DATA STRUCTURES USING C

NOTES

Prepared by

Prof. Prabhu Kichadi, M. Tech
9880437187

UNIT – IV

CONTENTS

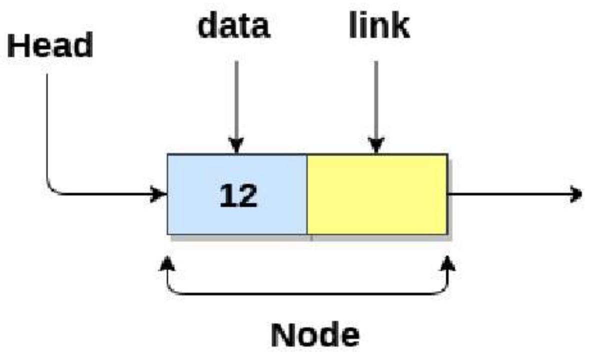
Linked list: Basic Concepts – Definition and Representation of linked list, Types of linked lists - Singly linked list, doubly linked list, Circular linked list Doubly Circular Linked list; Representation of Linked list in Memory; Operations on Singly linked lists – Traversing, Searching, Insertion, Deletion;
Trees: Definition; Tree terminologies –node, root node, parent node, ancestors of a node, siblings, terminal & non-terminal nodes, degree of a node, level, edge, path, depth;
Binary tree: Type of binary trees - strict binary tree, complete binary tree, binary search tree and heap tree; Array representation of binary tree. Traversal of binary tree; preorder, in order and post order traversal

Linked List:

*Linked List is a linear data structure and it is a collection of objects called nodes that are randomly stored in the memory. A node contains two fields i.e., **1. Info Field** - data stored at that address and **2. Address Field** - the pointer which contains the address of the next node in the memory.*

Example:

A linked list is like a train where each bogie is connected with links. Different types of linked lists exist to make lives easier, like an image viewer, music player, or when you navigate through web pages.

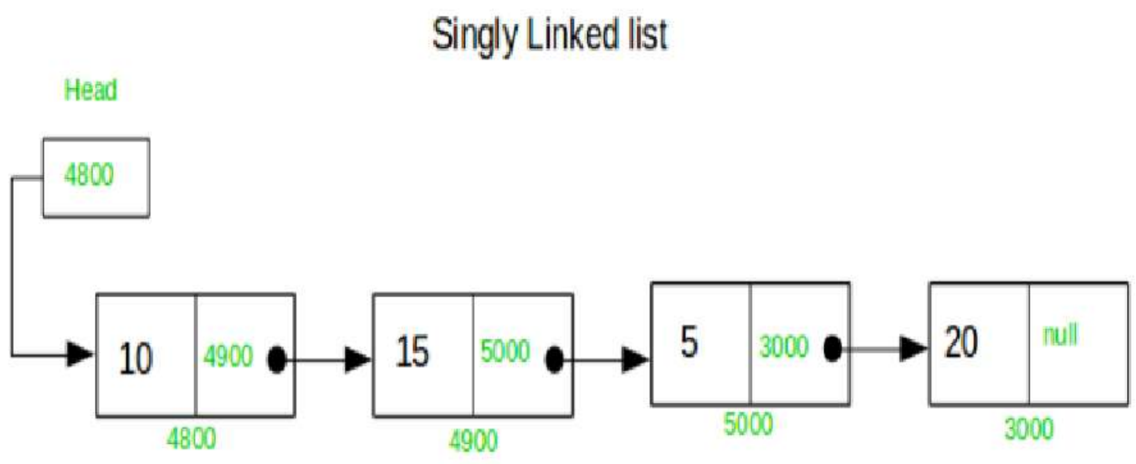


Uses of Linked List:

- The list is not required to be contiguously present in the memory. The node can reside anywhere in the memory and linked together to make a list. This achieves optimized utilization of space.
- list size is limited to the memory size and does not need to be declared in advance.
- Empty node cannot be present in the linked list.
- We can store values of primitive types or objects in the singly linked list.

