left of 33

Step 5: The next element is 55. To make it a complete binary tree, we will add the node 55 to the right of 33

As we can observe in the above figure that it does not satisfy the property of the max heap because 33<55, so we will swap these two values

Step 6: The next element is 88. The left subtree is completed so we will add 88 to the left of 44

As we can observe in the above figure that it does not satisfy the property of the max heap because 44<88, so we will swap these two values

Again, it is violating the max heap property because 88>77 so we will swap these two values

Step 7: The next element is 66. To make a complete binary tree, we will add the 66 element to the right side of 77 as shown below:

In the above figure, we can observe that the tree satisfies the property of max heap; therefore, it is a heap tree

**********END**********