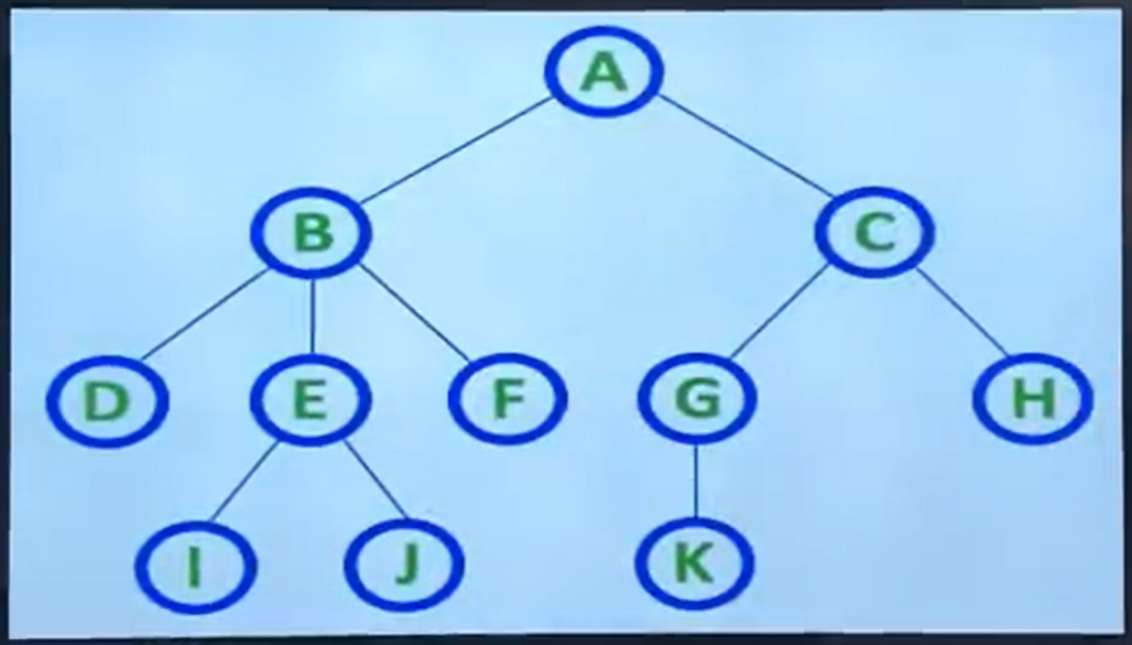
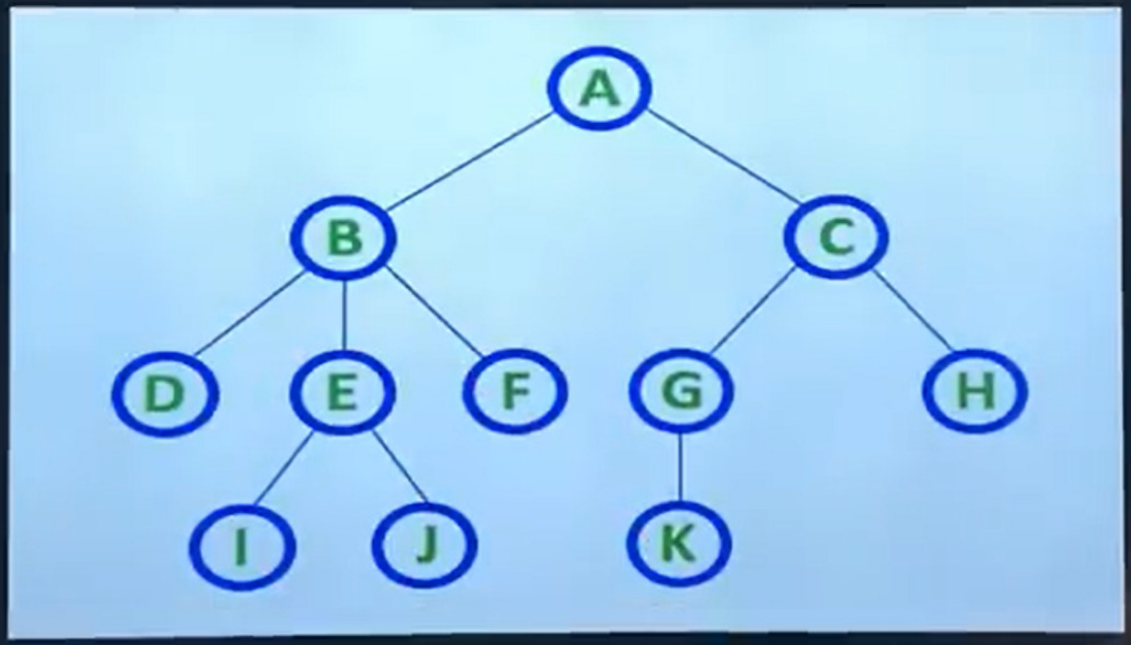
Tree

- The tree is one of the most powerful, flexible, versatile and nonlinear advanced data structures, it represents hierarchical relationship existing between several data items. it is used in wide range of applications.

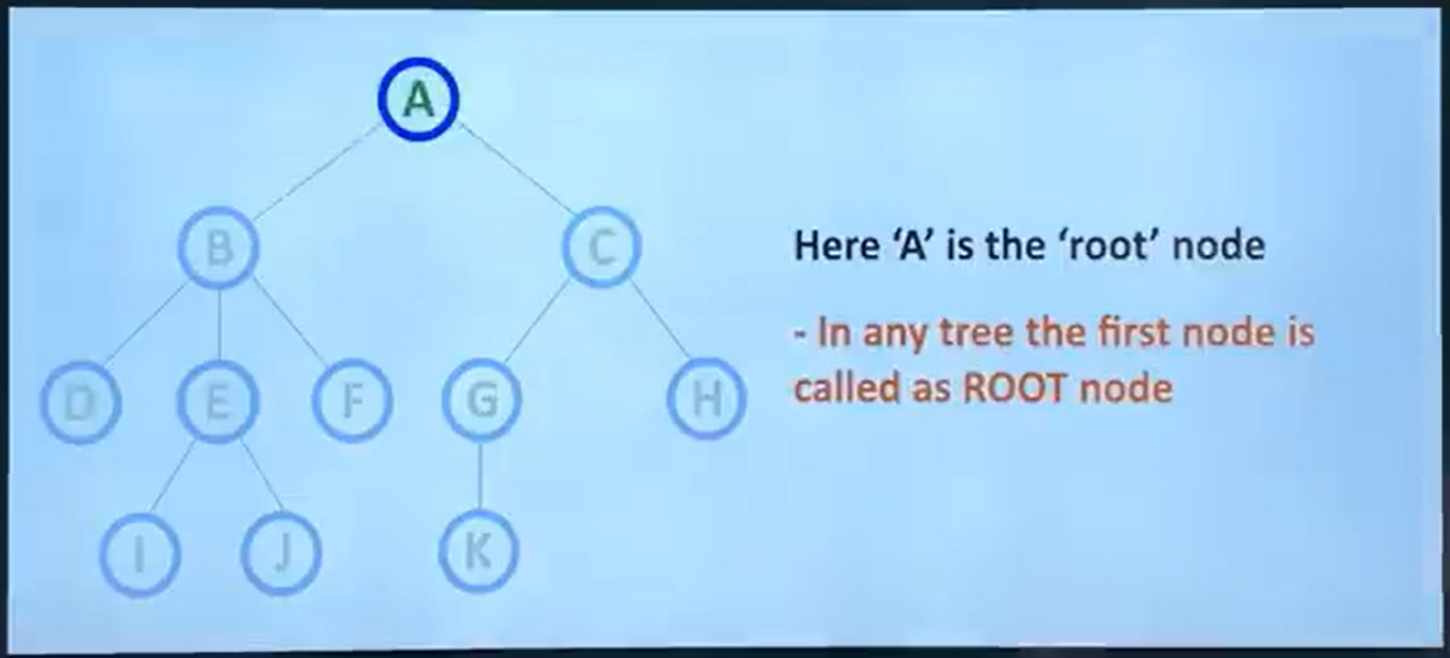


- A tree is a finite set of one or more data items(nodes) such that
  - There is a special data item called root of the tree
  - And its remaining data items are partitioned into number of mutually exclusive (disjoint) subsets, each of which is itself a tree and they are called subtree. i.e. Every node (exclude a root) is connected by a directed edge *from* exactly one other node; A direction is: *parent -> children*



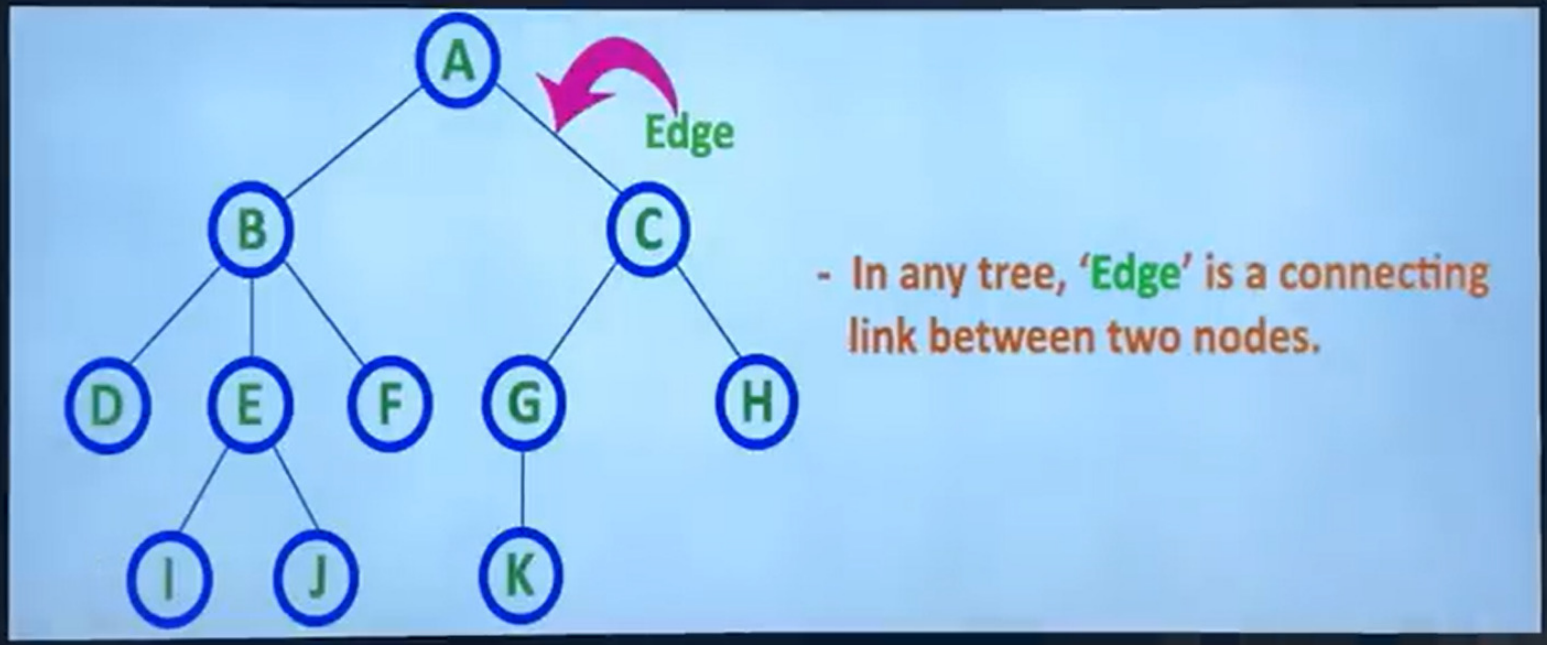
Root

- The first/Top most node is called as Root Node. We always have exactly one root node in every tree. We can say that root node is the origin of tree data structure.