Representation Of Linked List:

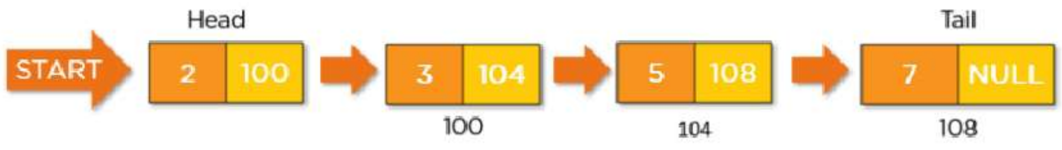
A node in the singly linked list consists of two parts: **data part** and **link part**. Data part of the node stores actual information that is to be represented by the node while the link part of the node stores the address of its immediate successor.



Types Of Linked Lists: There are 4 types of linked list can be identified.

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List
4. Doubly Circular Linked List

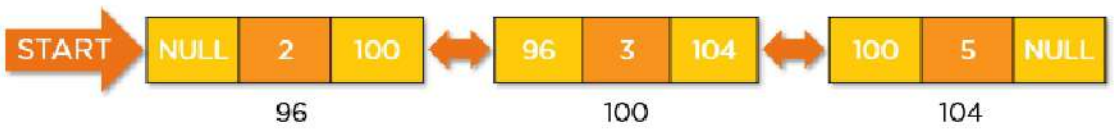
1. Singly Linked List: A singly linked list is a unidirectional linked list. So, you can only traverse it in one direction, i.e., from head node to tail node.



There are many applications for singly linked lists. One common application is to store a list of items that need to be processed in order. For example, a singly linked list can be used to store a list of tasks that need to be completed, with the head node representing the first task to be completed and the tail node representing the last task to be completed.

2. Doubly Linked List:

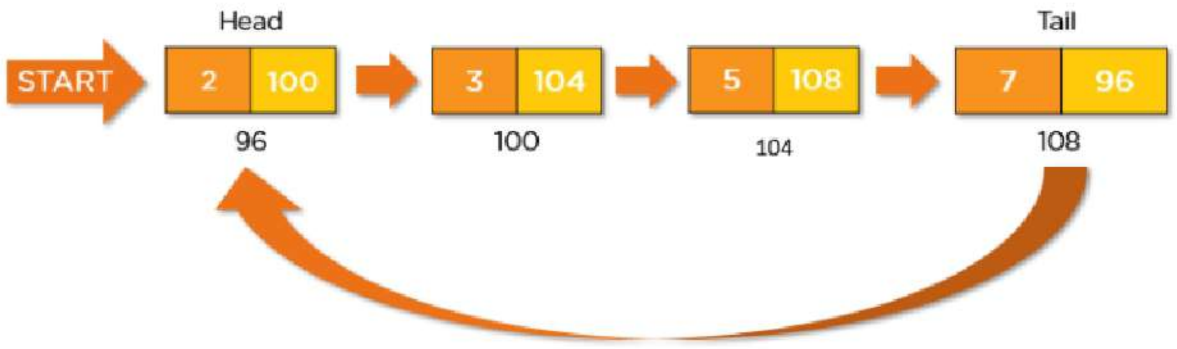
A doubly linked list is a bi-directional linked list. So, you can traverse it in both directions. Unlike singly linked lists, its nodes contain one extra pointer called the previous pointer. This pointer points to the previous node.



A doubly linked list of singly linked lists is a data structure that consists of a set of singly linked lists (SLLs), each of which is doubly linked. It is used to store data in a way that allows for fast insertion and deletion of elements.

