Unit 4: Non Linear List

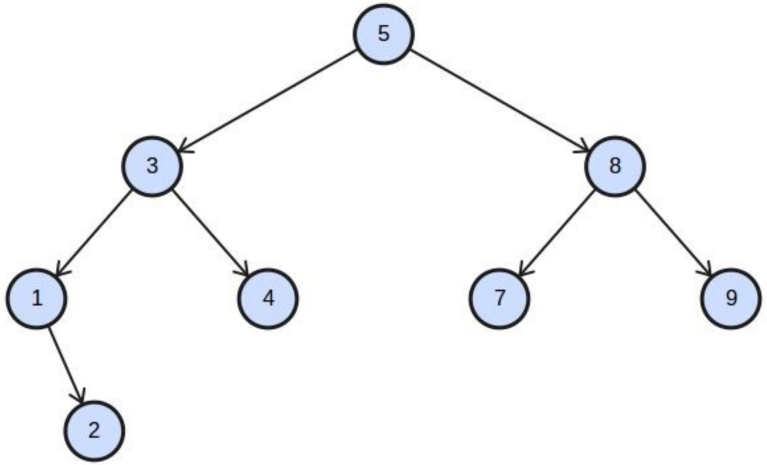
Prepared By:
Tanvi Patel
Asst. Prof. (CSE)

Contents

- **Tree**- Definitions
- Introduction to General Tree
- Representation of binary tree,
- Tree traversal -Inorder, postorder, preorder,
- Binary search trees,
- Conversion of General Trees To Binary Trees. (Refer from textbook)
- Threaded binary tree,

Definition

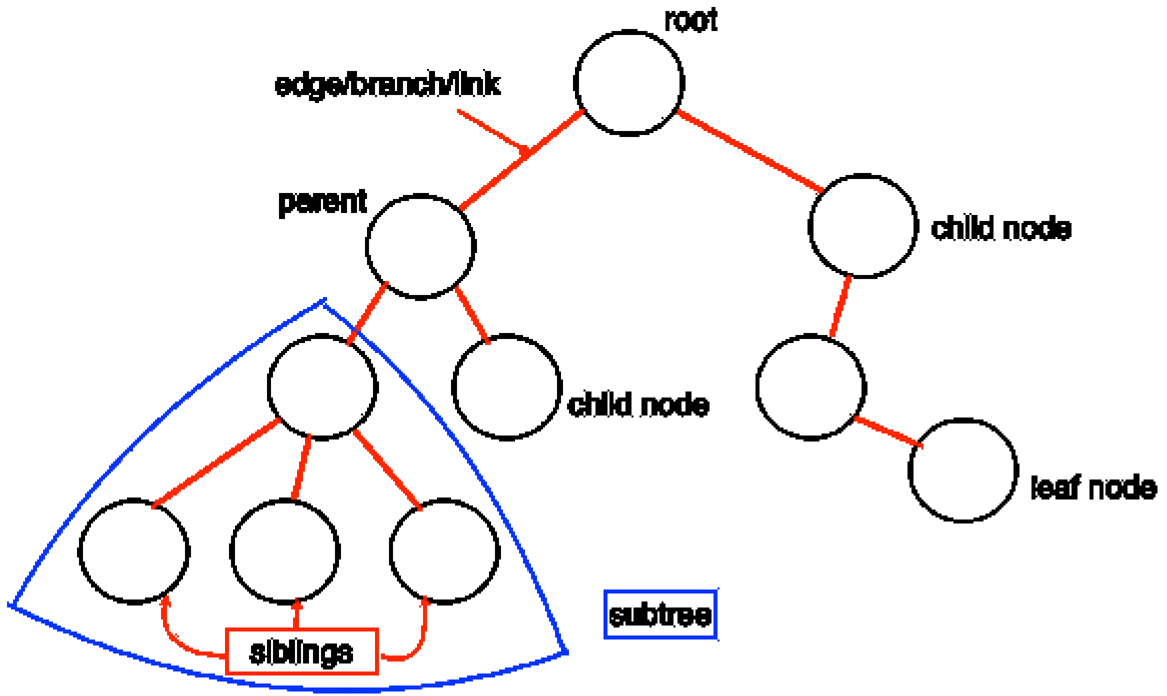
- In computer science, a **tree** is a **data structure** that is modeled after nature.
- Unlike trees in nature, the tree data structure is upside down: the root of the tree is on top.
- A tree consists of nodes and its connections are called edges.
- The bottom nodes are also named leaf nodes.
- A tree may not have a cycle.
- A tree with eight nodes.
- The root of the tree (5) is on top.
- Python does not have built-in support for trees.



Introduction

- Tree represents the nodes connected by edges. It is a non-linear data structure. It has the following properties.
- One node is marked as Root node.
- Every node other than the root is associated with one parent node.
- Each node can have an arbitrary number of child node.
- We create a tree data structure in python by using the concept of node discussed earlier.
- We designate one node as root node and then add more nodes as child nodes.