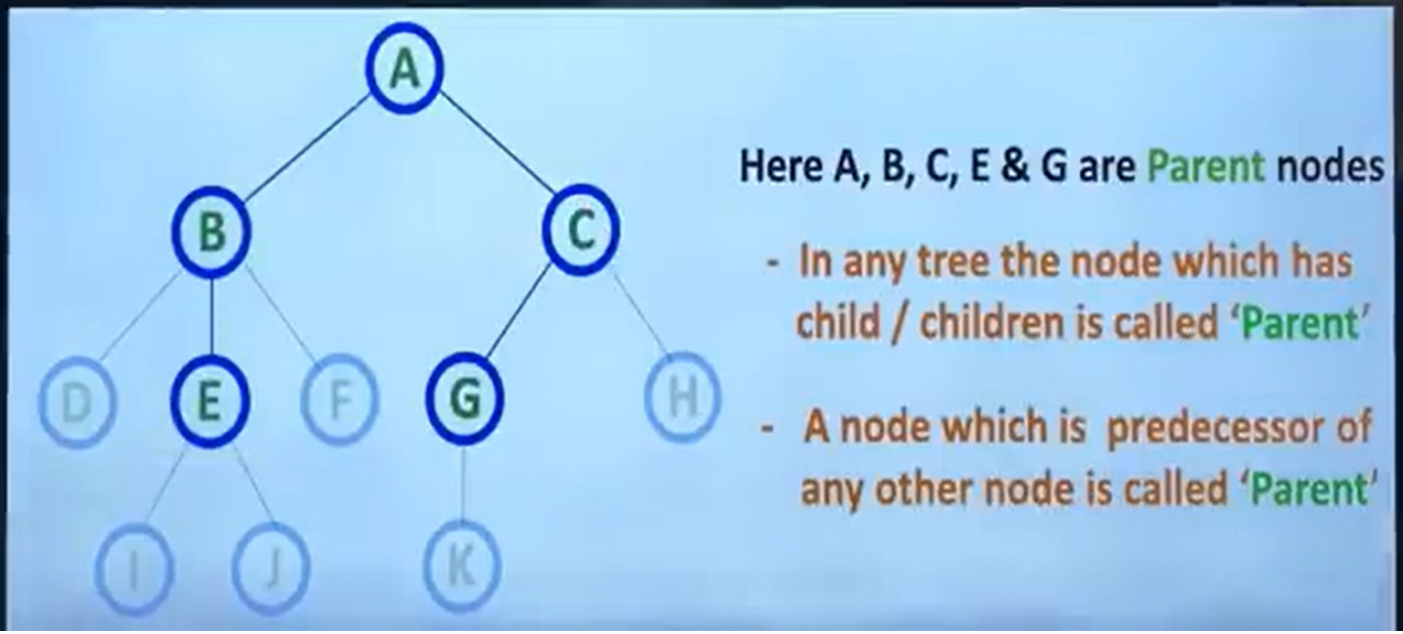
Edge

- In a tree data structure, the connecting link between any two nodes is called as EDGE. In a tree with 'N' number of nodes there will be exactly of 'N-1' number of edges.



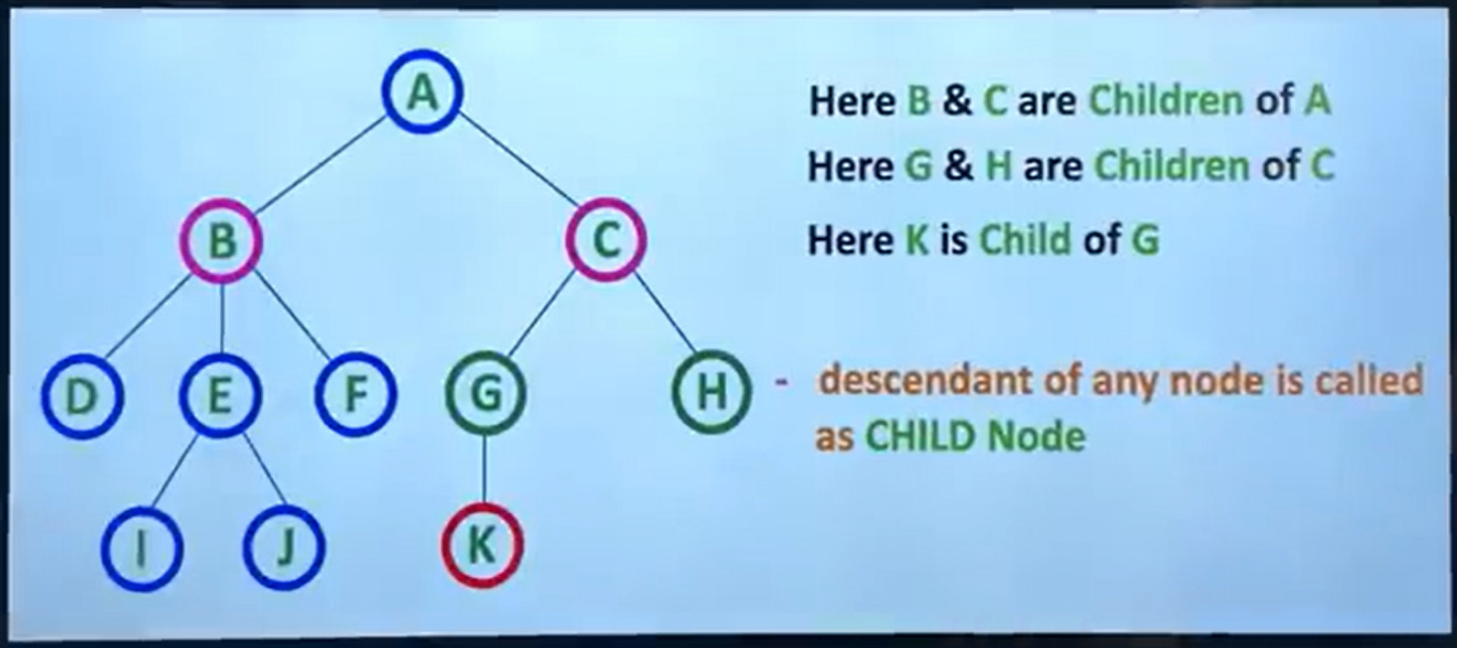
Parent

- In a tree data structure, the node which is predecessor of any node is called as PARENT NODE.
- In simple words, the node which has branch from it to any other node is called as parent node. Parent node can also be defined as "The node which has child / children".



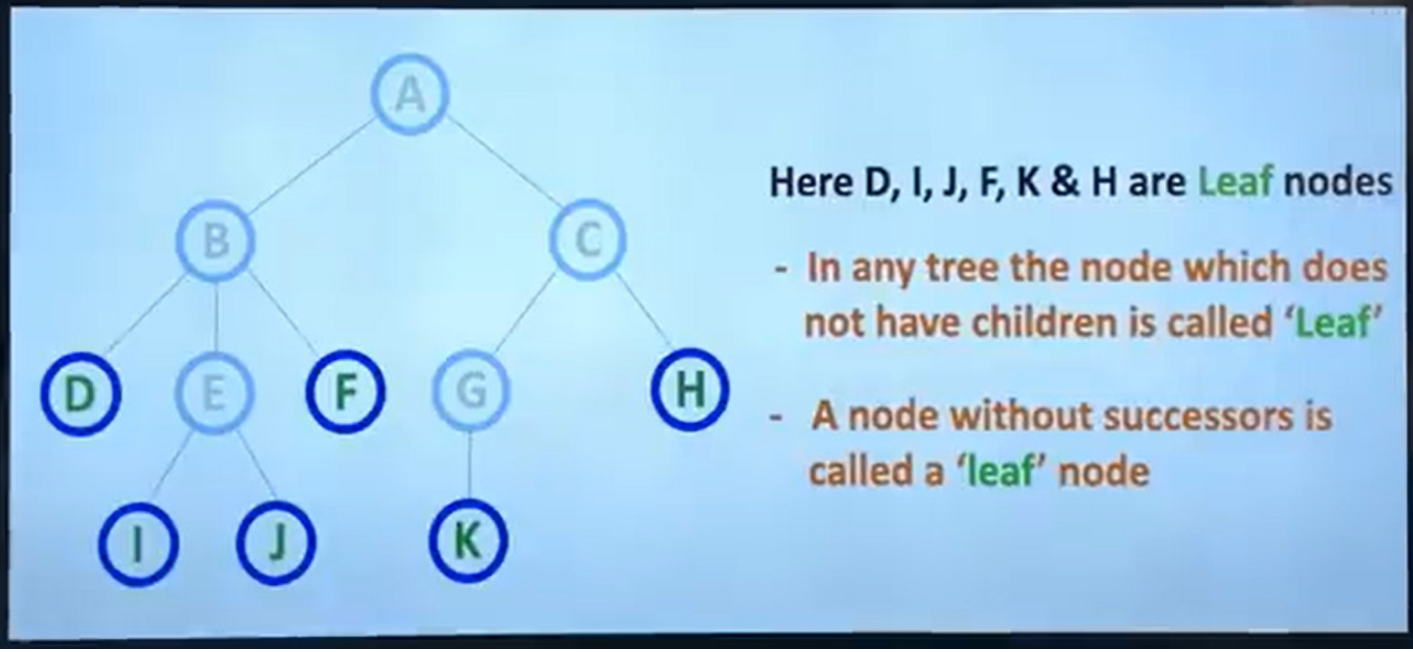
Child

- In a tree data structure, the node which is descendant of any node is called as CHILD Node.
- In simple words, the node which has a link from its parent node is called as child node. In a tree, any parent node can have any number of child nodes. In a tree, all the nodes except root are child nodes.