3. Circular Linked List:

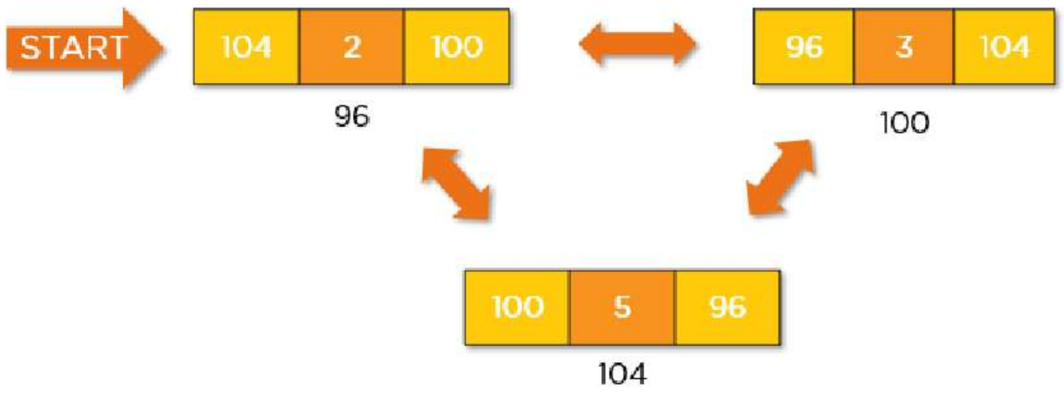
A circular Linked list is a unidirectional linked list.
So, you can traverse it in only one direction. But this type of linked list has its last node pointing to the head node. So while traversing, you need to be careful and stop traversing when you revisit the head node.



A circular linked list is a type of data structure that uses linked list technology to store data in a linear, sequential fashion. However, unlike a traditional linked list, a circular linked list has no beginning or end – it is essentially a ring of nodes. This makes circular linked lists ideal for applications where data needs to be processed in a continuous loop, such as in real-time applications or simulations.

4. Doubly Circular Linked List:

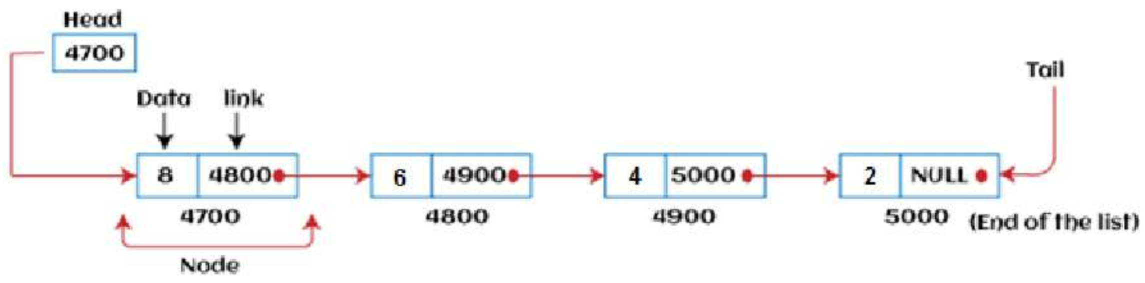
A circular doubly linked list is a mixture of a doubly linked list and a circular linked list.



A doubly circular linked list (DCL) is a variation of the standard doubly linked list. In a DCL, the first and last nodes are linked together, forming a circle. This allows for quick and easy traversal of the entire list without the need for special case handling at the beginning or end of the list.

Representation Of Linked List in Memory:

Linked list can be represented as the connection of nodes in which each node points to the next node of the list. The representation of the linked list is shown below –



Advantages of Linked list

The advantages of using the Linked list are given as follows -

- **Dynamic data structure -** The size of the linked list may vary according to the requirements. Linked list does not have a fixed size.
- **Insertion and deletion -** Unlike arrays, insertion, and deletion in linked list is easier. Array elements are stored in the consecutive location, whereas the elements in the linked list are stored at a random location. To insert or delete an element in an array, we have to shift the elements for creating the space. Whereas, in linked list, instead of shifting, we just have to update the address of the pointer of the node.
- **Memory efficient -** The size of a linked list can grow or shrink according to the requirements, so memory consumption in linked list is efficient.
- **Implementation -** We can implement both stacks and queues using linked list.