Introduction

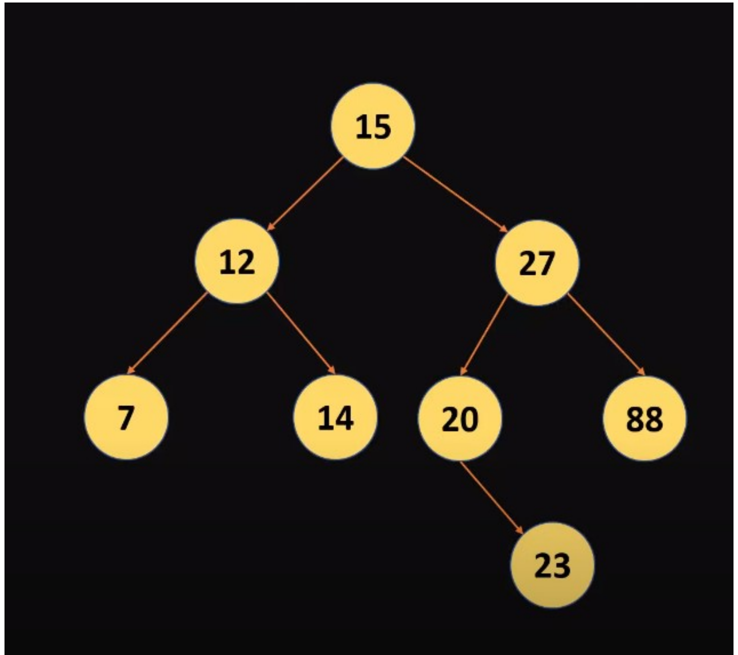


Introduction

- Tree represents the nodes connected by edges. It is a non-linear data structure. It has the following properties.
- One node is marked as Root node.
- Every node other than the root is associated with one parent node.
- Each node can have an arbiatry number of chid node.
- We create a tree data structure in python by using the concept of node discussed earlier.
- We designate one node as root node and then add more nodes as child nodes.

Binary Tree Data Structure

- A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.



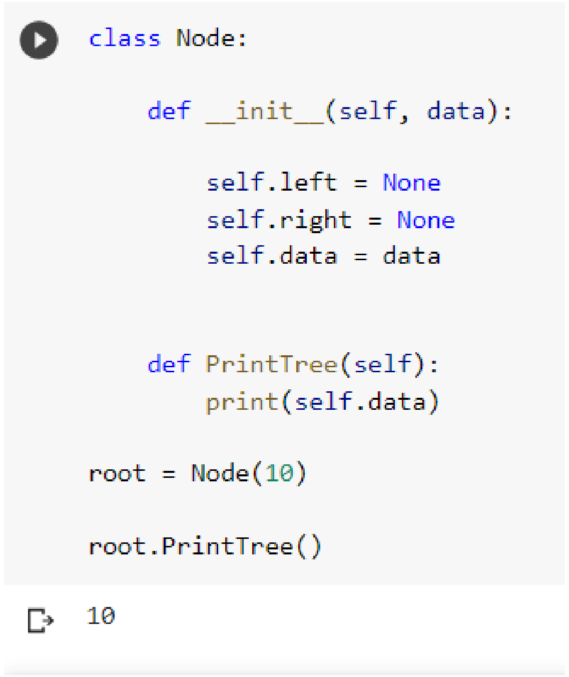
Binary Search Tree (Cont.)

- A Binary Tree node contains following parts.
- Data
- Pointer to left child
- Pointer to right child

Create Root

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

Binary Search Tree (Cont.)



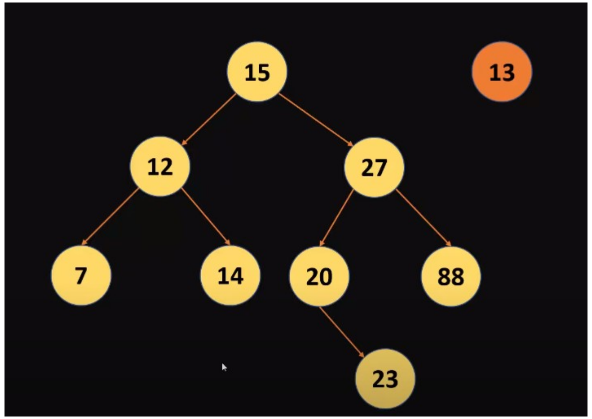
```
class Node:

    def __init__(self, data):

        self.left = None
        self.right = None
        self.data = data


    def PrintTree(self):
        print(self.data)

root = Node(10)

root.PrintTree()
```

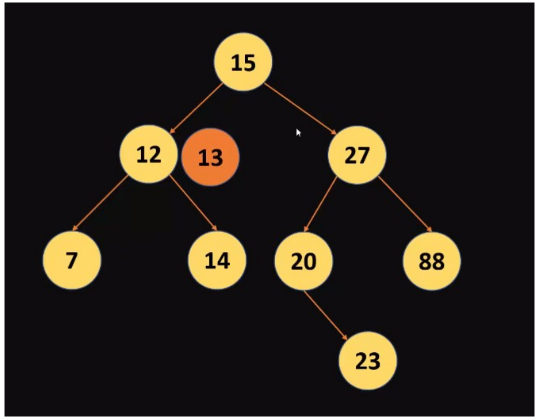
```
10
```

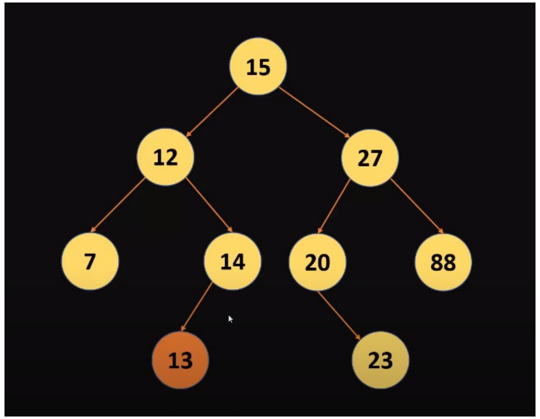
Binary Search Tree(Cont.)

