CHAPTER 4.

CONTENT:

4.1 INTRODUCTION

- String, arrays, lists, stacks and Queues are the linear type of data structure.
- Tree is nonlinear data structure. This structure is mainly used to represent data containing a hierarchical relationship between records, family trees and tables of contents.
- First we discuss a special kind of tree, called a binary tree, which can be easily maintained in the computer but may seem to be very restrictive.

BINARY TREES:

- A binary tree *T* is defined as a finite set of elements, called nodes, such that:

T is empty (Called the **null tree** or **empty tree**),

- *T* contains a distinguished node ***R***, called the root of *T*, and the remaining nodes of *T* from an ordered pair of disjoint binary trees T_1, **and** T_2.
- If *T* does contain a root *R*, then the two trees *T1* and *T2* are called, respectively, the ***left and right*** *subtrees* of *R*. If *T1* is nonempty, then its root is called the ***left successor*** of *R*; similarly if *T2* is nonempty, then its root is called ***right successor*** of *R*.

- A binary tree *T* is frequently presented by means of a diagram. Following diagram represents a binary tree *T* as follows.
- *T* consist of 11 nodes, represented by the letters A through *L*, excluding *I*.
- The root of *T* is the node A at the top of the diagram.
- A left-downward slanted line from a node *N* indicates a left successor of *N*, and a right - downward slanted line from *N* indicates a right successor on *N*
- Observe that:
- B is left successor and C is a right successor of the node A.
- The left subtree of the root A consist of the nodes B, D, E and F and the right subtree of A consist of the nodes C, G, H, J, K and L.
- Any node N in a binary tree T has either 0, 1 or 2 successors. The nodes A, B, C and H have two successors, the nodes E and J have only one successor, and the nodes *D, F, G, L* and *K* have no successors. The nodes with no successors are called *terminal nodes*.
-

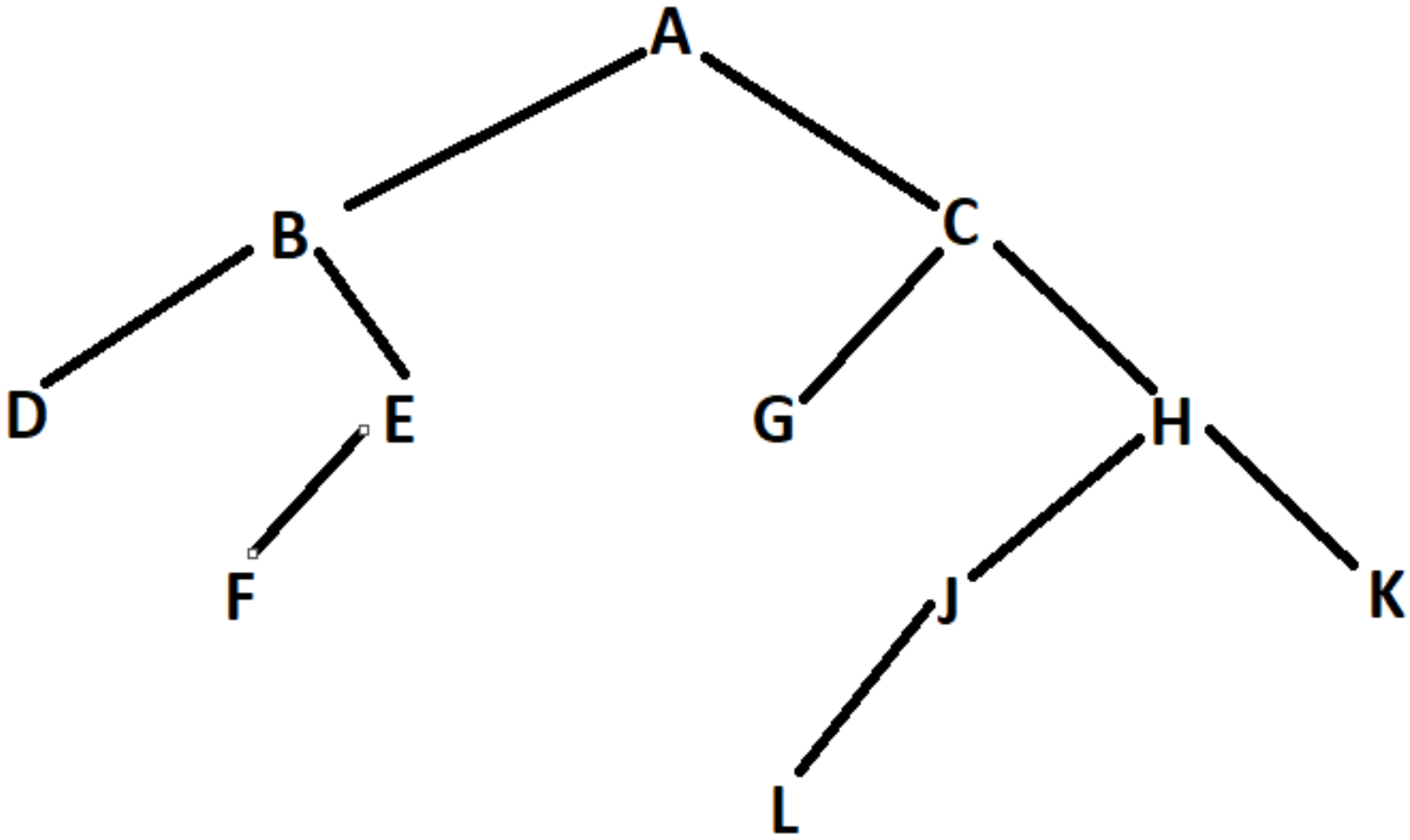


Fig . binary tree structure

- The above definition of the binary tree T is recursive since T is defined in terms of the binary subtrees T_1 and T_2. This means, in particular, that every node N of T contains a left and a right subtree. Moreover, if N is a terminal node, then both its left and right subtrees are empty.
- Binary trees T and T' are said to be similar if they have the same structure or, in other words, if they have the same shape. The trees are said to be *copies* if they are similar and if they have the same contents at corresponding nodes.

4.2 BINARY TREE & IT'S MEMORY REPRESENTATION

- There are two ways of representing T in memory.
- A) Linked Representation of binary tree
- B) Sequential Representation of binary tree