Operations On Singly Linked Lists – Traversing, Searching, Insertion, Deletion:

Self-Referential Structure:

```
struct node
{
        int data;
        struct node *next;
    }

    Struct node *head, *temp;
```

In the above declaration, we have defined a structure named as **node** that contains two variables, one is **data** that is of integer type, and another one is **next** that is a pointer which contains the address of next node.

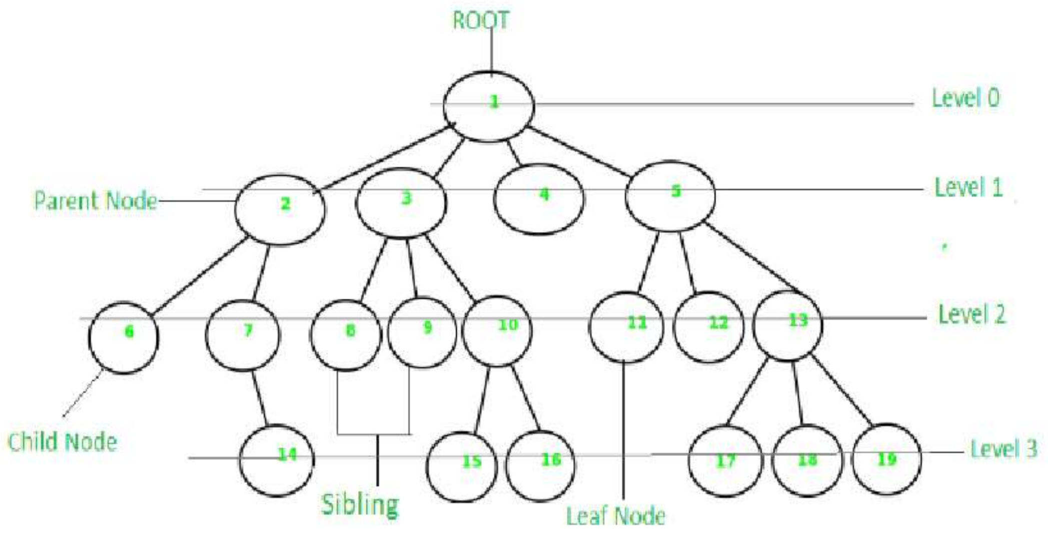
1. Creation of node:

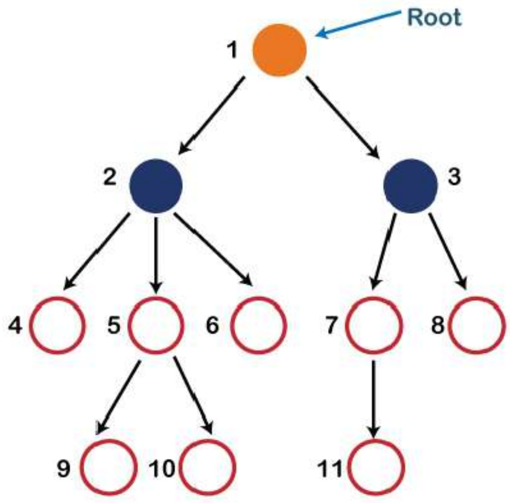
Trees: ***A tree is a nonlinear data structure, is defined as a collection of objects or entities known as nodes that are linked together to represent or simulate hierarchy.***

Features of Trees:

- 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.