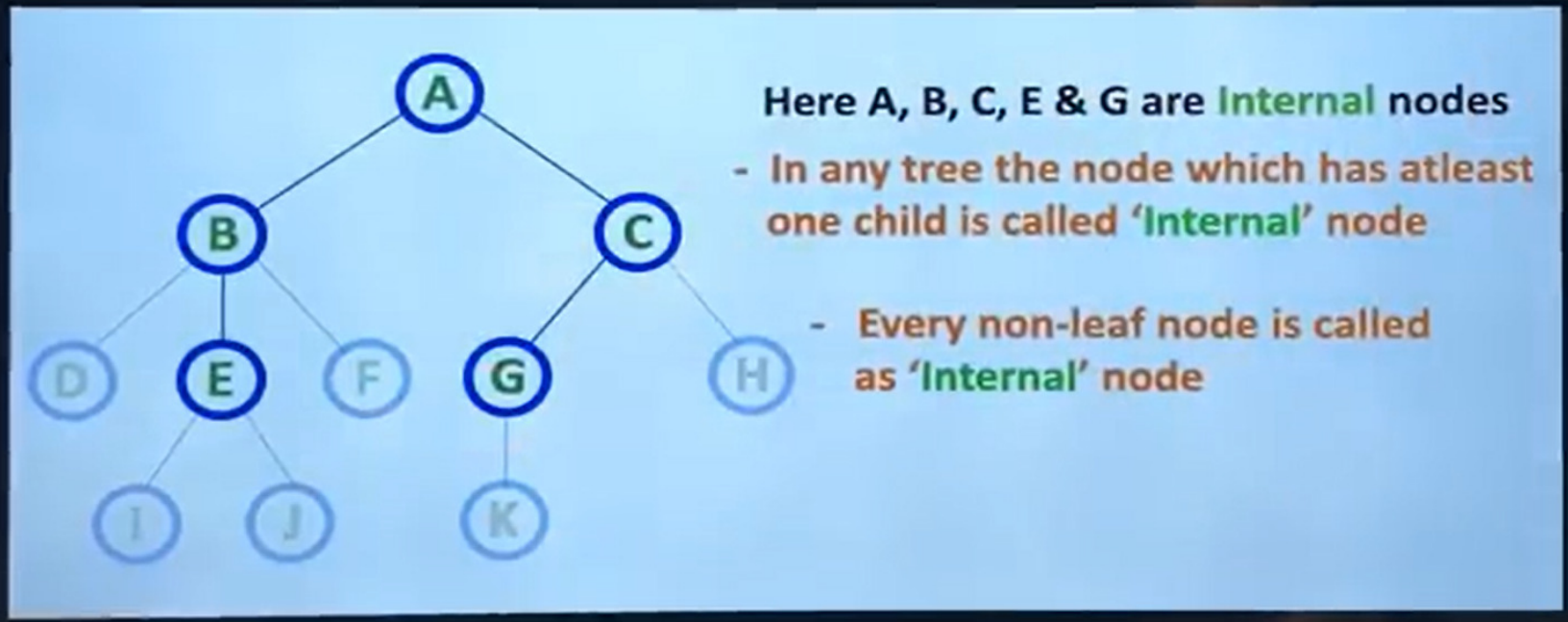
Leaf / External

- In a tree data structure, the node which does not have a child is called as LEAF Node. In simple words, a leaf is a node with no child.
- In a tree data structure, the leaf nodes are also called as External Nodes. External node is also a node with no child. In a tree, leaf node is also called as 'Terminal' node.



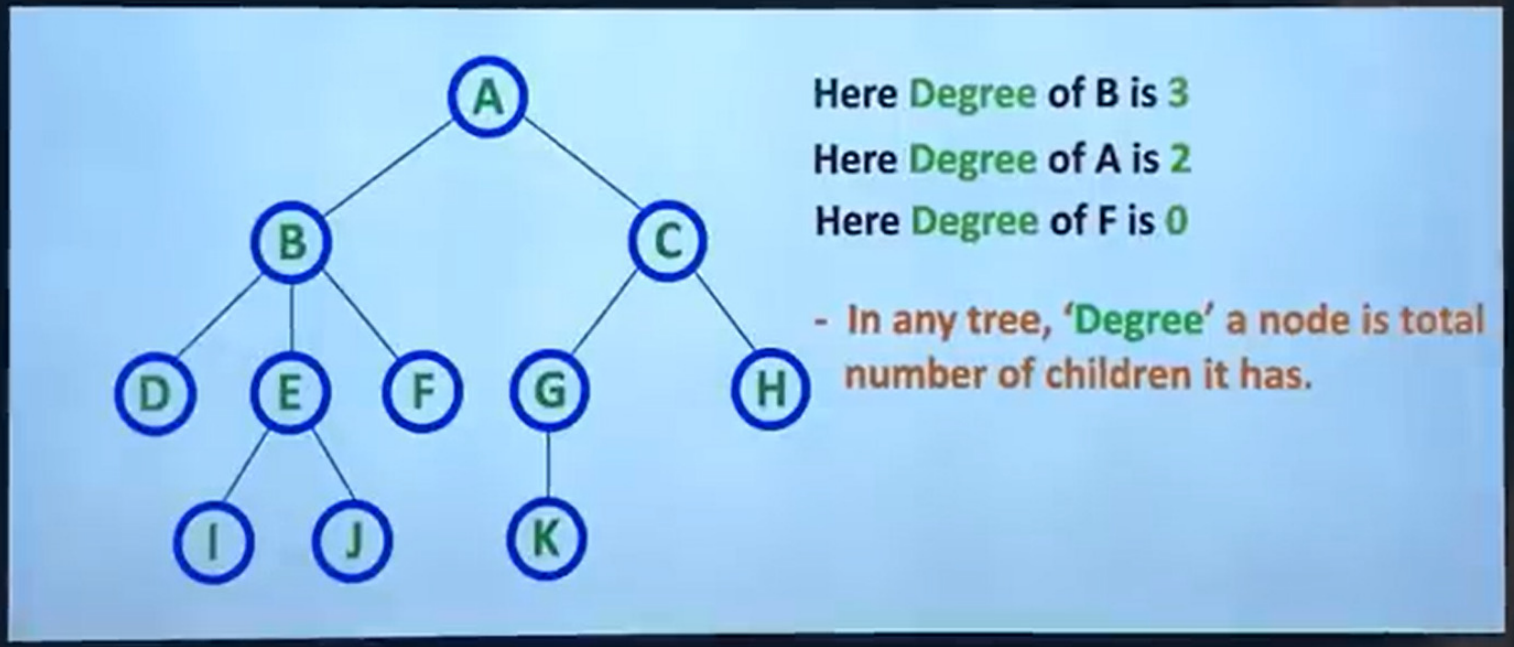
Internal Nodes

- In a tree data structure, the node which has at least one child is called as INTERNAL Node. In simple words, an internal node is a node with at least one child.
- In a tree data structure, nodes other than leaf nodes are called as Internal Nodes. The root node is also said to be Internal Node if the tree has more than one node. Internal nodes are also called as 'Non-Terminal' nodes.



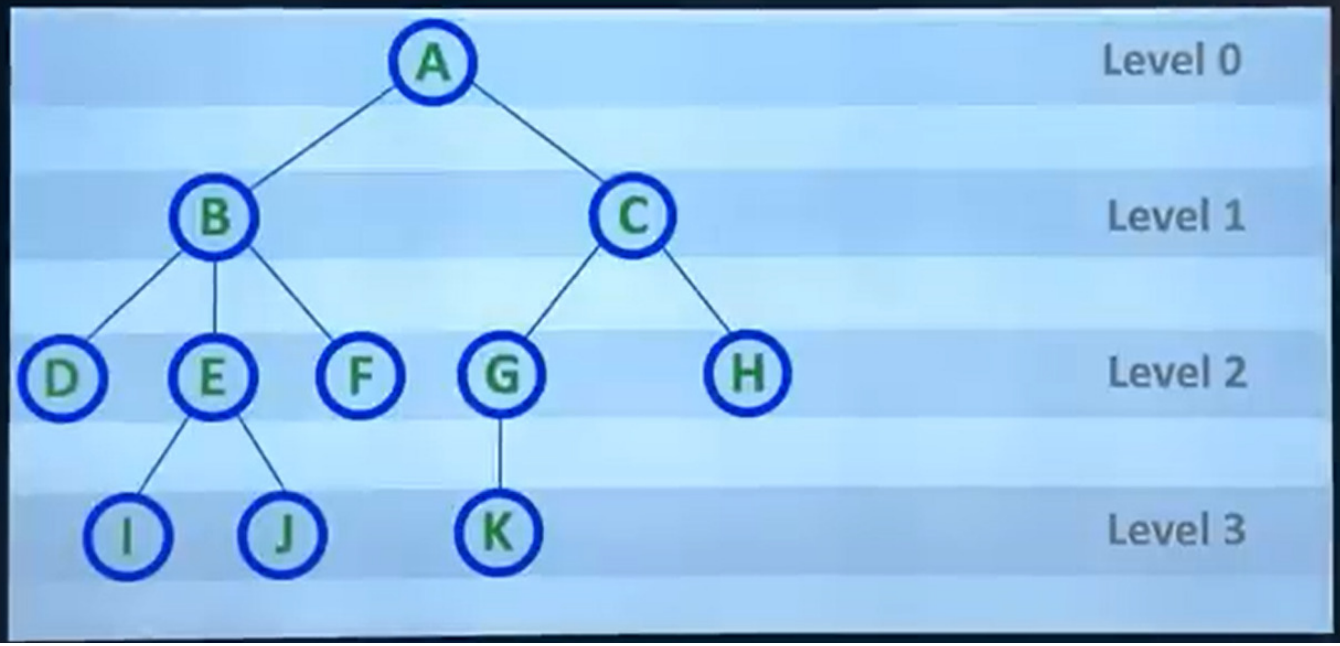
Degree

- In a tree data structure, the total number of children of a node is called as DEGREE of that Node. In simple words, the Degree of a node is total number of children it has.
- The highest degree allowed of a node in a tree is called as 'Degree of Tree'