Inserting into a Tree

- To insert into a tree we use the same node class created above and add an insert method to it The insert method compares the value of the node to the parent node and decides to add it as a left node or a right node.







- Insert Complexity: O(log n)

Binary Search Tree(Cont.)

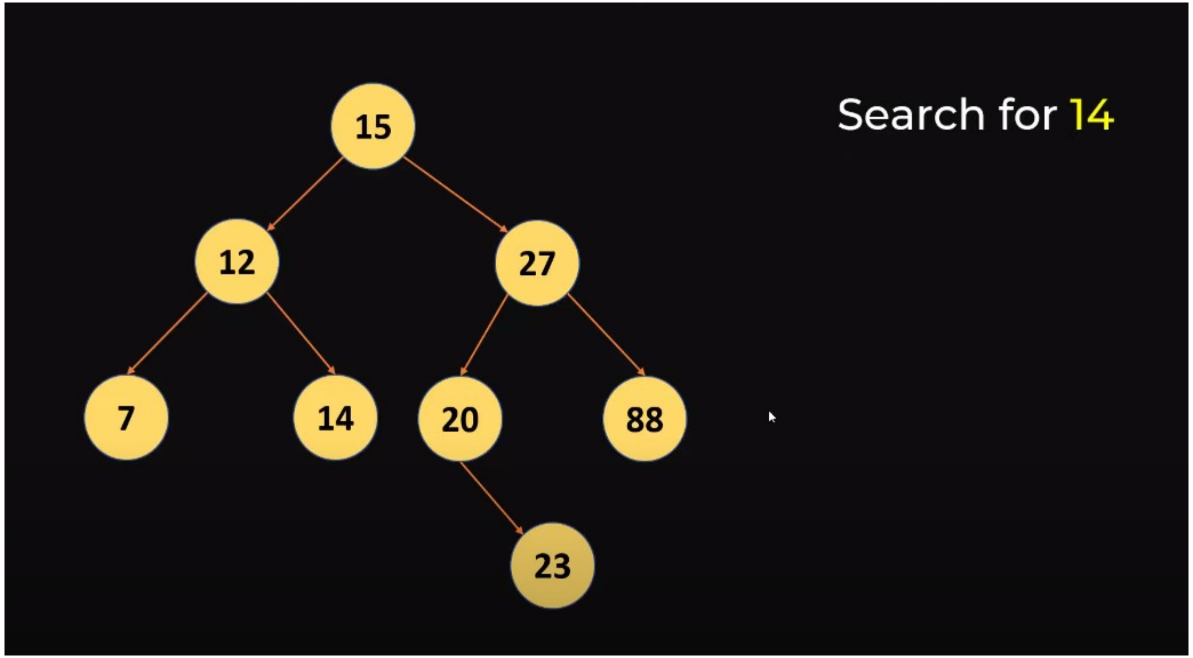
Inserting into a Tree

- In a Binary Search Tree (BST), all keys in left subtree of a key must be smaller and all keys in right subtree must be greater. So a Binary Search Tree by definition has distinct keys.
  How to allow duplicates where every insertion inserts one more key with a value.
- A **Simple Solution** is to allow same keys on right side. For example consider insertion of keys 12, 10, 20, 9, 11, 10, 12, 12 in an empty Binary Search Tree.