A) LINKED REPRESENTATION OF BINARY TREE

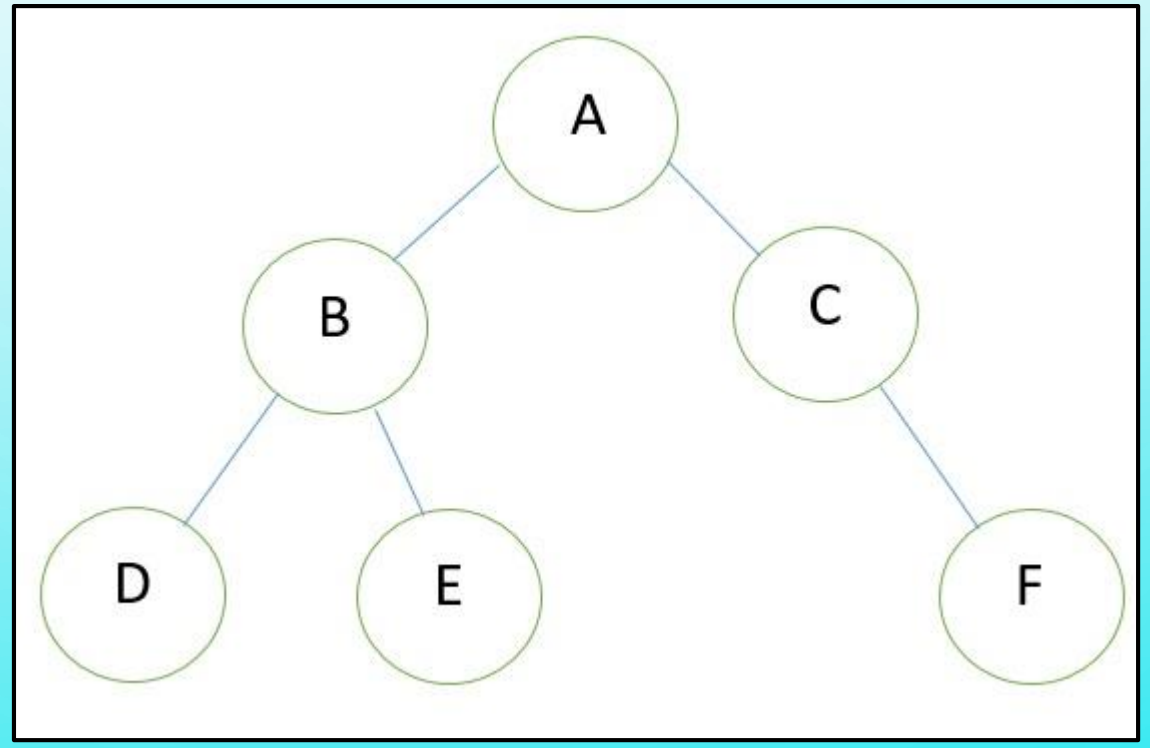
- Consider a binary tree T. T will be maintained in memory by means of a linked representation which uses three parallel arrays, INFO, LEFT and RIGHT and a pointer ROOT as follows. In Binary Tree each node **N** of **T** will correspond to a location **k** such that
- **LEFT** [k] contains the location of the left child of node **N.**
- **INFO** [k] contains the data at the node **N.**
- **RIGHT** [k] contains the location of right child of node **N.**
- **Representation of a node:**

REPRESENTATION OF A NODE:

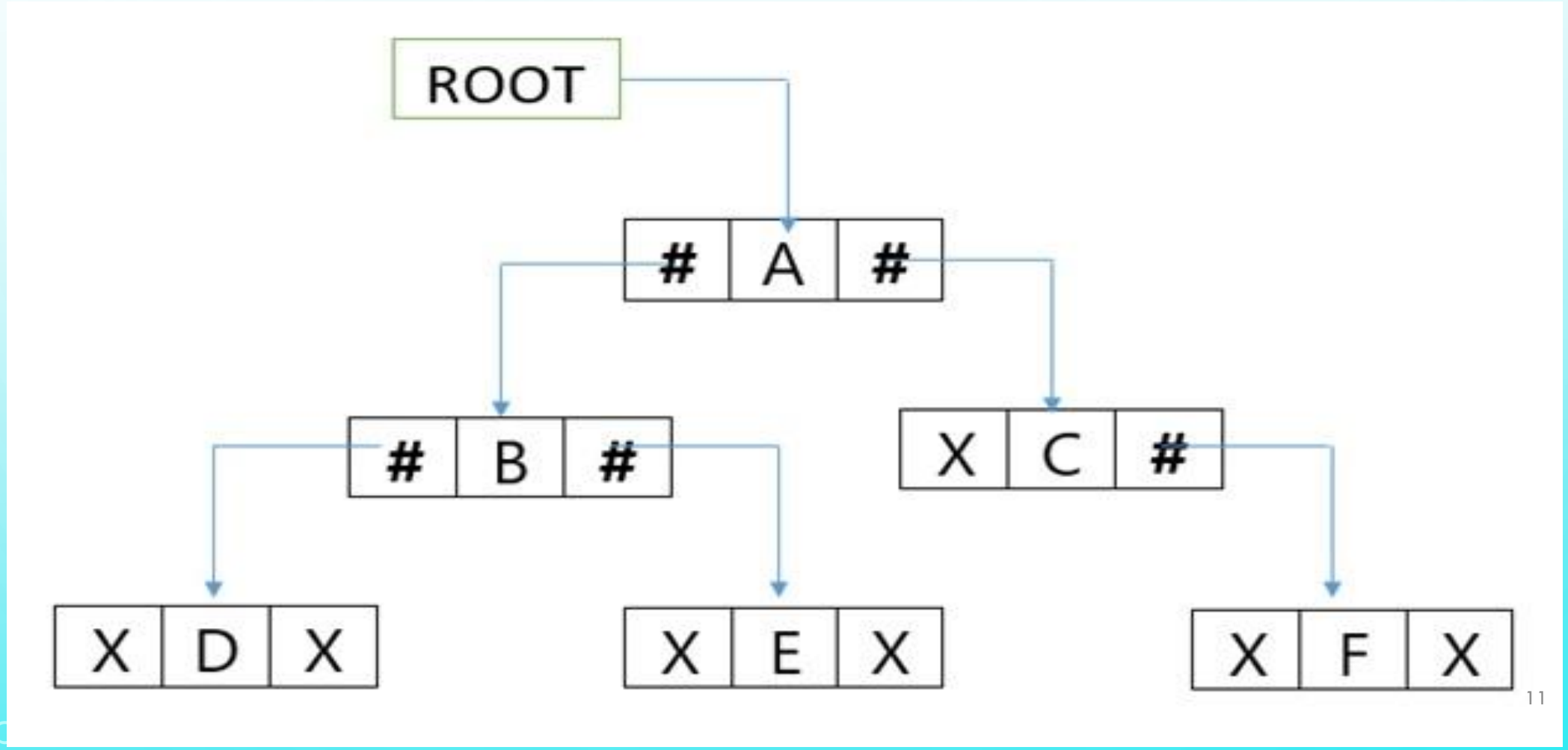
- In this representation of binary tree root will contain the location of the root **R** of **T**. If any one of the
- subtree is empty, then the corresponding pointer will contain the null value if the tree **T** itself is empty, the **ROOT** will contain the null value.

EXAMPLE

- Consider the binary tree **T** in the figure. A schematic diagram of the linked list representation of **T**appears in the following figure. Observe that each node is pictured with its three fields, and that the empty subtree is pictured by using x for null entries.