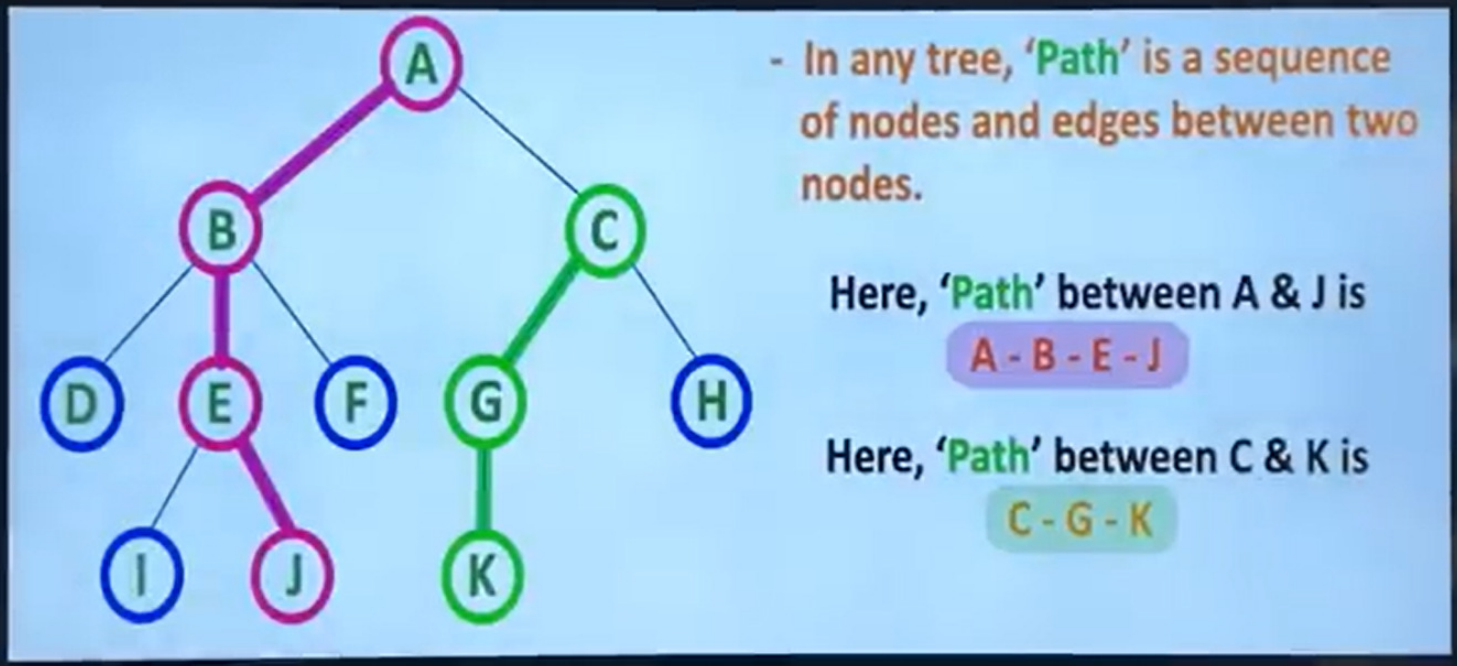
Level / Depth / Height

- In a tree data structure, the root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2 and so on...
- In simple words, in a tree each step from top to bottom is called as a Level and the Level count starts with '0' and incremented by one at each level (Step).



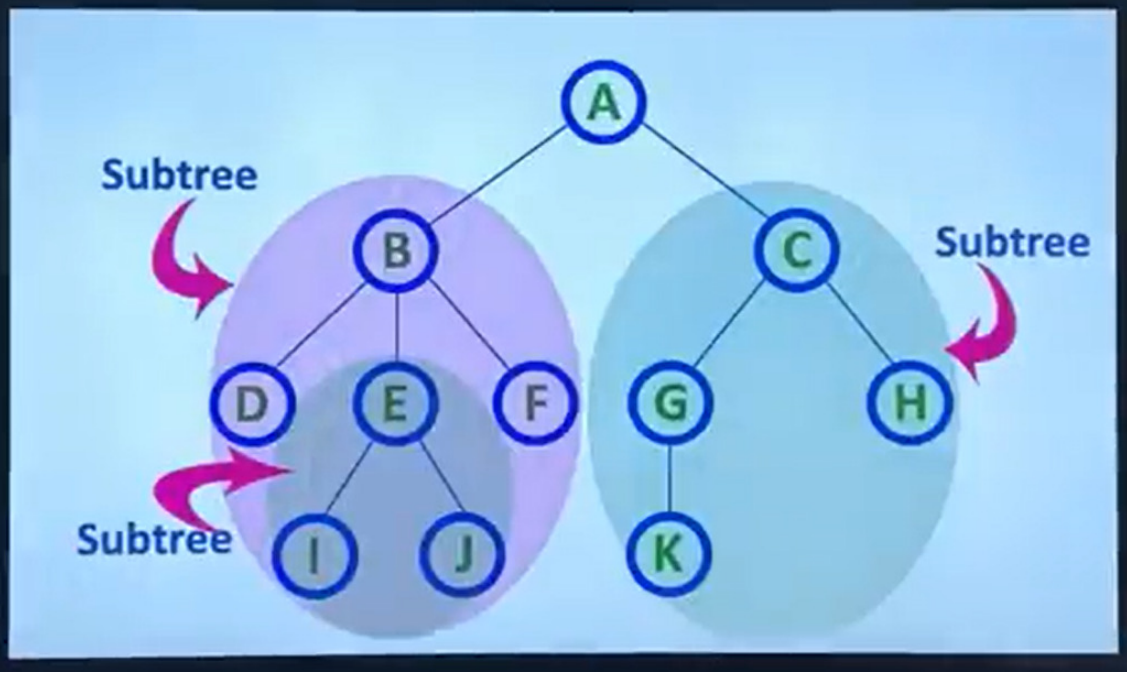
Path

- In a tree data structure, the sequence of Nodes and Edges from one node to another node is called as PATH between that two Nodes. Length of a Path is total number of edge in that path. In below example the path A - B - E - J has length 4.



Sub Tree

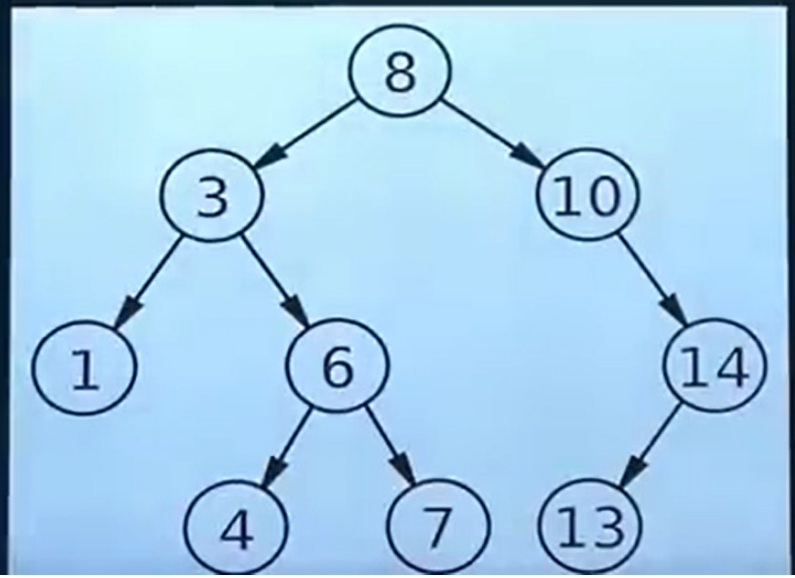
- In a tree data structure, each child from a node forms a subtree recursively. Every child node will form a subtree on its parent node.



Binary tree

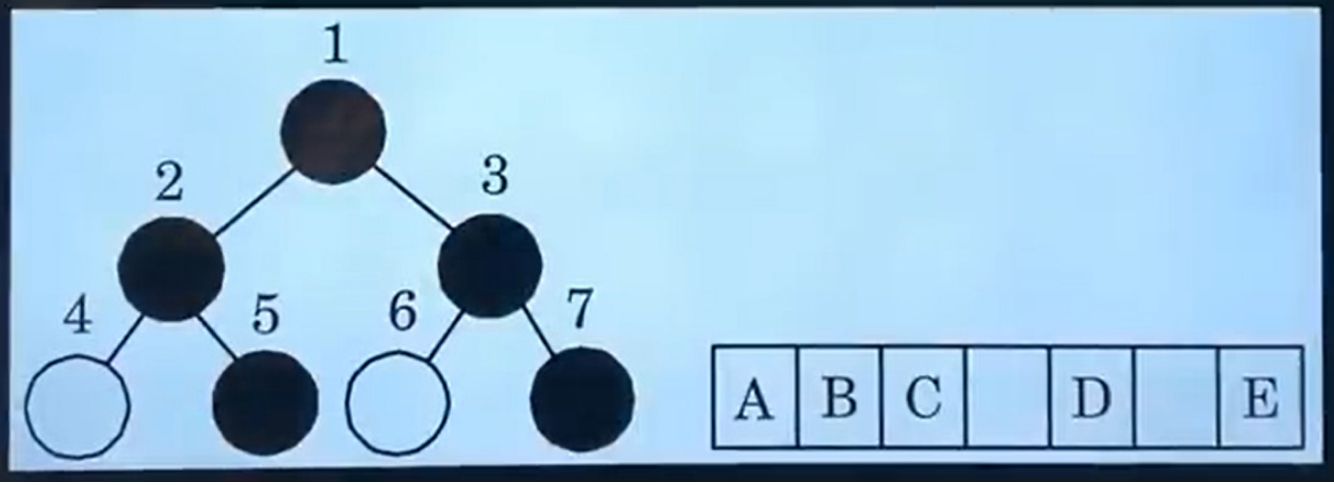
- A binary tree T is defined as a finite set of elements called nodes such that,
  - T is empty (null tree)
  - T contain a distinguished node R, called the root of T, and the remaining nodes of T form an ordered pair of disjoint binary tree T_1 and T_2
- Direct: - A tree T in which any node can have maximum two children (left and right)

```
struct node {
int data;
struct node* left;
struct node* right;
}
```