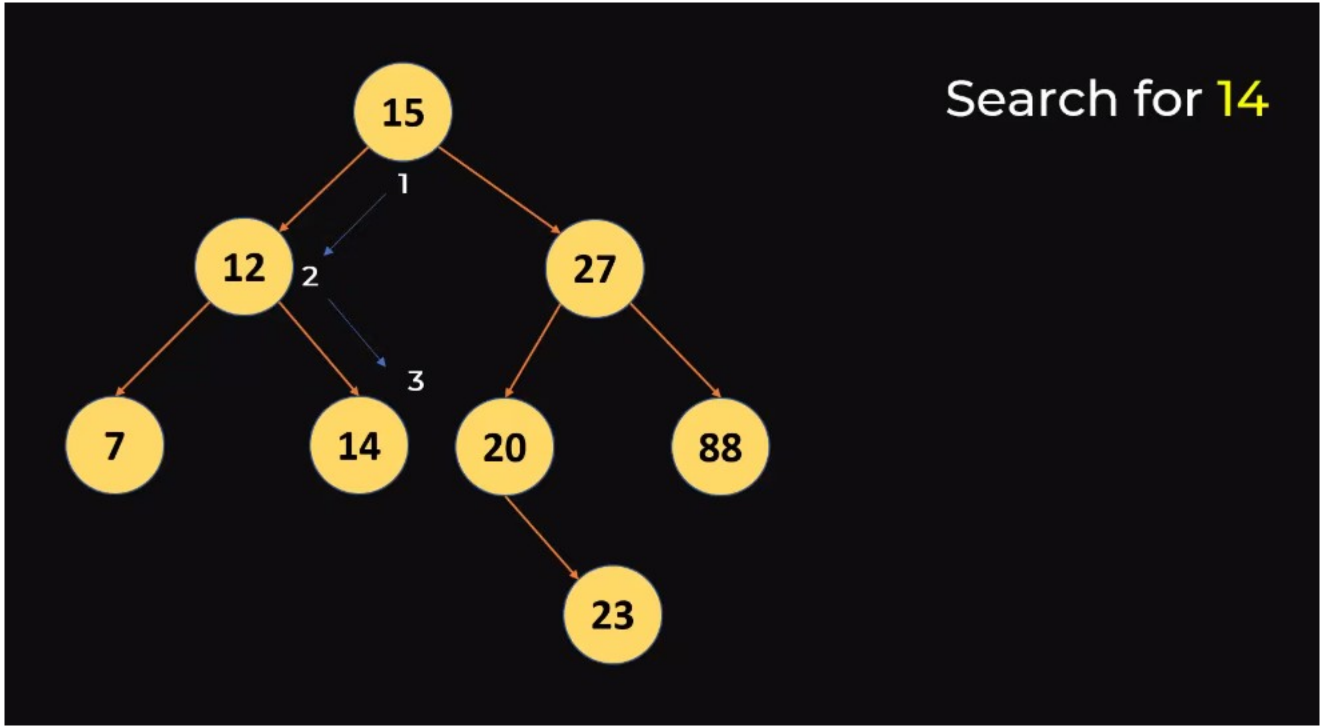
```
          12
        /     \
      10       20
    /   \     /
   9    11   12
       /       \
     10         12
```

Binary Search Tree (Cont.)

- **Searching into a Tree**



Binary Search Tree (Cont.)



Binary Search Tree (Cont.)

Search Complexity

- Every iteration we reduce search space by ½

- Total no. of nodes in tree i.e., N=8, then when searching we reduce by half so 8->4->2->1
- So, total 3 iterations
- Therefore, Log28 = 3

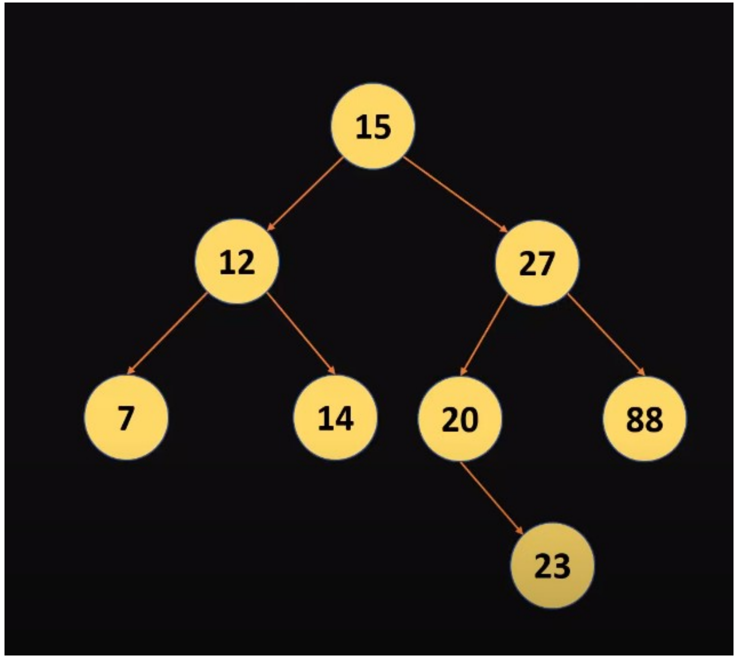
- Search Complexity = O(log n)

Traversal in Binary Tree

- Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree
- There are two approaches to search in Binary tree.
  - Breadth first Search
  - Depth first Search
- There are three ways which we use to traverse a tree in Depth first Search:
  - In-order Traversal
  - Pre-order Traversal
  - Post-order Traversal