Tree Terminologies –



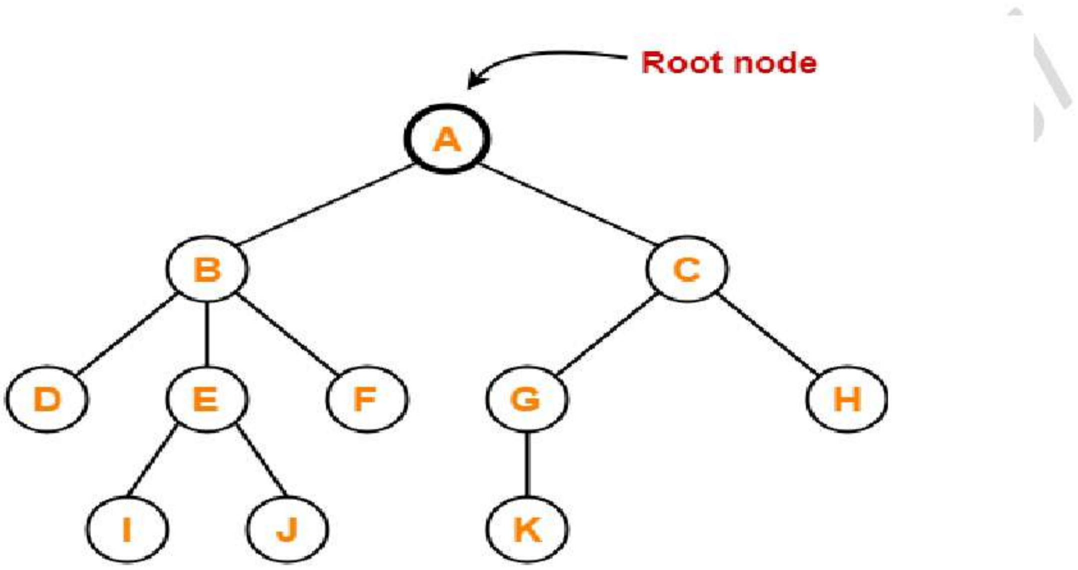


1. **Node:** ***Node is an entity of a tree which contains information and reference/link to the children.***

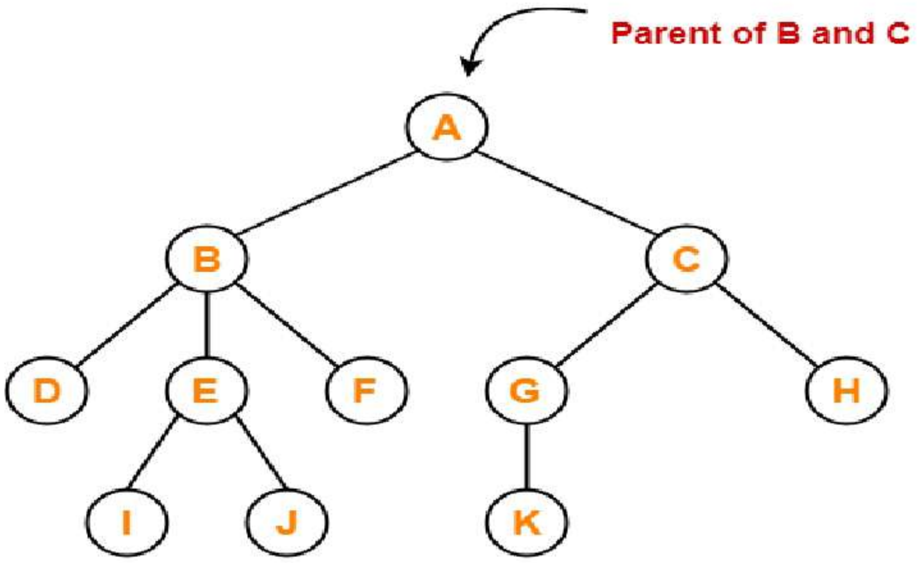
2. **Root Node:** ***The root node is the topmost node in the tree hierarchy. In other words, the root node is the one that doesn't***

have any parent. In the above structure, node numbered 1 is the root node of the tree.