Binary tree representation using array

- Binary tree can be represented using an array
- General representation
- The root is at index '1'
- For any given node at position 'i'
  - Left Child is at position 2*i
  - Right Child is at position 2*i + 1
- If a node does not have a left or right child, that position in the array remains empty or is filled with a special value indicating it's vacant (like null or -1)



Linked representation of binary tree

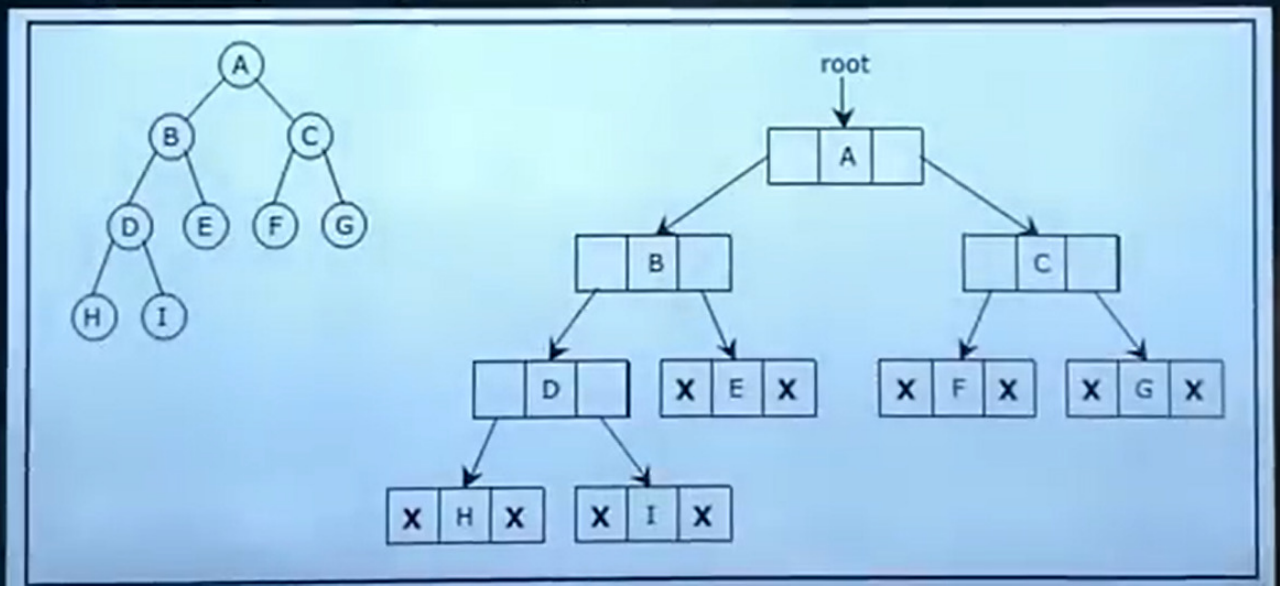
1. Consider a binary tree T which uses three parallel arrays, INFO, LEFT and RIGHT, and a pointer variable ROOT.
2. First of all, each node N of T will correspond to a location K such that :
   a. INFO[K] contains the data at the node N.
   b. LEFT[K] contains the location of the left child of node N.
   c. RIGHT[K] contains the location of the right child of node N.
3. ROOT will contain the location of the root R of T.
4. If any subtree is empty, then the corresponding pointer will contain the null value.
5. If the tree T itself is empty, then ROOT will contain the null value.
6. INFO may actually be a linear array of records or a collection of parallel arrays.

Binary tree representation using Linked List

- A binary tree can be efficiently represented using a linked list structure where each node of the tree is represented by a separate node in the linked list. This linked structure is typically referred to as a "node-based" representation.

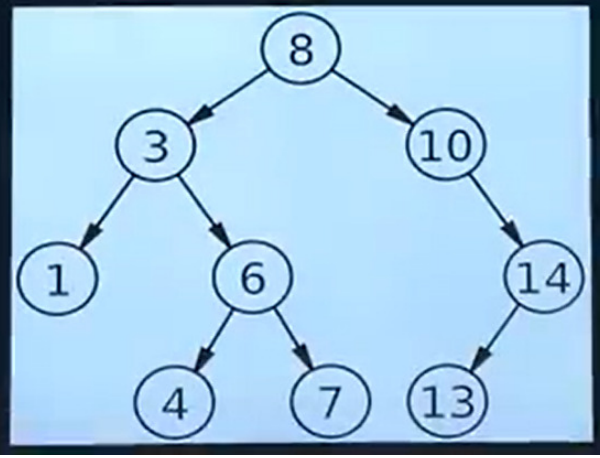
Each node in the linked list contains the following components:

- **Data**: The value stored in the node.
- **Left Pointer**: A pointer pointing to the left child node.
- **Right Pointer**: A pointer pointing to the right child node.



Traversal of binary tree

- The process of visiting (checking and/or updating) each node in a tree data structure, exactly once in called tree traversal. Such traversals are classified by the order in which the nodes are visited.
- Unlike linked lists, one-dimensional arrays and other linear data structures, which are canonically traversed in linear order, trees may be traversed in multiple ways.
- They may be traversed in depth-first or breadth-first order. There are three common ways to traverse them in depth-first order: in-order, pre-order and post-order. Beyond these basic traversals, various more complex or hybrid schemes are possible, such as depth-limited searches like iterative deepening depth-first search.



- Some applications do not require that the nodes be visited in any particular order as long as each node is visited precisely once. For other applications, nodes must be visited in an order that preserves some relationship.
- These steps can be done *in any order*. If (L) is done before (R), the process is called left-to-right traversal, otherwise it is called right-to-left traversal. The following methods show left-to-right traversal:

Pre-order (Root L R)

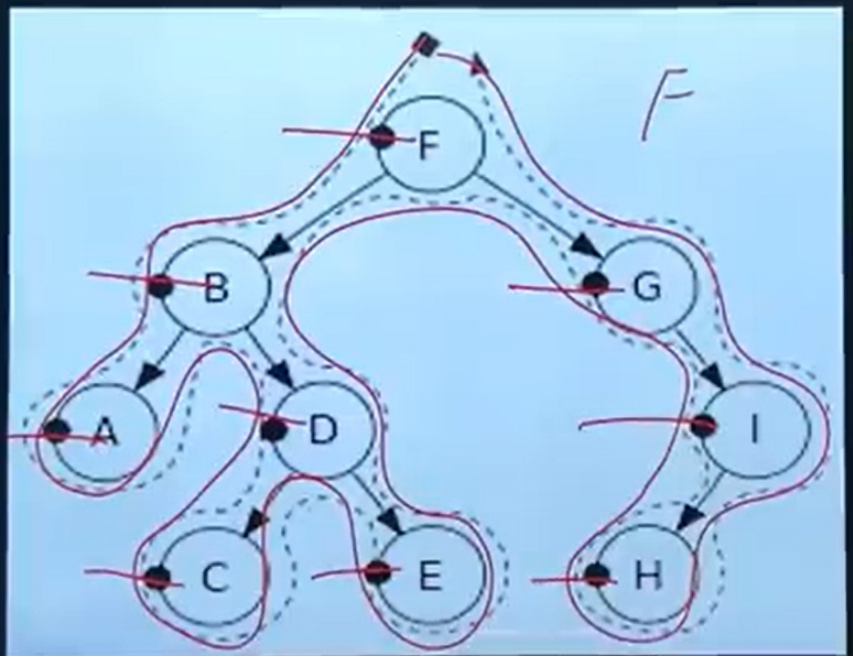
Root L R

L Root R

L R Root

Pre-order: F, B, A, D, C, E, G, I, H.

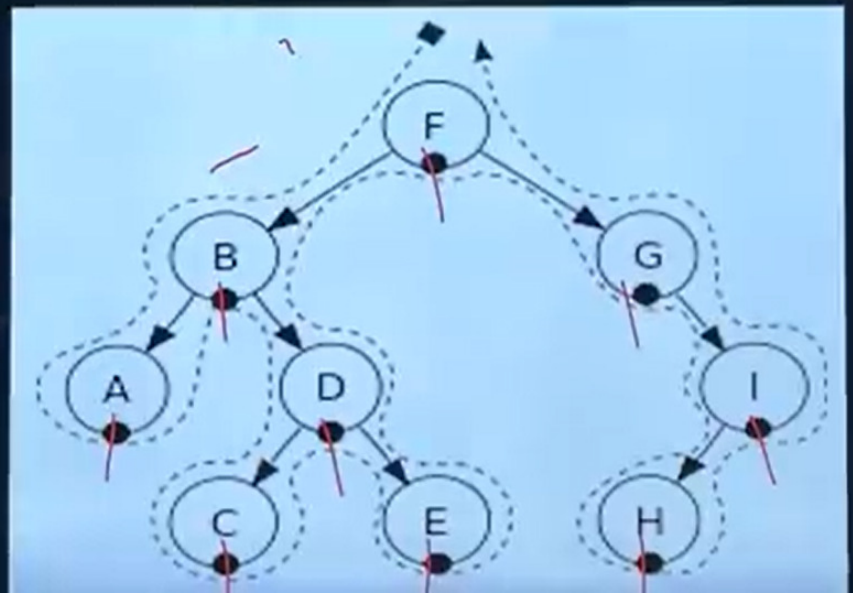
- Check if the current node is empty or null.
- Display the data part of the root (or current node).
- Traverse the left subtree by recursively calling the pre-order function.
- Traverse the right subtree by recursively calling the pre-order function.



In-order (L root R)

In-order: A, B, C, D, E, F, G, H, I.