Traversal in Binary Tree

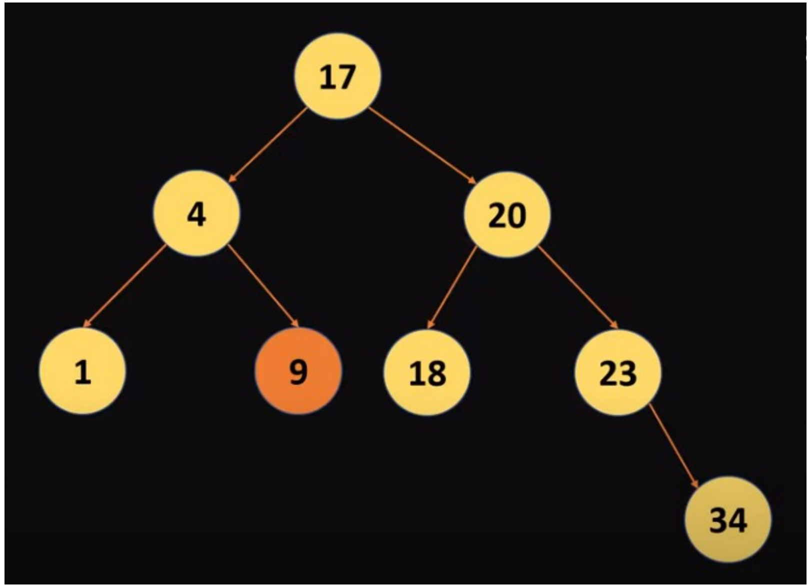
- Inorder Traversal:
  [7,12,14,15,20,23,27,88]
- Pre order Traversal:
  [15,12,7,14,27,20,23,88]
- Post order Traversal:
  [7,14,12,23,20,88,27,15]

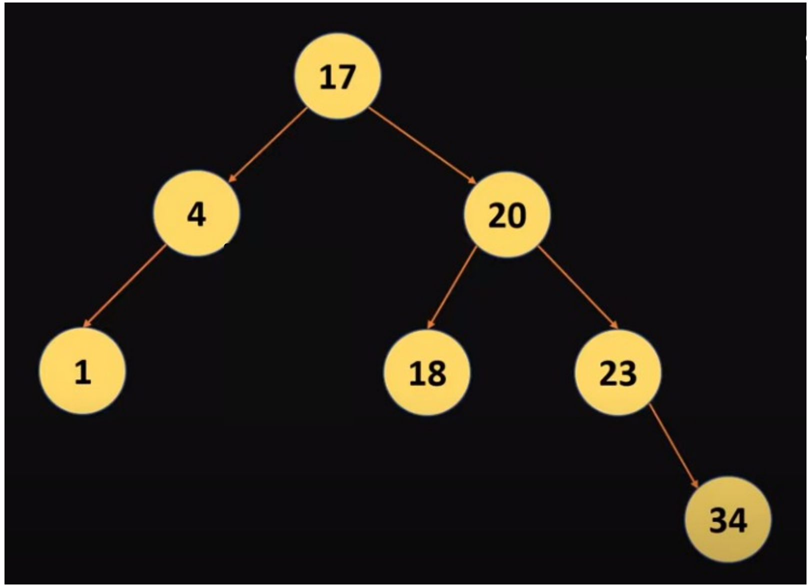


Binary Search Tree (Cont.)

Deleting a Node

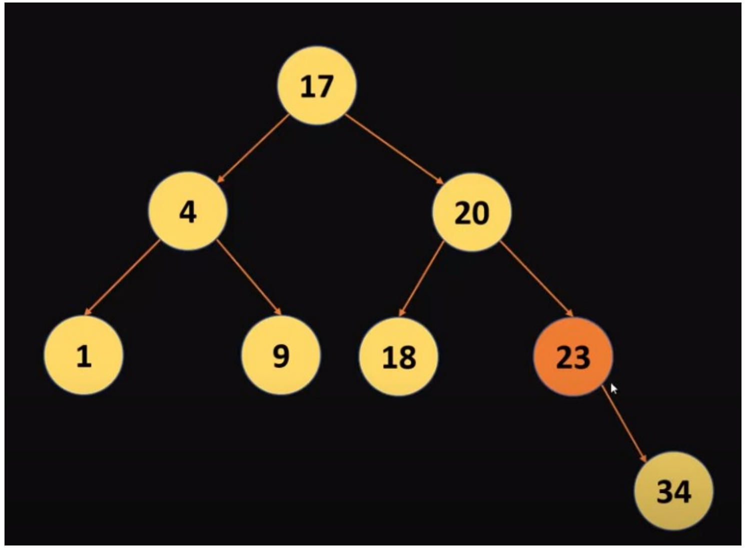
- Case1: Delete the node with no child.

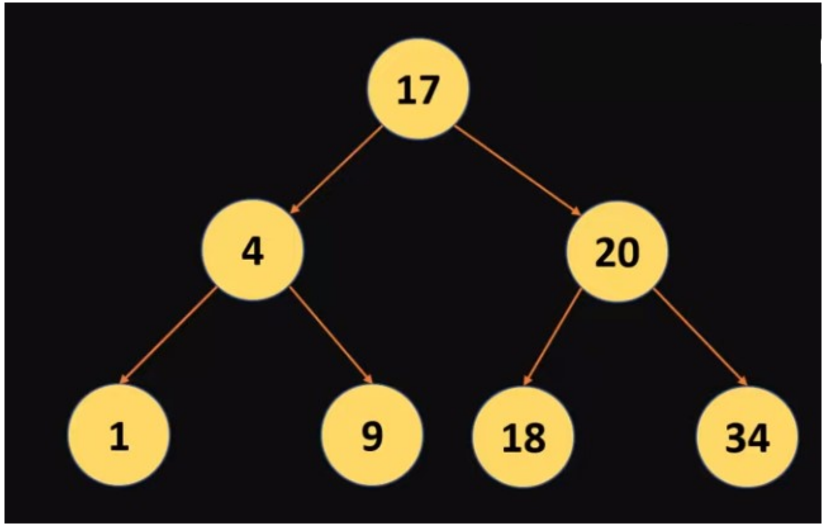




Binary Search Tree (Cont.)