LINKED REPRESENTATION OF THE BINARY TREE



REPRESENTATION OF BINARY TREE IN LINKED LIST IS AS FOLLOWS

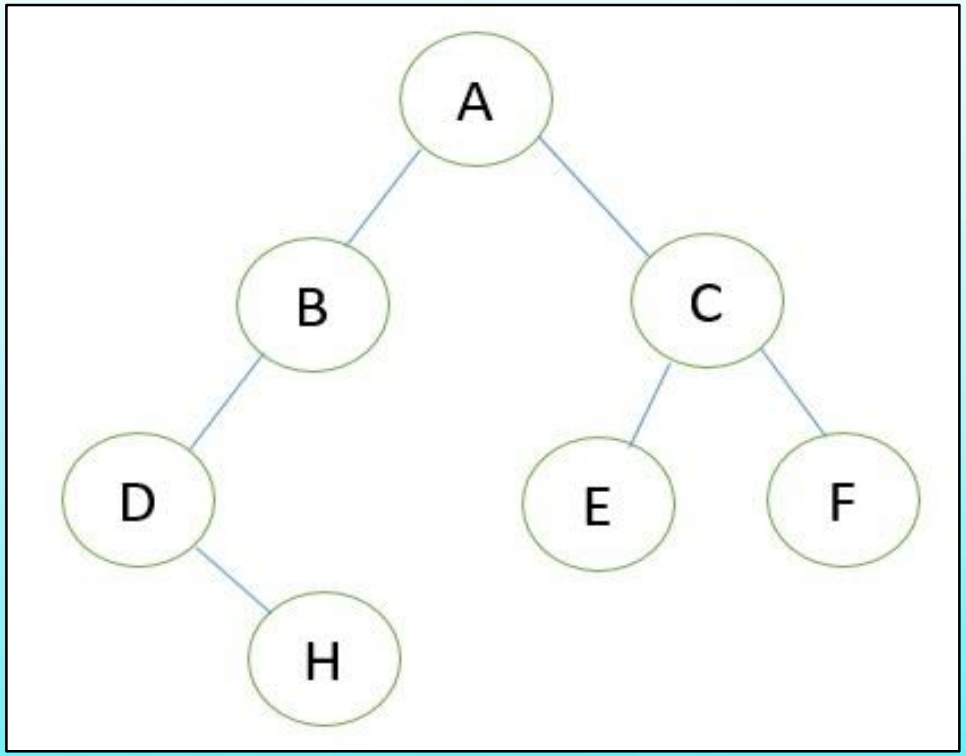
ROOT
5

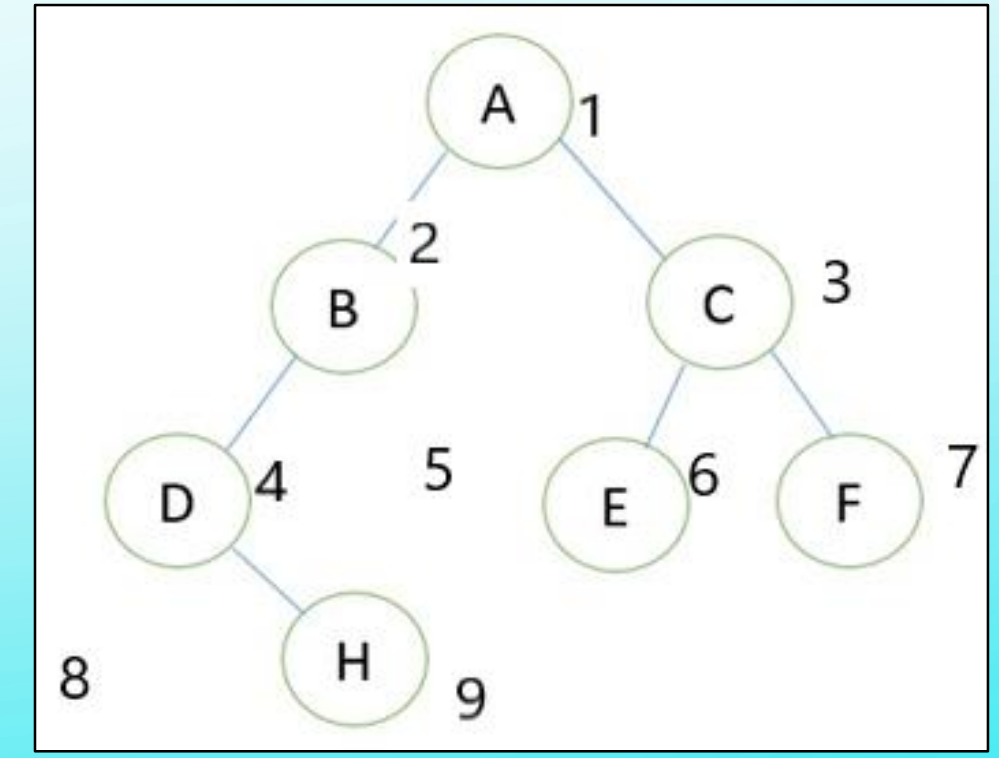
	INFO	LEFT	RIGHT
1			
2	C	0	7
3			
4	E	0	0
5	A	10	2
6			
7	F	0	0
8	D	0	0
9			
10	B	8	4

B) SEQUENTIAL REPRESENTATION OF BINARY TREE

- Let us consider that we have a tree **T**. let our tree **T** is a binary tree that us complete binary tree. Then there is an efficient way of representing **T** in the memory called the sequential representation or array representation of **T**. This representation uses only a linear array TREE as follows:
- The root **N** of **T** is stored in **TREE [1]**.
- If a node occupies **TREE [k]** then its left child is stored in **TREE [2 * k]** and its right child is stored into **TREE [2 * k + 1]**.

FOR EXAMPLE:
CONSIDER THE FOLLOWING TREE:





ITS SEQUENTIAL REPRESENTATION IS AS FOLLOW:

Its sequential representation is as follow:

A	B	C	D	-	E	F	-	H		

A	B	C	D		E	F		H		
1	2	3	4	5	6	7	8	9		

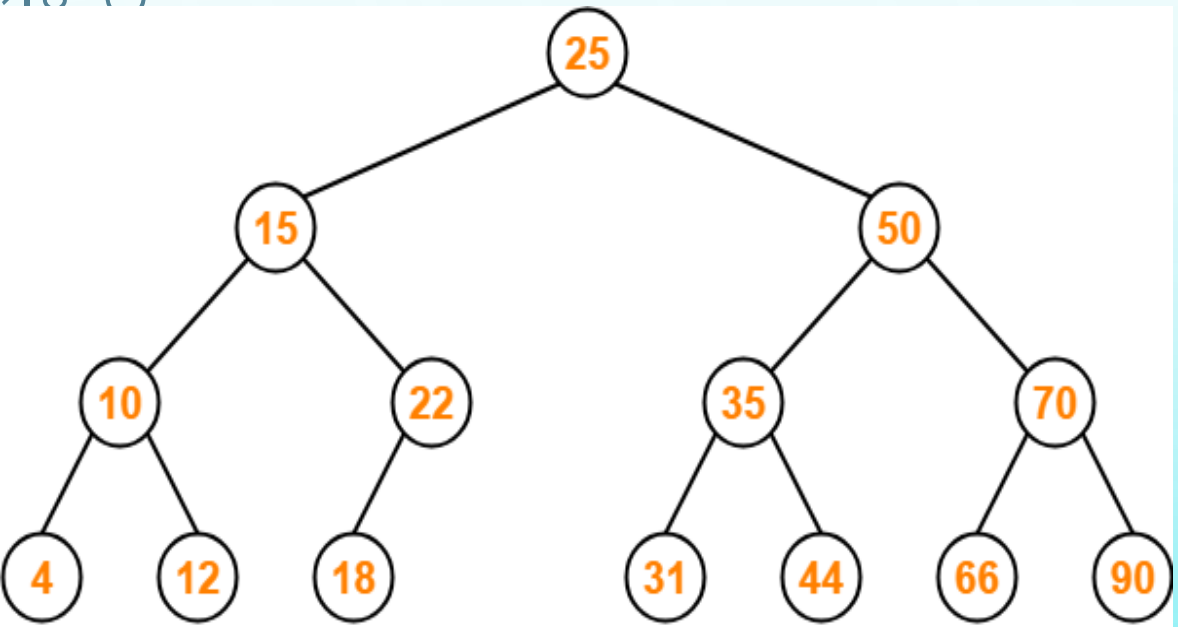
BINARY SEARCH TREE OPERATIONS-

1. SEARCH OPERATION-

- **Rules-**
- For searching a given key in the BST,
- Compare the key with the value of root node.
- If the key is present at the root node, then return the root node.
- If the key is greater than the root node value, then recur for the root node's right subtree.
- If the key is smaller than the root node value, then recur for the root node's left subtree.

EXAMPLE-

CONSIDER KEY = 45 HAS TO BE SEARCHED IN THE GIVEN BST-



Binary Search Tree

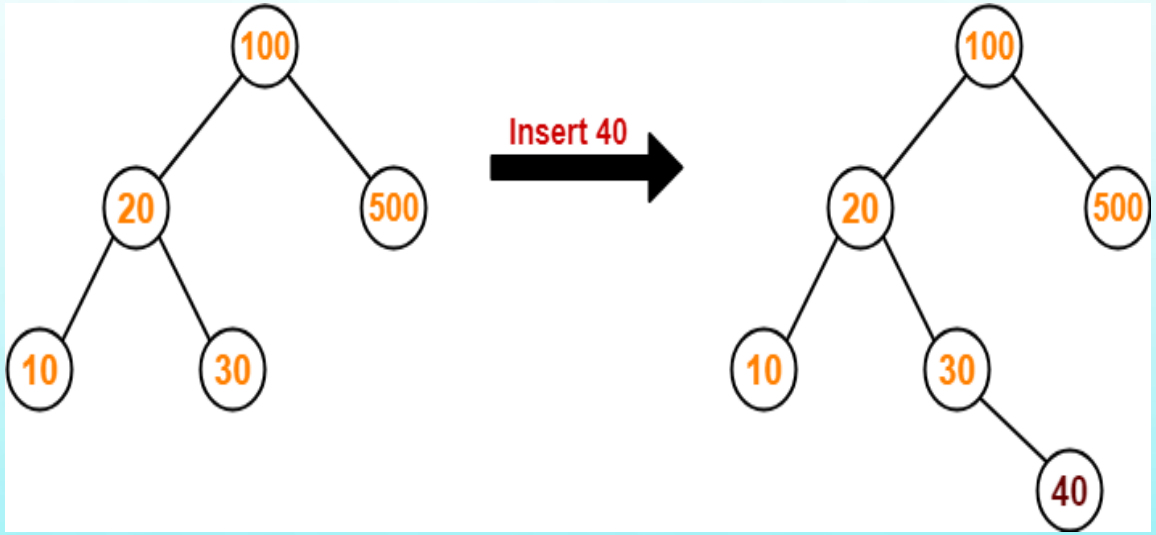
- We start our search from the root node 25.
- As 45 > 25, so we search in 25's right subtree.
- As 45 < 50, so we search in 50's left subtree.
- As 45 > 35, so we search in 35's right subtree.
- As 45 > 44, so we search in 44's right subtree but 44 has no subtrees.
- So, we conclude that 45 is not present in the above BST.

INSERTION OPERATION

- **Rules-**
- The insertion of a new key always takes place as the child of some leaf node.
- For finding out the suitable leaf node,
- Search the key to be inserted from the root node till some leaf node is reached.
- Once a leaf node is reached, insert the key as child of that leaf node.

EXAMPLE-

CONSIDER THE FOLLOWING EXAMPLE WHERE KEY = 40 IS INSERTED IN THE GIVEN BST-

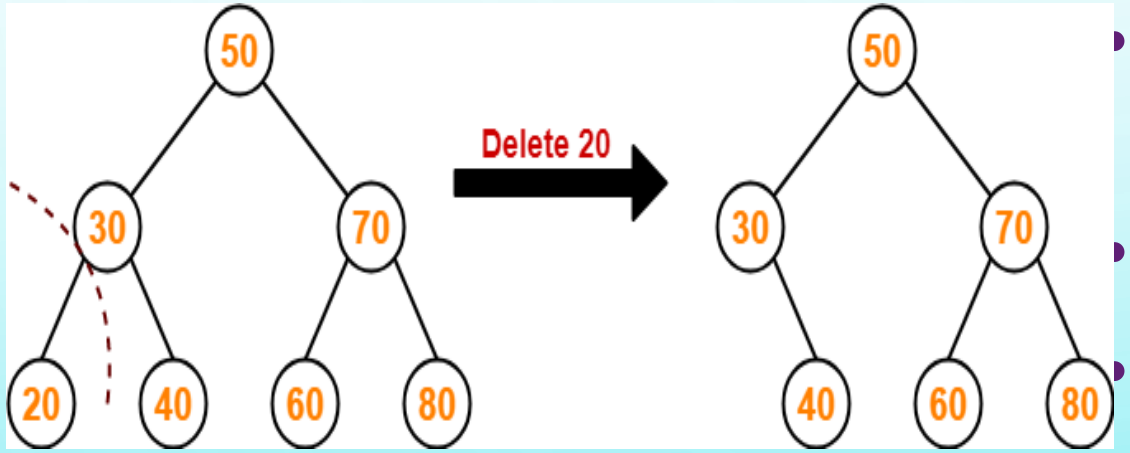


- We start searching for value 40 from the root node 100.
- As 40 < 100, so we search in 100's left subtree.
- As 40 > 20, so we search in 20's right subtree.
- As 40 > 30, so we add 40 to 30's right subtree.