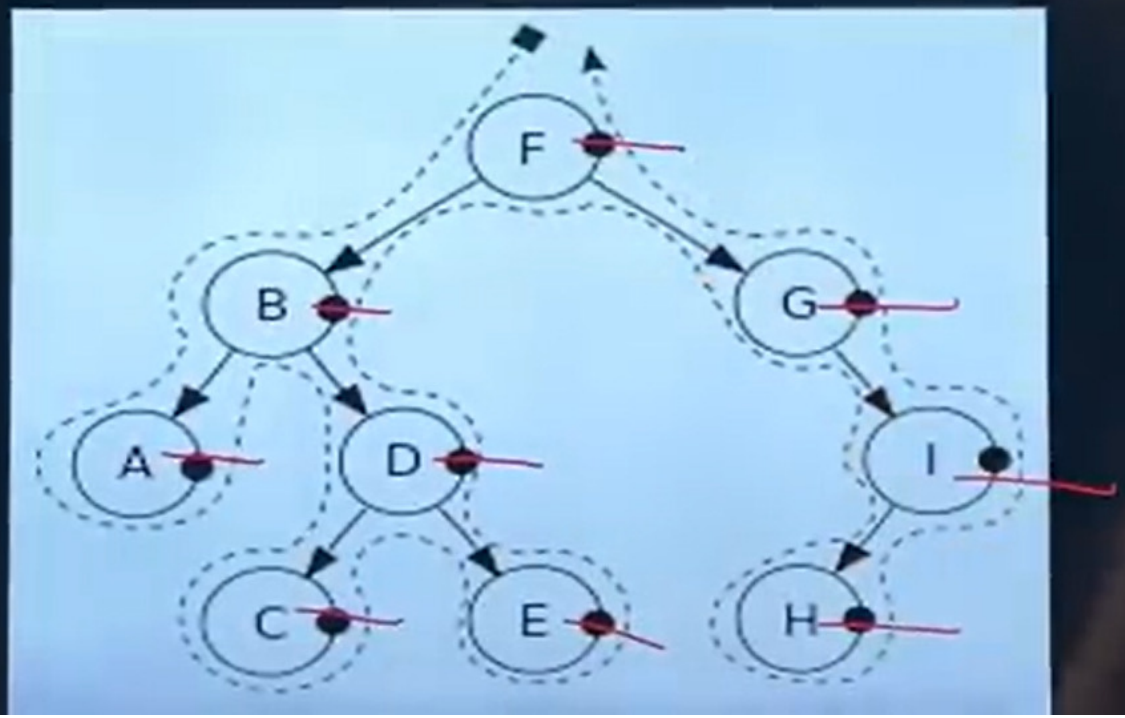
- Check if the current node is empty or null.
- Traverse the left subtree by recursively calling the in-order function.
- Display the data part of the root (or current node).
- Traverse the right subtree by recursively calling the in-order function.
- In a binary search tree, in-order traversal retrieves data in sorted order.



Post-order (L R Root)

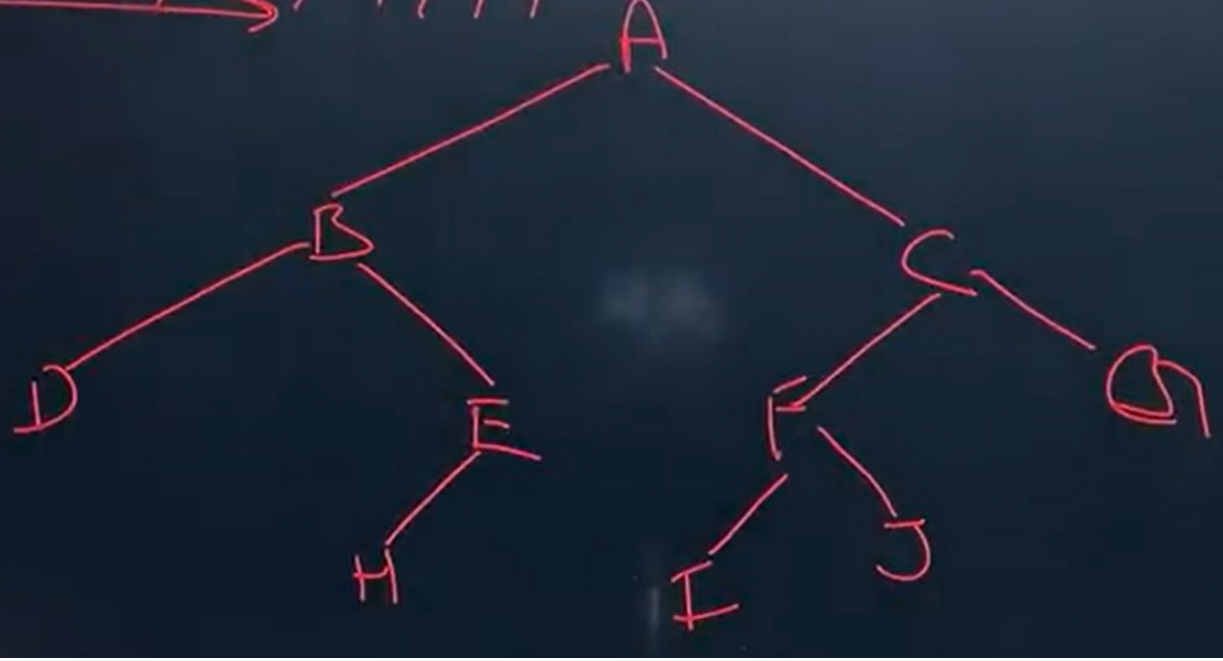
Post-order: A, C, E, D, B, H, I, G, F.

- Check if the current node is empty or null.
- Traverse the left subtree by recursively calling the post-order function.
- Traverse the right subtree by recursively calling the post-order function.
- Display the data part of the root (or current node).



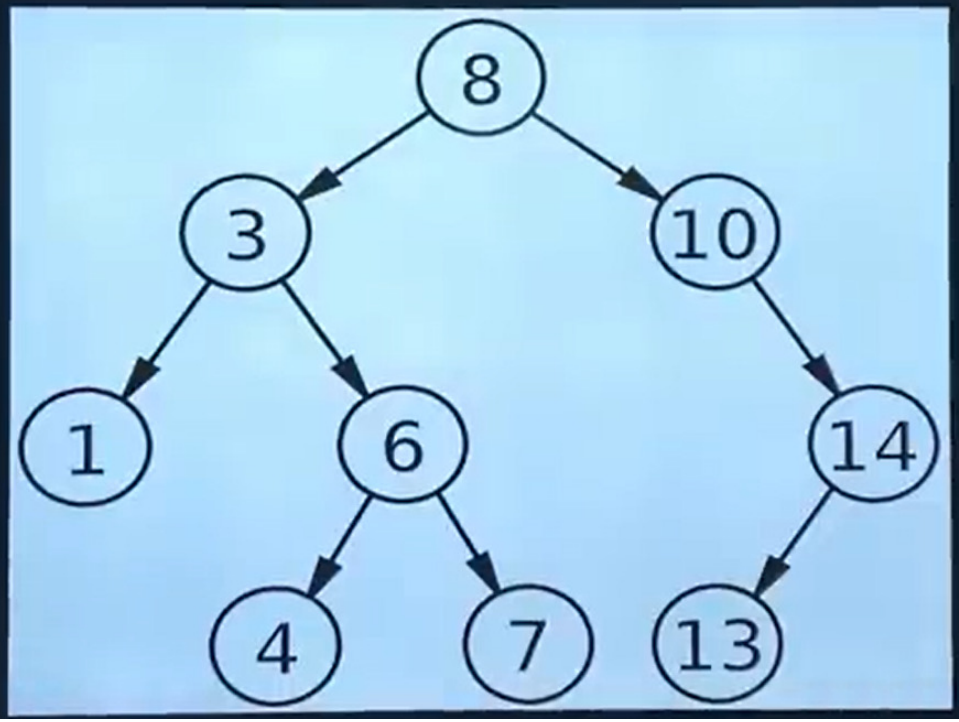
Inorder : D B H E A I F J C G

Preorder : A B D E H C F I J G



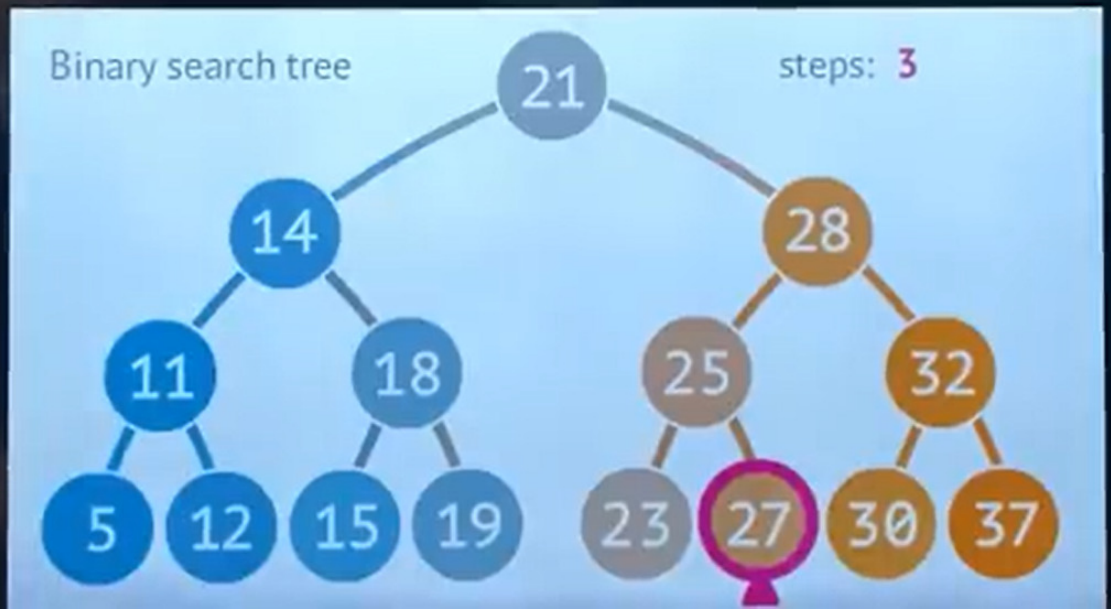
Binary search tree / Ordered tree / Sorted binary tree

- A binary search tree (BST) is a binary tree in which left subtree of a node contains a key less than the node's key and right subtree of a node contains only the nodes with key greater than the node's key. Left and right sub tree must each also be a binary search tree.