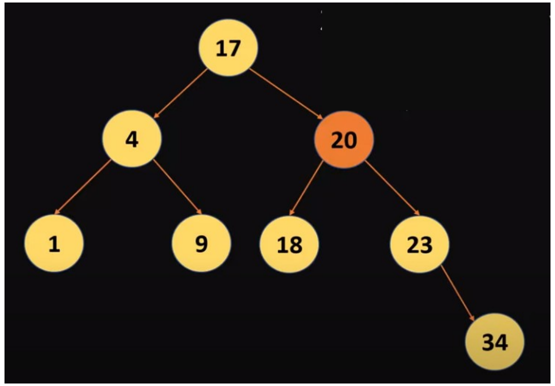
- Case2: Delete the node with one child.

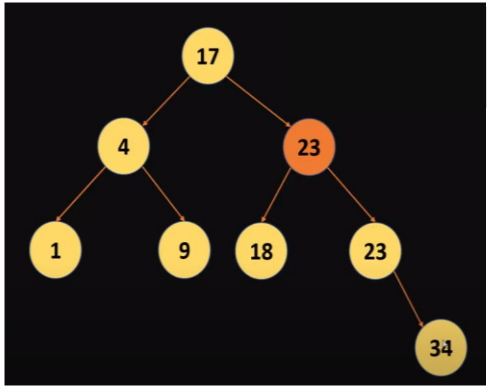


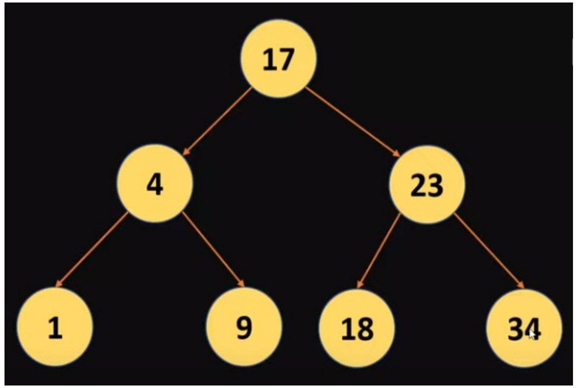


Binary Search Tree (Cont.)

- Case3: Delete the node with two child.
- Copy the minimum from the right subtree.
- Delete the duplicate node from the right subtree.

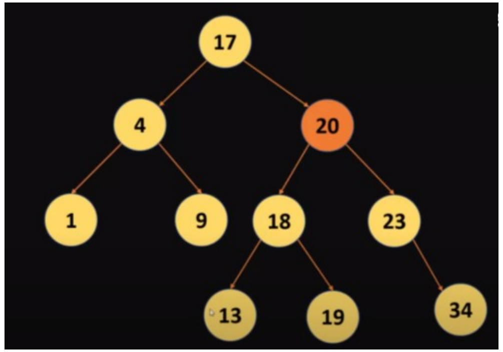


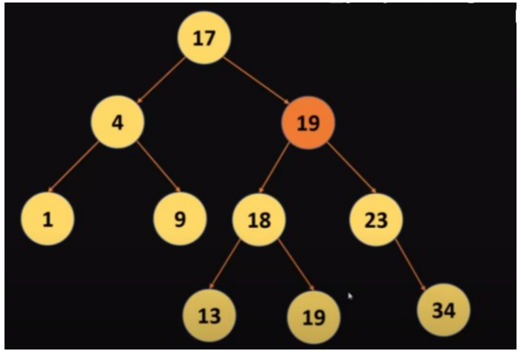


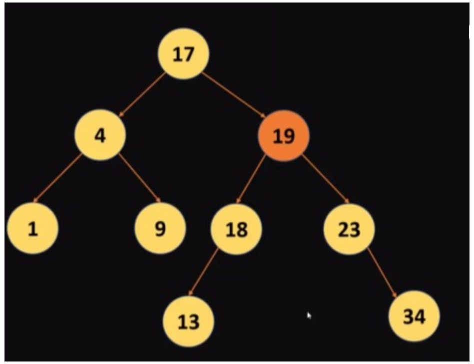


Binary Search Tree (Cont.)

- Alternate approach
- Case3: Delete the node with two child.
- Find the maximum from the left subtree.
- Delete the duplicate node from the left subtree.







Threaded Binary Tree

- A binary tree can be represented using array representation or linked list representation. When a binary tree is represented using linked list representation, the reference part of the node which doesn't have a child is filled with a NULL pointer. In any binary tree linked list representation, there is a number of NULL pointers than actual pointers. Generally, in any binary tree linked list representation, if there are **2N** number of reference fields, then **N+1** number of reference fields are filled with NULL (**N+1 are NULL out of 2N**). This NULL pointer does not play any role except indicating that there is no link (no child).
- A. J. Perlis and C. Thornton have proposed new binary tree called "***Threaded Binary Tree***", which makes use of NULL pointers to improve its traversal process. In a threaded binary tree, NULL pointers are replaced by references of other nodes in the tree. These extra references are called as ***threads***.
- **Threaded Binary Tree is also a binary tree in which all left child pointers that are NULL (in Linked list representation) points to its in-order predecessor, and all right child pointers that are NULL (in Linked list representation) points to its in-order successor.**
- If there is no in-order predecessor or in-order successor, then it points to the header node.