3. DELETION OPERATION-

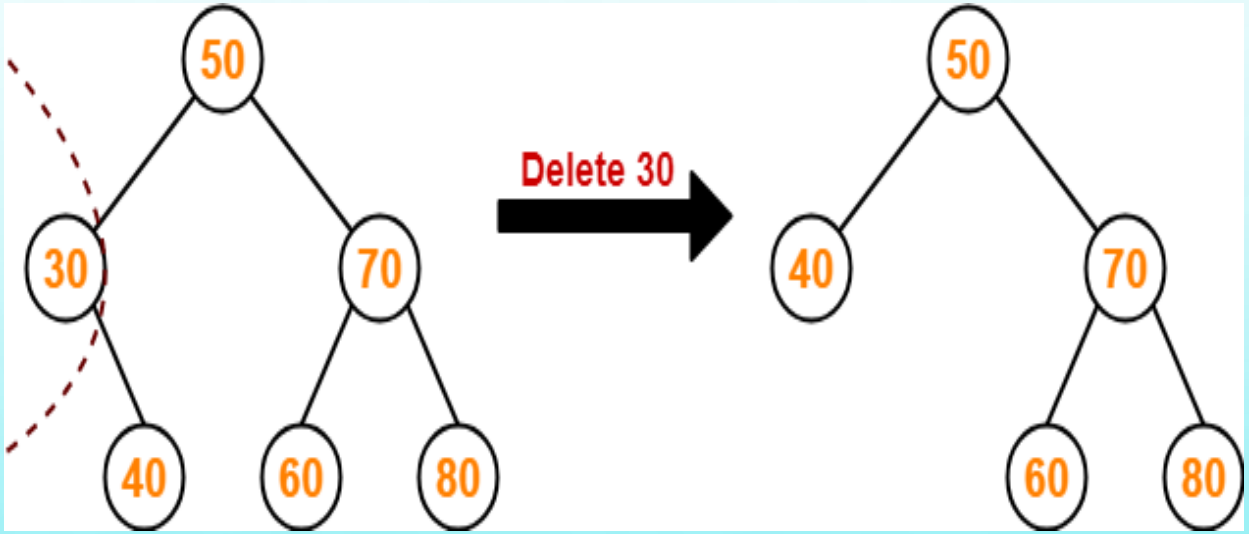
- Deletion Operation is performed to delete a particular element from the Binary Search Tree
- Case-01: Deletion Of A Node Having No Child (Leaf Node)-
- Case-02: Deletion Of A Node Having Only One Child-
- Case-03: Deletion Of A Node Having two Child-

CASE-01: DELETION OF A NODE HAVING NO CHILD (LEAF NODE)-



- Just remove / disconnect the leaf node that is to deleted from the tree.
- Example-
- Consider the following example where node with value = 20 is deleted from the BST-

CASE-02: DELETION OF A NODE HAVING ONLY ONE CHILD-

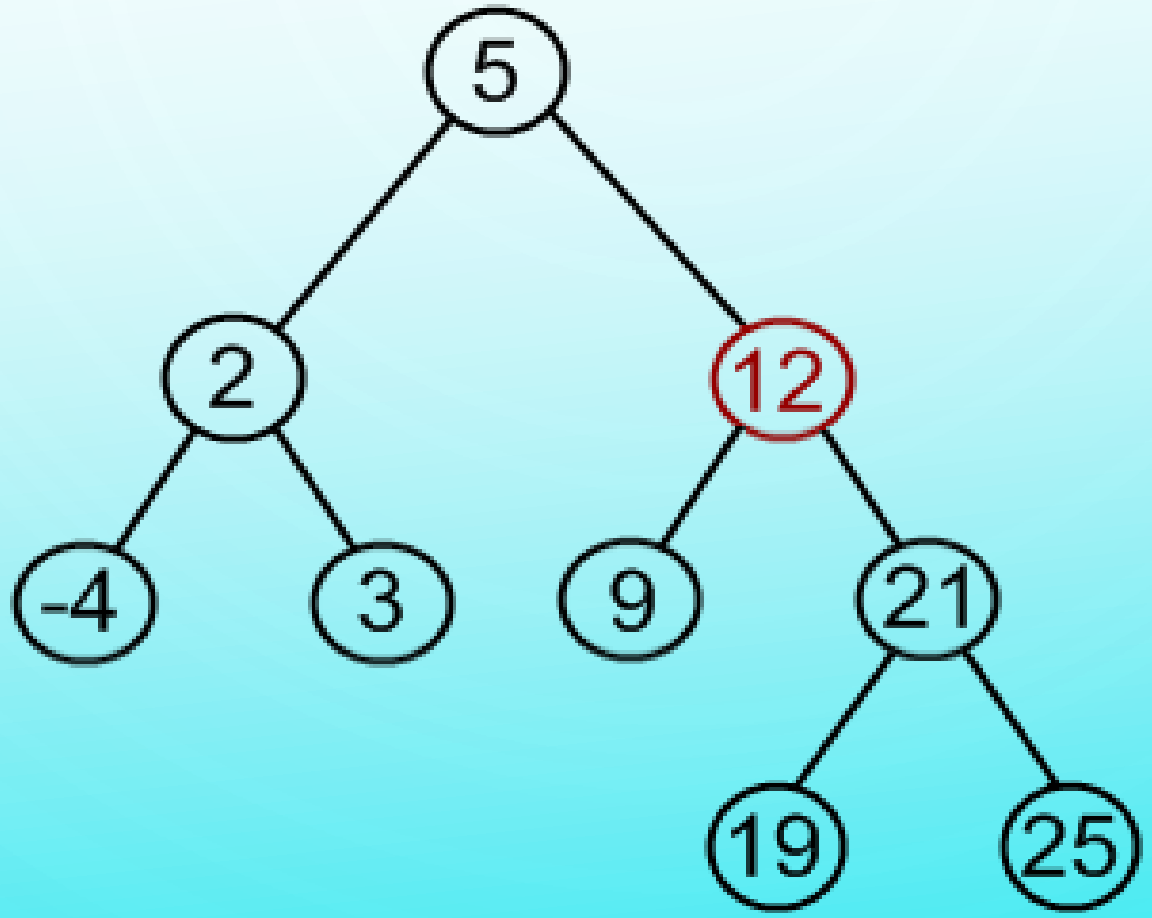


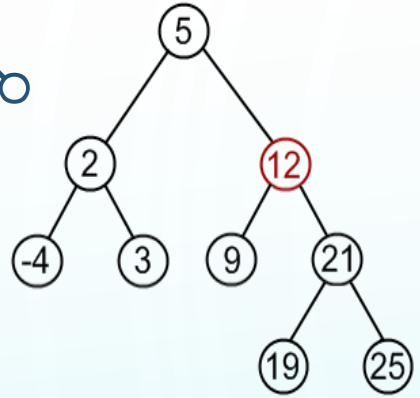
- Just make the child of the deleting node, the child of its grandparent.
- Example-
- Consider the following example where node with value = 30 is deleted from the BST-

CASE-03: DELETION OF A NODE HAVING TWO CHILD-

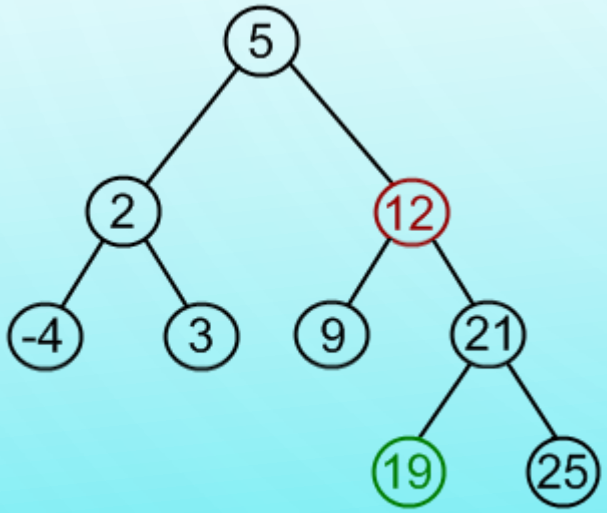
- Just make the child of the deleting node, the child of its grandparent.
- Example-
- This is the most complex case. To solve it, let us see one useful BST property first. We are going to use the idea, that the same set of values may be represented as different binary-search trees. For example those BSTs:

EXAMPLE. REMOVE 12 FROM A BST.