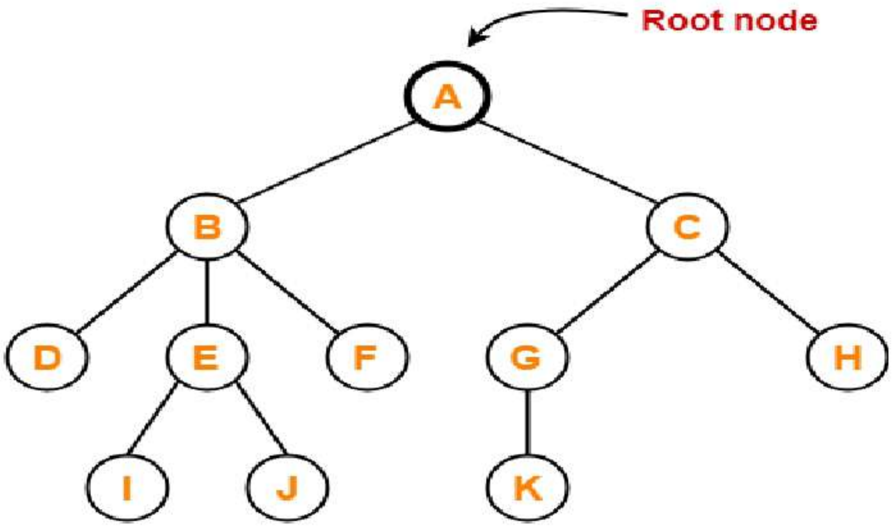
- The first node from where the tree originates is called as a **root node**.
- In any tree, there must be only one root node.
- We can never have multiple root nodes in a tree data structure.



3. Parent Node: ***If the node contains any sub-node, then that node is said to be the parent of that sub-node/children.***



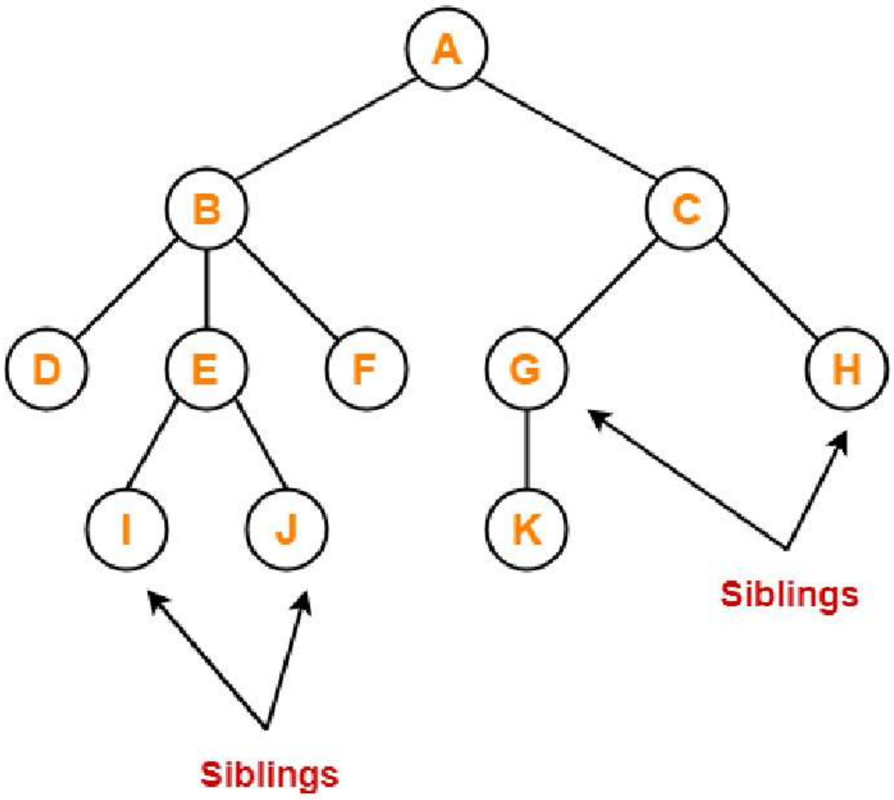
4. **Ancestors Of a Node:** ***An ancestor of a node is any predecessor node on a path from the root to that node. The root node doesn't have any ancestors.***



For J node E, B and A are ancestors.