Threaded Binary Tree

Why do we need Threaded Binary Tree?

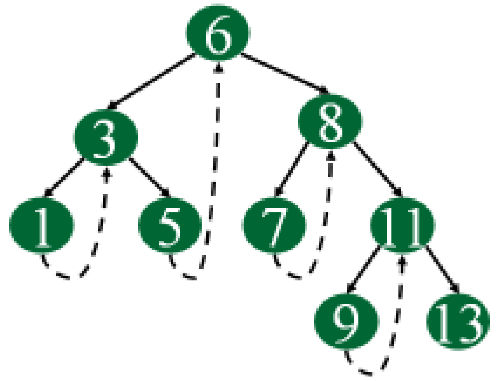
- Binary trees have a lot of wasted space: the leaf nodes each have 2 null pointers. We can use these pointers to help us in inorder traversals.
- Threaded binary tree makes the tree traversal faster since we do not need stack or recursion for traversal.

Types of threaded binary trees:

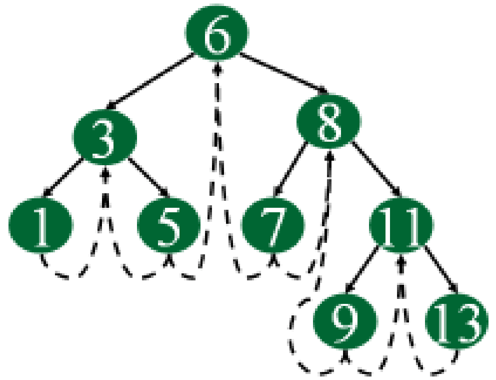
- **Single Threaded**: each node is threaded towards either the in-order predecessor or successor (left **or** right) means all right null pointers will point to inorder successor **OR** all left null pointers will point to inorder predecessor.
- **Double threaded:** each node is threaded towards both the in-order predecessor and successor (left **and** right) means all right null pointers will point to inorder successor **AND** all left null pointers will point to inorder predecessor.

Threaded Binary Tree

- Infix Traversal: [1,3,5,6,7.8.9,11,13]



Single Threaded Binary Tree



Double Threaded Binary Tree

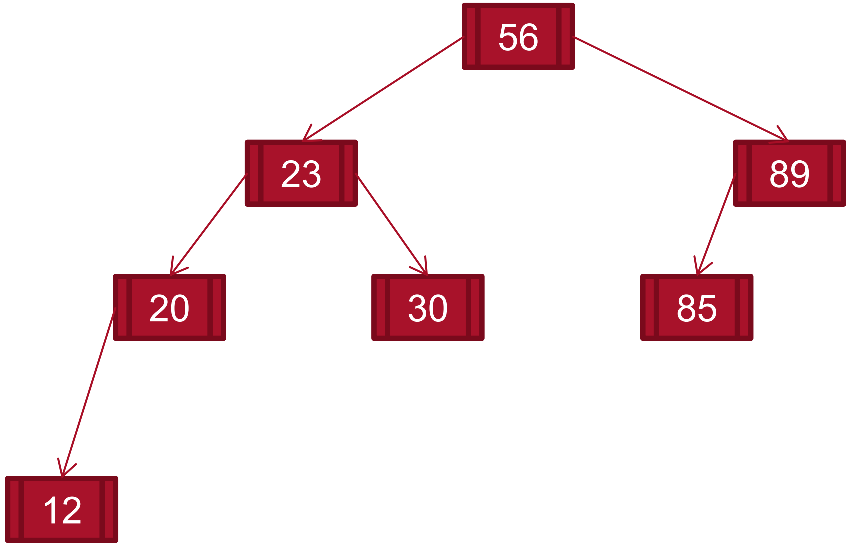
Threaded Binary Tree

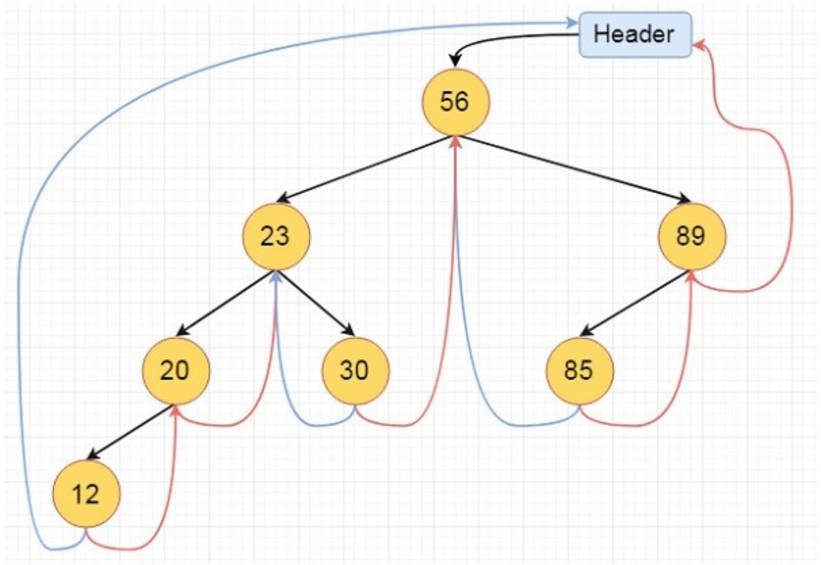
- For fully threaded binary tree, each node has five fields. Three fields like normal binary tree node, another two fields to store Boolean value to denote whether link of that side is actual link or thread.