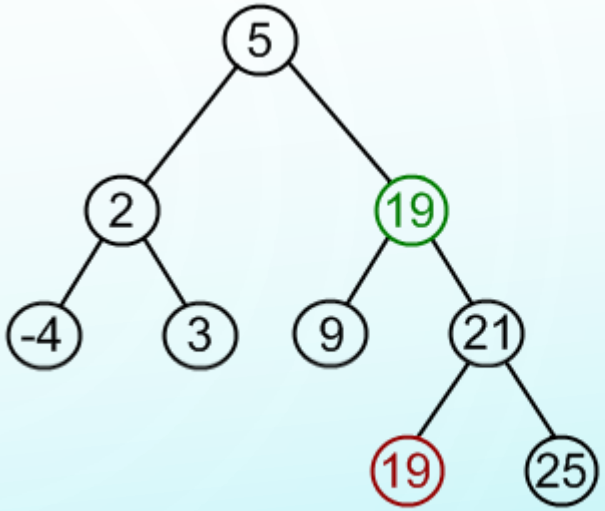




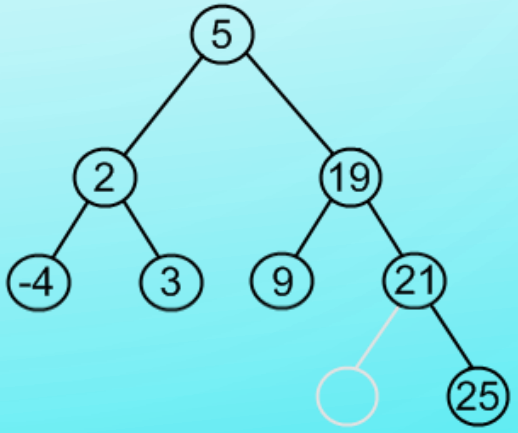
Find minimum element in the right subtree of the node to be removed. In current example it is 19.



Replace 12 with 19. Notice, that only values are replaced, not nodes. Now we have two nodes with the same value.



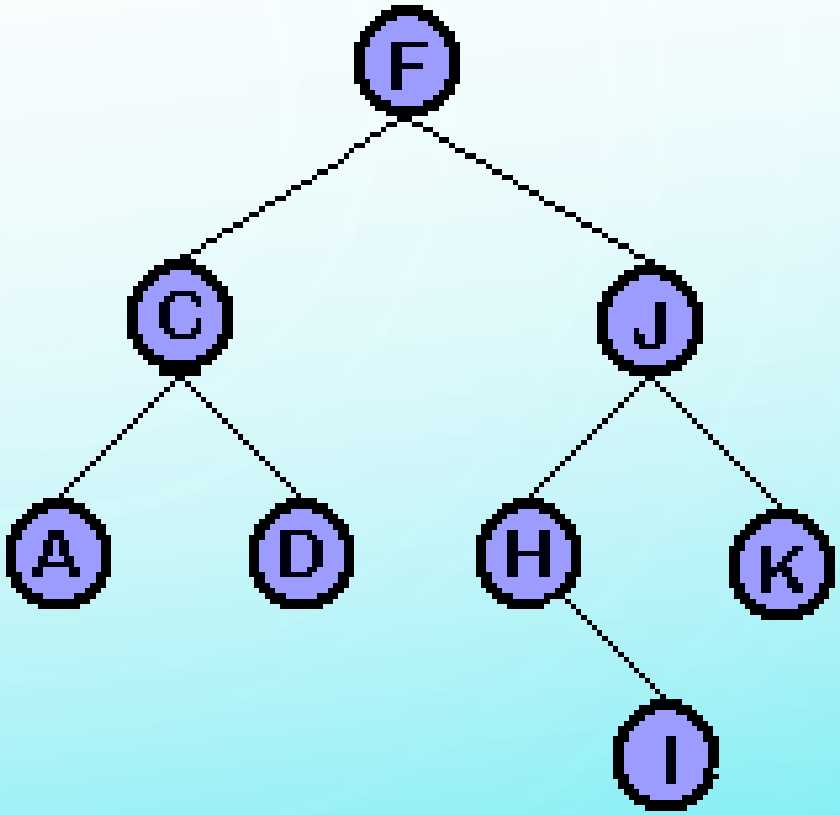
Remove 19 from the left subtree.



PREORDER

- A pre order traversal prints the contents of a sorted tree, in pre order. In other words, the contents of the root node are printed first, followed by left subtree and finally the right subtree. So in Figure a pre order traversal would result in the following string: FCADJHIK

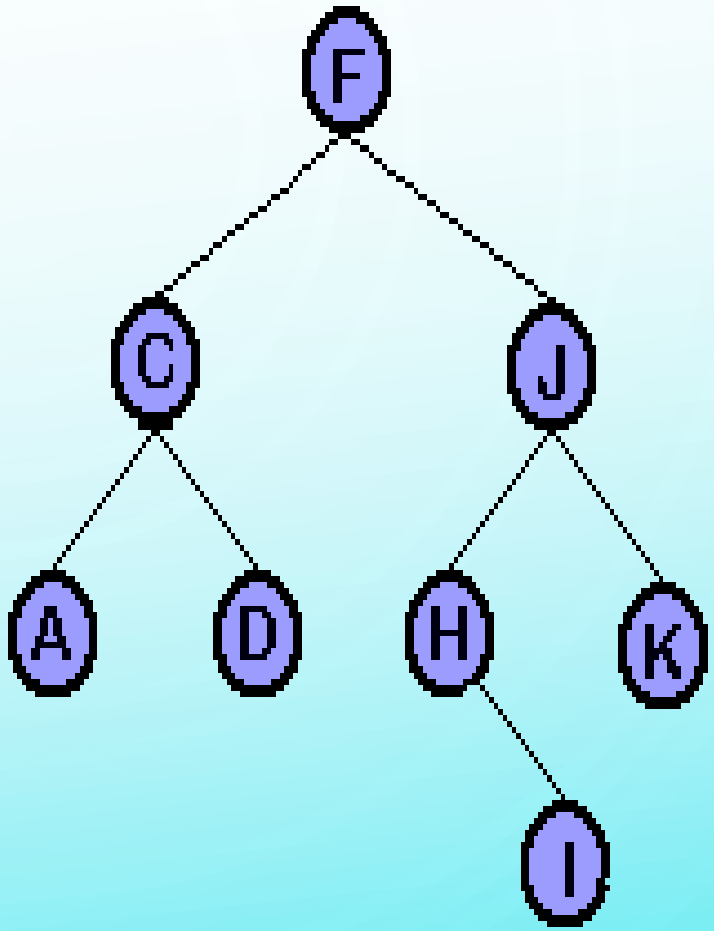
```
PreOrder (T)
    If T < > Null
    then print (T.data)
    else print('empty tree')
    If T.lp < > null
    then PreOrder(T.lp)
    If T.rp < > null
    then preorder (T.rp)
    end.
```



INORDER

- An in order traversal prints the contents of a sorted tree, in order. In other words, the lowest in value first, and then increasing in value as it traverses the tree. The order of a traversal would be 'a' to 'z' if the tree uses strings or characters, and would be increasing numerically from 0 if the tree contains numerical values. So in Figure 1.1, an in order traversal would result in the following string: ACDFHIJK.