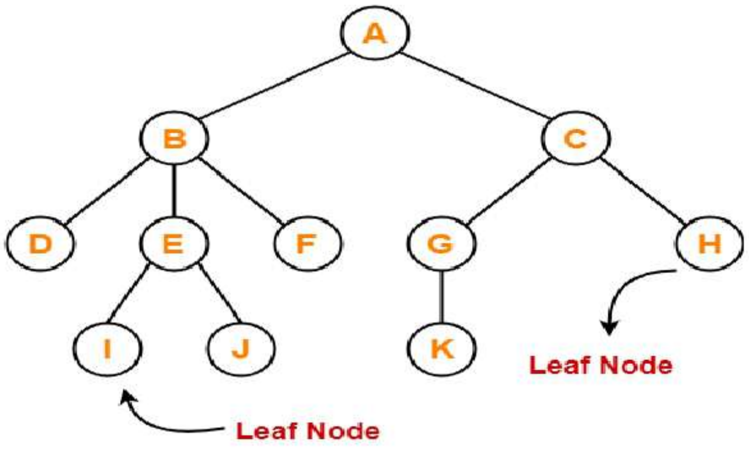
Siblings: ***The nodes that have the same parent are known as siblings.***

- Nodes which belong to the same parent are called as siblings.
- In other words, nodes with the same parent are sibling nodes.



Terminal & Non-Terminal Nodes:

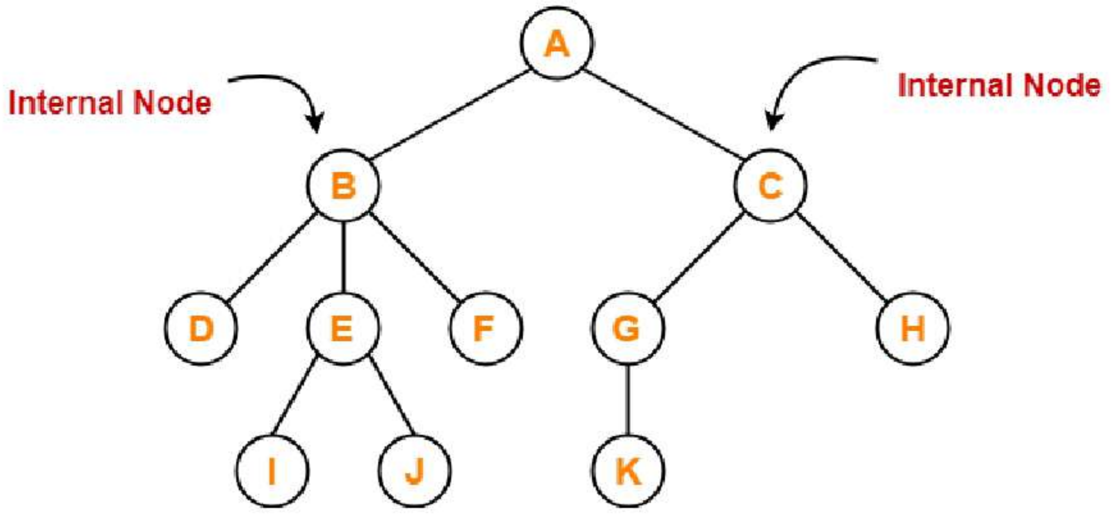
Termina Node/Leaf Node/External Node: ***A terminal node is a node like a leaf at the edge of the tree; no other nodes exist beyond it.***



Non-Terminal Node/Internal Node: ***A non-terminal node is any node that exists in the graph between other nodes.***