Left Thread Flag	Left Link	Data	Right Link	Right Thread Flag

Threaded Binary Tree

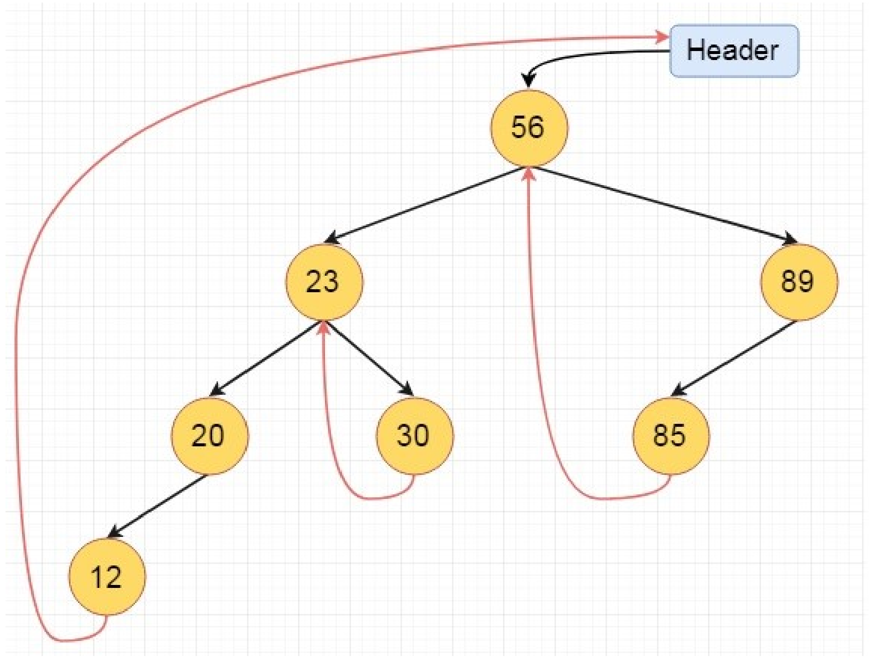
- Fully threaded binary tree
- Infix Traversal: [12,20,23,30,56,85,89]





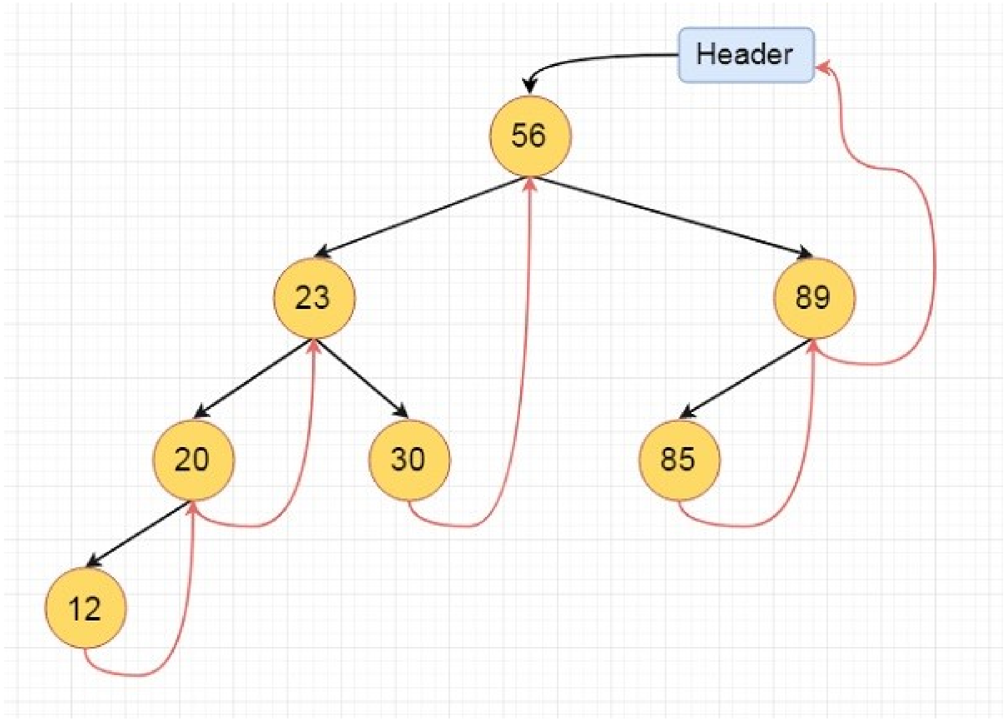
Threaded Binary Tree

- Left threaded tree



Threaded Binary Tree

- Right threaded tree



Thank You