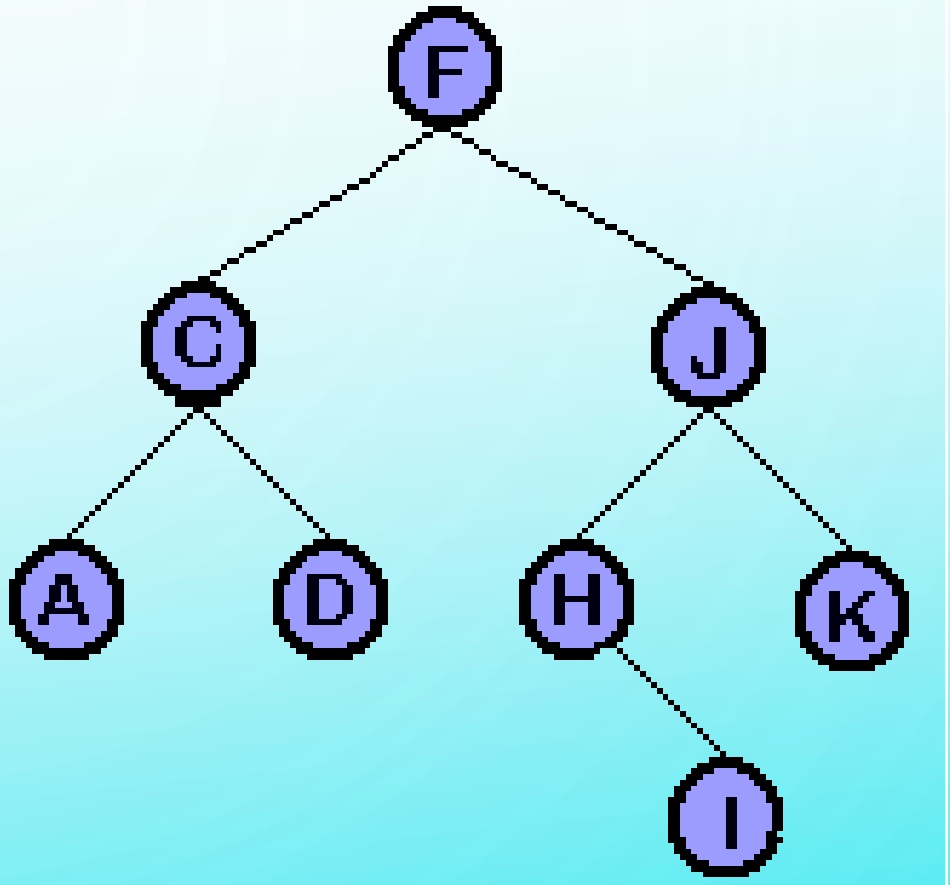
```
InOrder (T)
    If T < > null
    print ('empty tree')
    If T.lp < > null
    then InOrder(T.lp)
    print (T.data)
    If T.rp < > null
    then InOrder (T.lp)
    end.
```



POSTORDER

- A post order traversal prints the contents of a sorted tree, in post order. In other words, the contents of the left subtree are printed first, followed by right subtree and finally the root node. So in Figure 1.1, a post order traversal would result in the following string: ADCIHKJF.

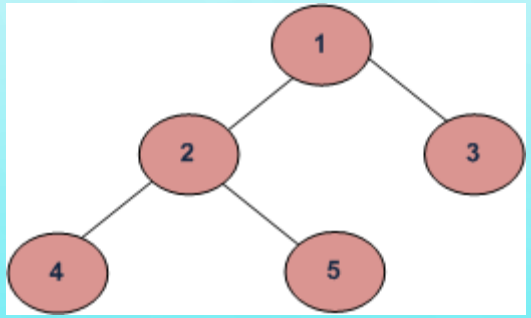
```
PostOrder (T)
    If T = null
    then print ('empty tree')
    If T.lp < > null
    then PostOrder(T.lp)
    If T.rp < > null
    then PostOrder(T.lp)
    Print(T.data)
    end.
```



4.4 THREADED BINARY TREE.

TREE TRAVERSALS (INORDER)

- Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.



- (a) Inorder (Left, Root, Right) : 4 2 5 1 3

Inorder Traversal (Practice):

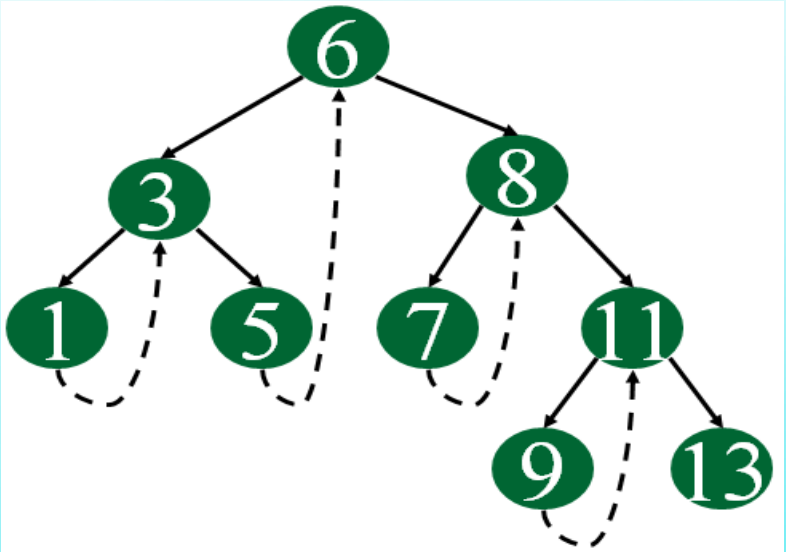
```
Algorithm Inorder(tree)
1. Traverse the left subtree, i.e.,
   call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e.,
   call Inorder(right-subtree)
```

Uses of Inorder

In case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST in non-increasing order, a variation of Inorder traversal where Inorder traversal s reversed can be used.

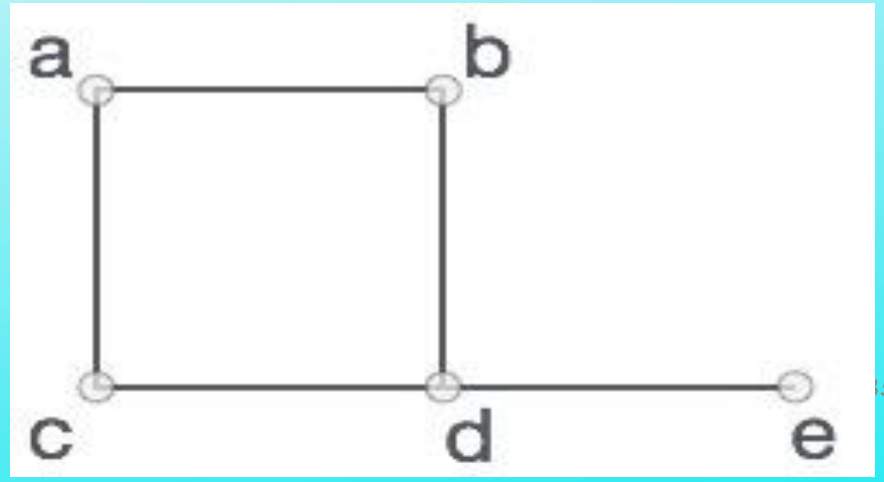
Example: Inorder traversal for the above-given figure is 4 2 5 1 3.