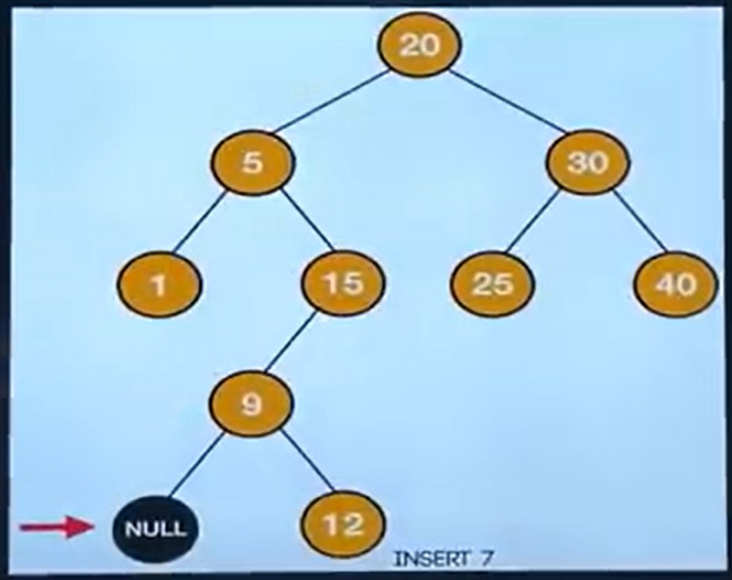
Searching

- We begin by examining the root node. If the tree is *null*, the key we are searching for does not exist in the tree. Otherwise, if the key equals that of the root, the search is successful and we return the node.
- If the key is less than that of the root, we search the left subtree. Similarly, if the key is greater than that of the root, we search the right subtree.
- This process is repeated until the key is found or the remaining subtree is *null*. If the searched key is not found after a *null* subtree is reached, then the key is not present in the tree.



Insertion

- Insertion begins as a search would begin; if the key is not equal to that of the root, we search the left or right subtrees as before.
- Eventually, we will reach an external node and add the new key-value pair (here encoded as a record 'new Node') as its right or left child, depending on the node's key.
- In other words, we examine the root and recursively insert the new node to the left subtree if its key is less than that of the root, or the right subtree if its key is greater than or equal to the root.

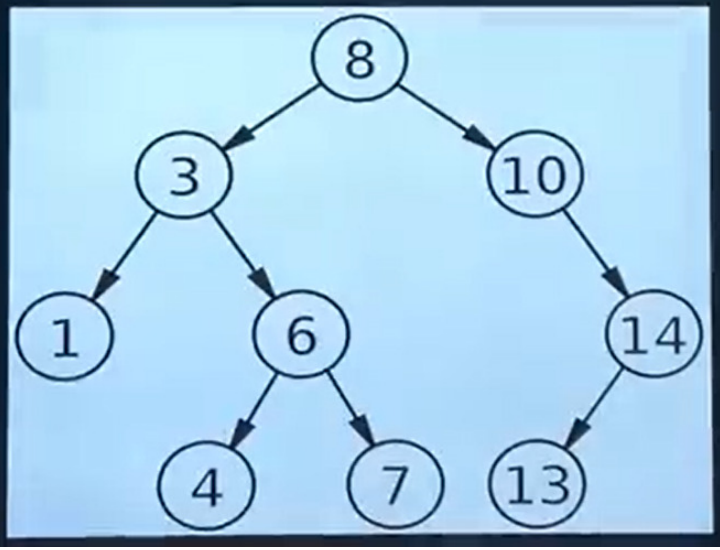


Deletion

- Deleting a node with no children: simply remove the node from the tree.
- Deleting a node with one child: remove the node and replace it with its child.
- Deleting a node with two children: call the node to be deleted *D*. Do not delete *D*.

Instead, choose either its in-order predecessor node or its in-order successor node as replacement node *E* (s. figure). Copy the user values of *E* to *D*.

If *E* does not have a child simply remove *E* from its previous parent *G*. If *E* has a child, say *F*, it is a right child. Replace *E* with *F* at *E*'s parent.

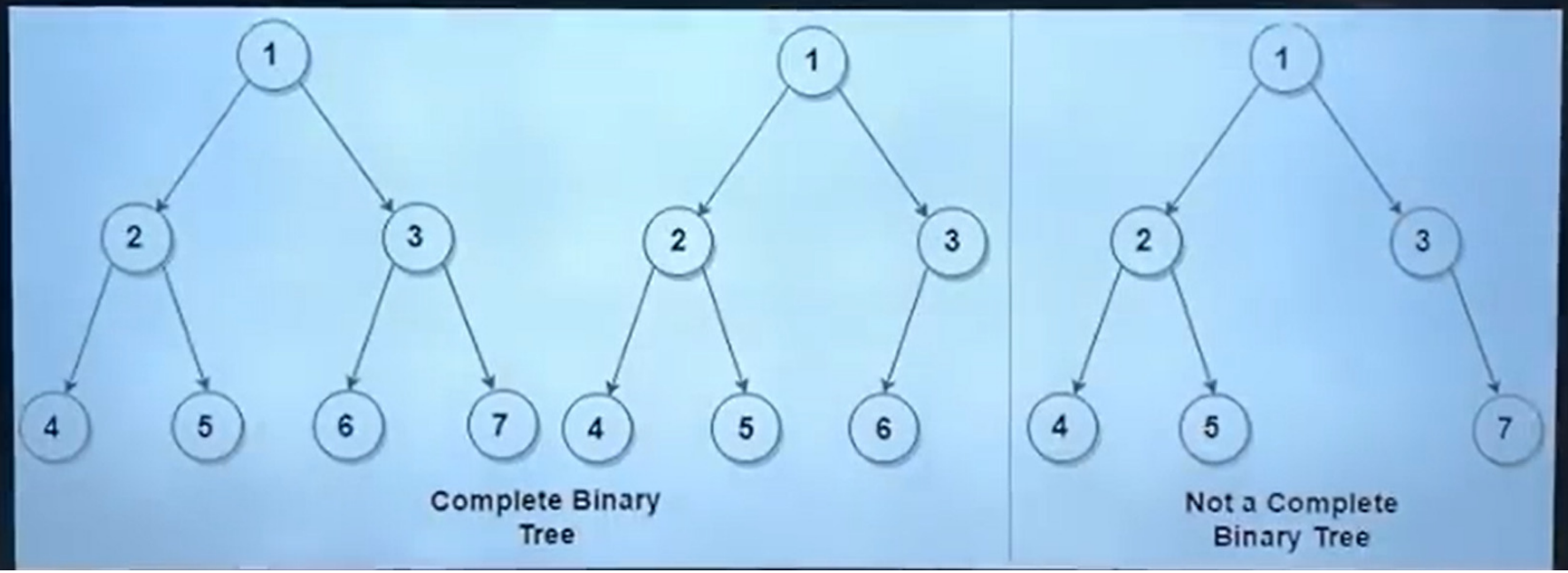


Algorithm	Average	Worst case
Space	O(n)	O(n)
Search	O(log n)	O(n)
Insert	O(log n)	O(n)
Delete	O(log n)	O(n)

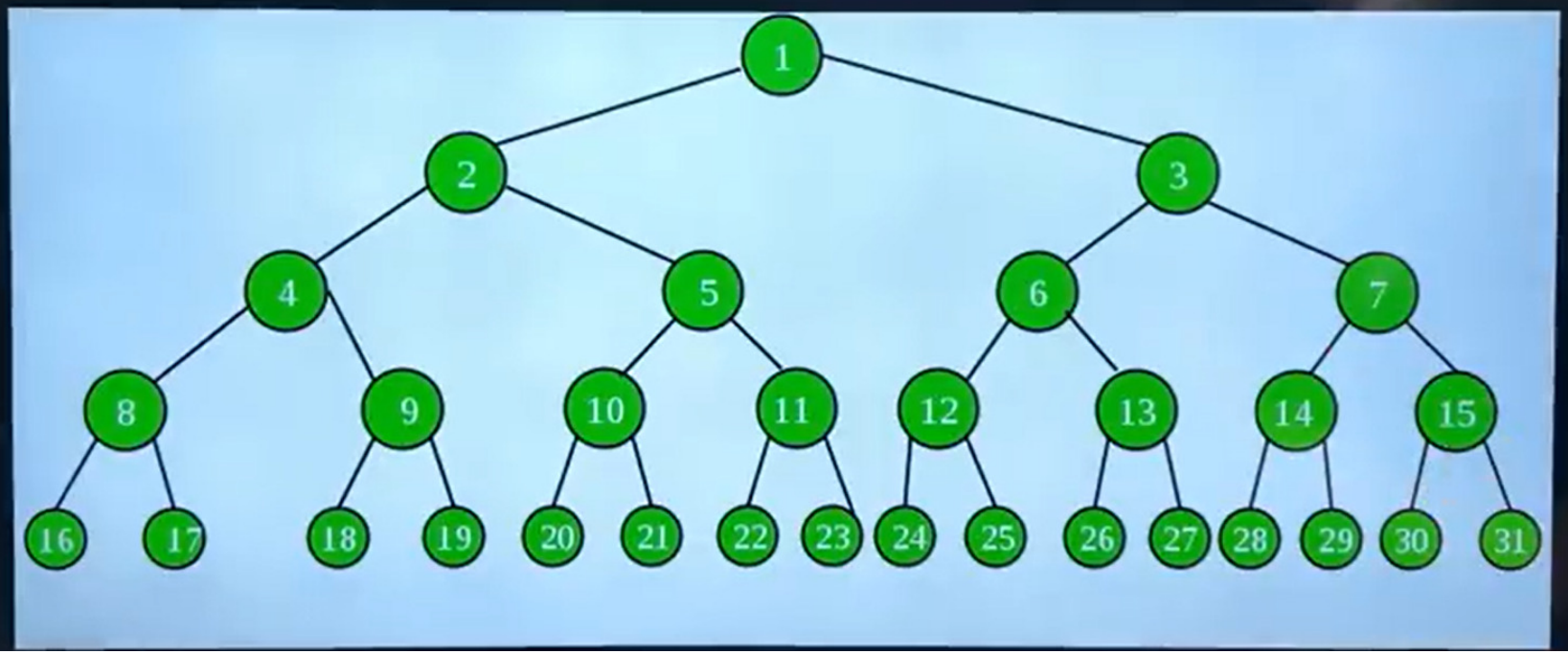
The major advantage of binary search trees over other data structures is that the related sorting algorithms and search algorithm such as in-order traversal can be very efficient; they are also easy to code

Complete Binary Tree

- Consider a binary tree T, the maximum number of nodes at height h is 2^h nodes.
- The binary tree T is said to be complete binary tree, if all its level except possibly the last, have the maximum number of nodes and if all the nodes at the last level appear as far left as possible.