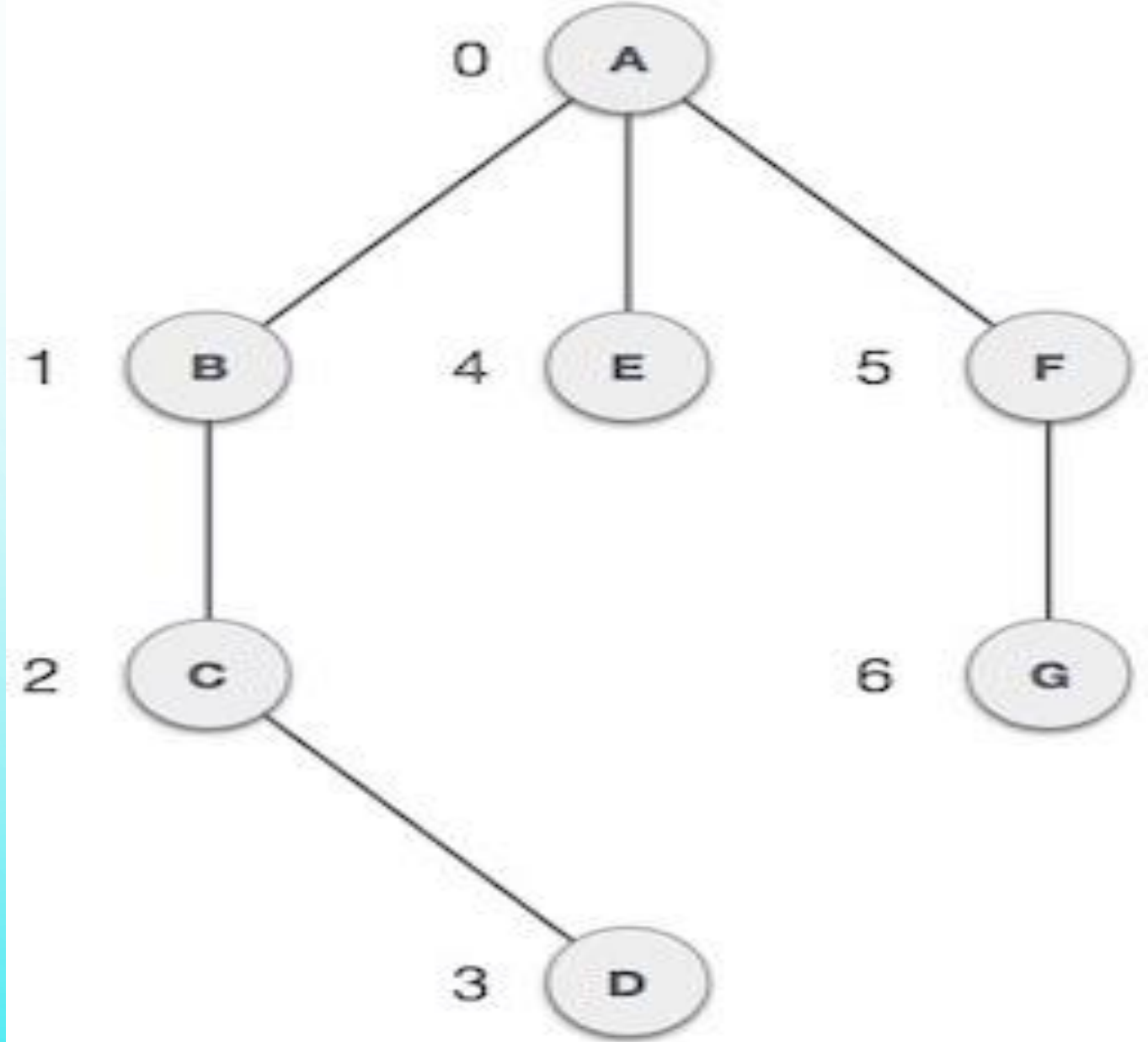
- The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).
- There are two types of threaded binary trees.
  Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)
- ***Double Threaded:*** Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.
- The threads are also useful for fast accessing ancestors of a node.
- Following diagram shows an example Single Threaded Binary Tree. The dotted lines represent threads.



WHAT IS GRAPH

- A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as **vertices**, and the links that connect the vertices are called **edges**.
- Formally, a graph is a pair of sets **(V, E)**, where **V** is the set of vertices and **E**is the set of edges, connecting the pairs of vertices. Take a look at the following graph –
- In the givan graph,
- V = {a, b, c, d, e}
- E = {ab, ac, bd, cd, de}

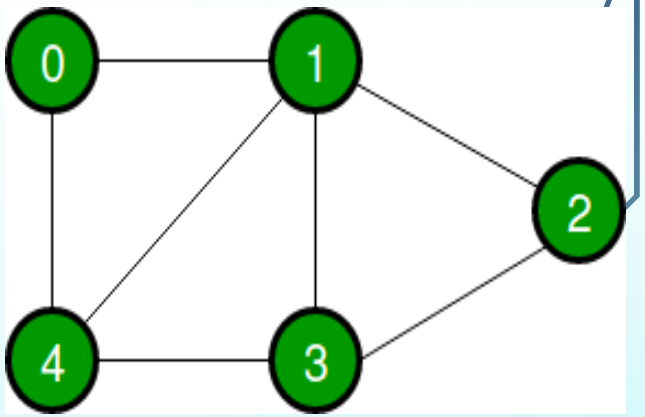



0
A
1
B
4
E
5
F
2
C
6
G
3
D

GRAPH DATA STRUCTURE

- Mathematical graphs can be represented in data structure. We can represent a graph using an array of vertices and a two-dimensional array of edges. Before we proceed further, let's familiarize ourselves with some important terms −
- **Vertex** − Each node of the graph is represented as a vertex. In the following example, the labeled circle represents vertices. Thus, A to G are vertices. We can represent them using an array as shown in the following image. Here A can be identified by index 0. B can be identified using index 1 and so on.

- **Edge** – Edge represents a path between two vertices or a line between two vertices. In the following example, the lines from A to B, B to C, and so on represents edges. We can use a two-dimensional array to represent an array as shown in the following image. Here AB can be represented as 1 at row 0, column 1, BC as 1 at row 1, column 2 and so on, keeping other combinations as 0.
- **Adjacency** – Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on.
- **Path** – Path represents a sequence of edges between the two vertices. In the following example, ABCD represents a path from A to D.



- Following two are the most commonly used representations of a graph.

1. Adjacency Matrix
2. Adjacency List

There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use.

- **Adjacency Matrix:**

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix for the above example graph is

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0