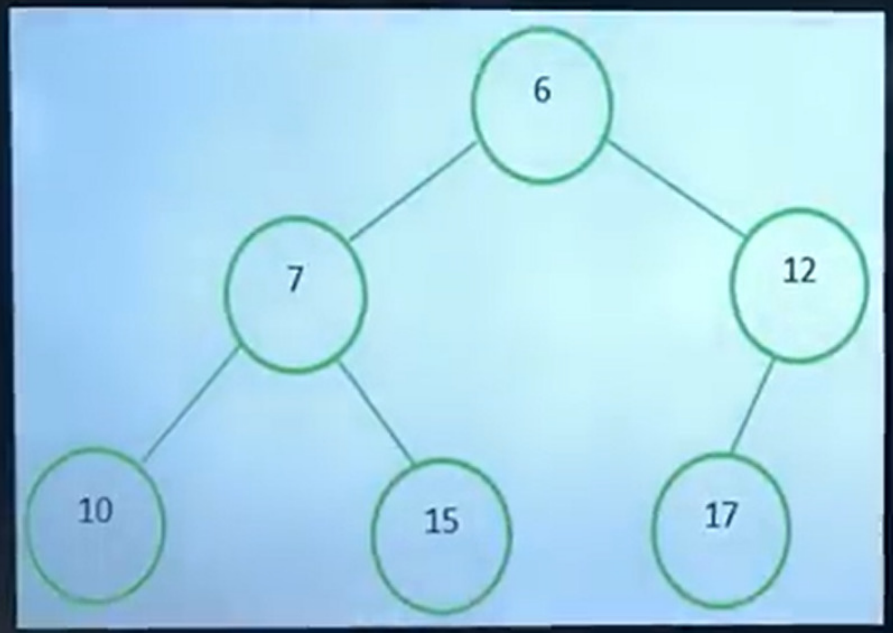


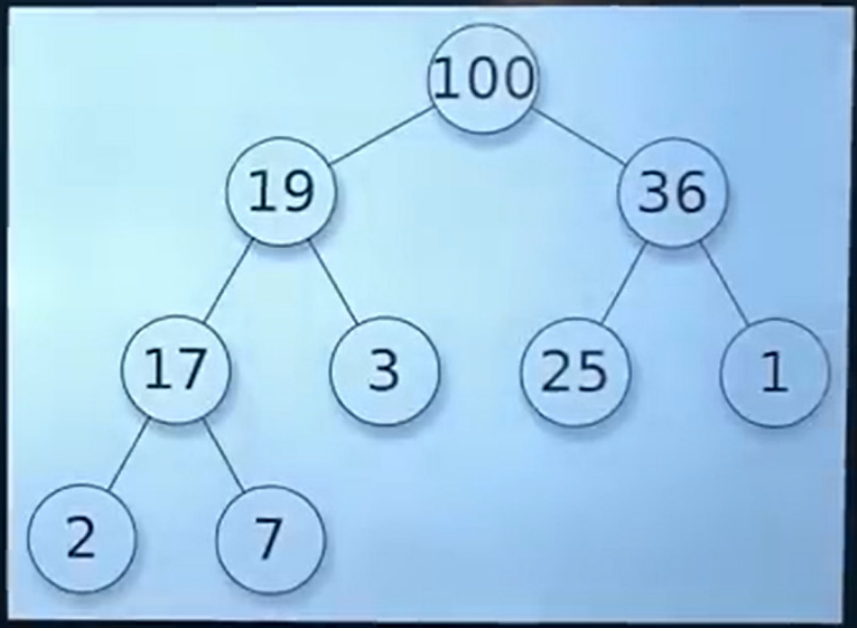
- One can easily determine the children and parent of a node k in any complete tree T
- Specially the left and right children of the node K are 2*k, 2*k + 1 and the parent of k is the node lower bound(k/2)



Heap

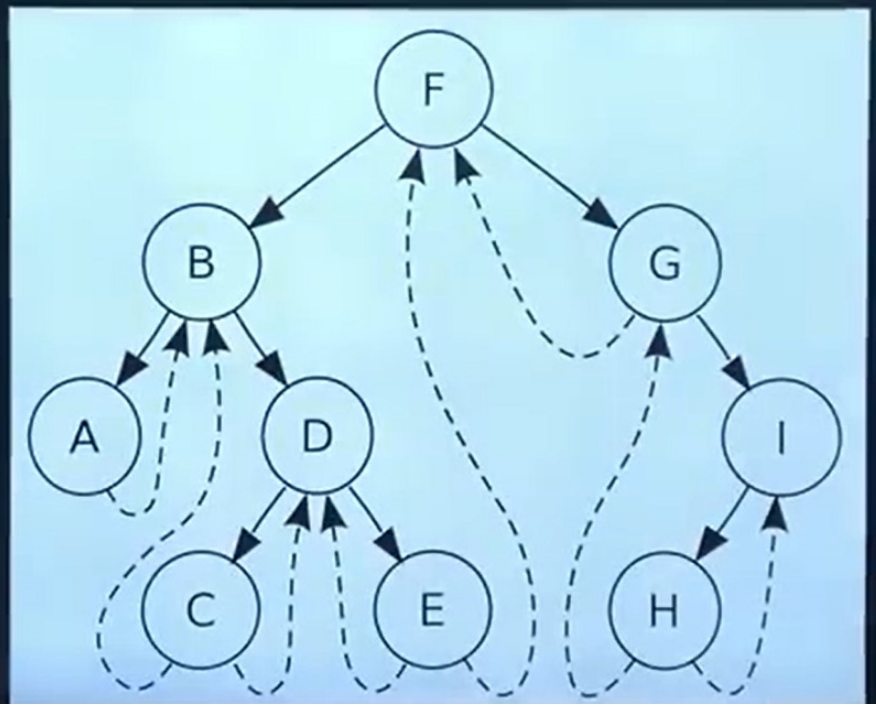
- Suppose H is a complete binary tree with n elements, H is called a Heap, if each node N of H has following properties:
  - The value of N is greater than to the value at each of the children of N then it is called Max heap.
  - A min heap is defined as the value at N is less than the value at any of the children of N.



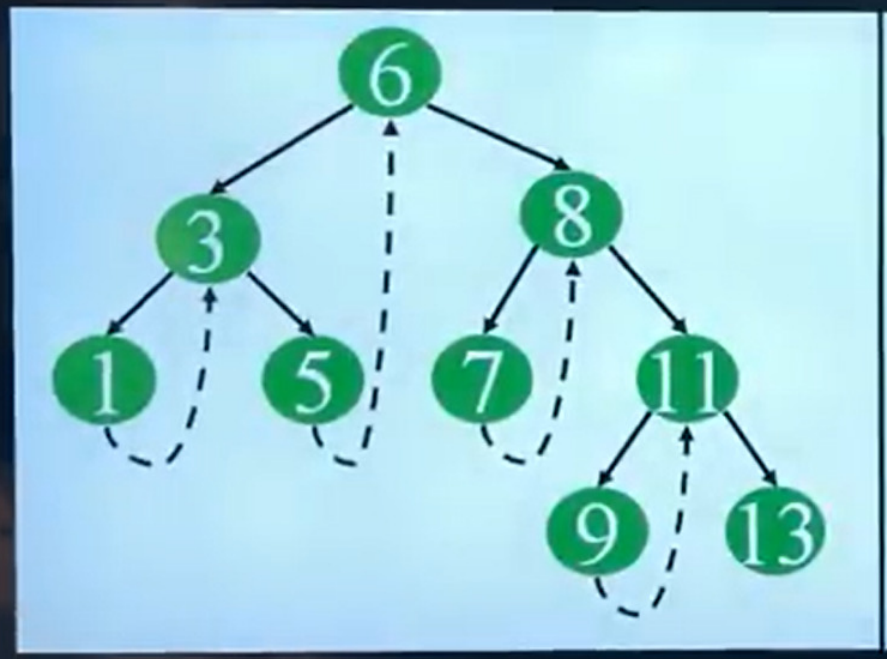


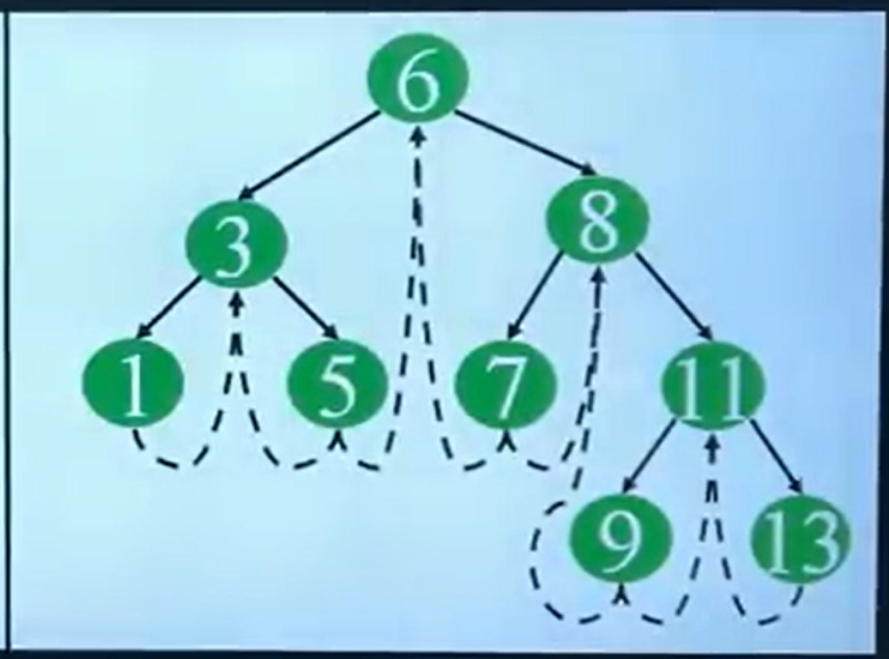
Threaded binary tree

- A threaded binary tree is a modified binary tree that uses null pointers to link to the next node in an in-order sequence, optimizing in-order traversal.
- **Purpose**: Utilizes null pointers to store references (threads) to nodes, aiding efficient in-order traversal without recursion or stacks.



- **Types**:
  - **Single Threaded**: Nodes threaded towards either in-order predecessor or successor.
  - **Double Threaded**: Nodes threaded towards both predecessor and successor.





- **Benefits**:
  - Allows stack-less in-order traversal.
  - Makes efficient use of memory by replacing null pointers with threads.
- This tree variant is beneficial when recursive or stack-based traversals aren't feasible. However, its popularity has decreased with newer data structures.