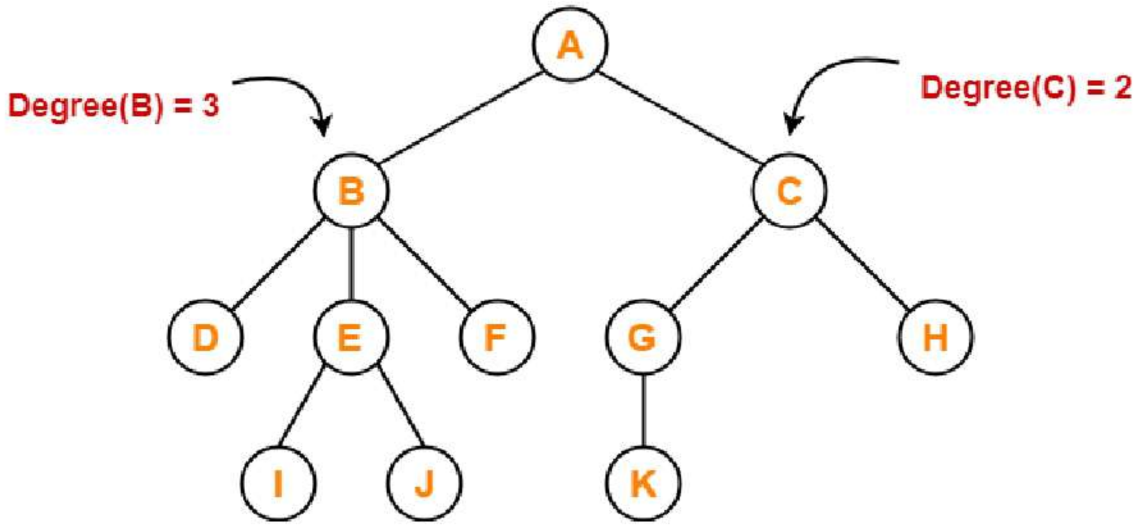
- The node which has at least one child is called as an internal node.
- Internal nodes are also called as non-terminal nodes.

- Every non-leaf node is an internal node.

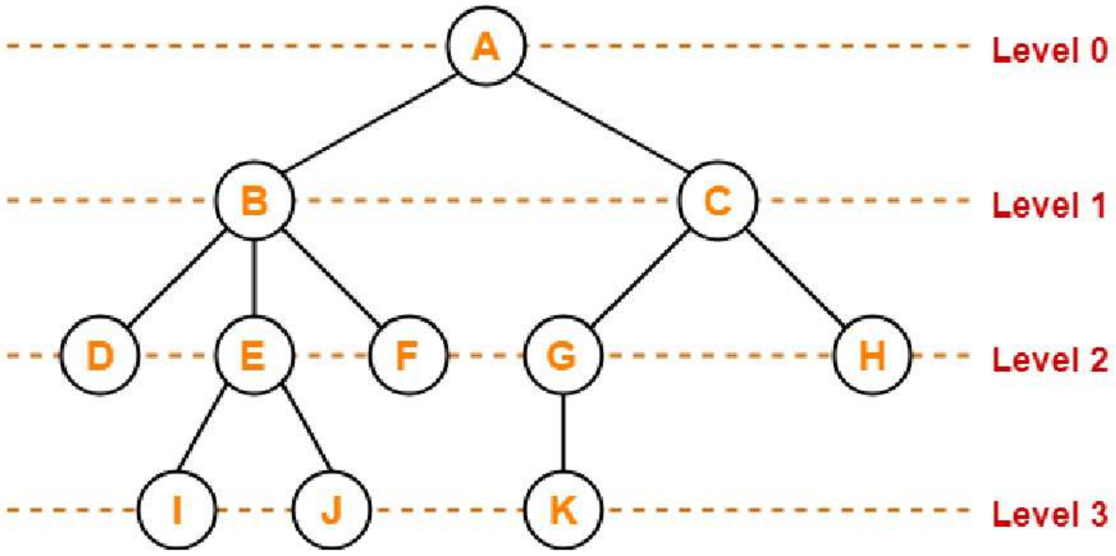


Degree Of a Node: ***Degree of a node is the total number of children of that node.***

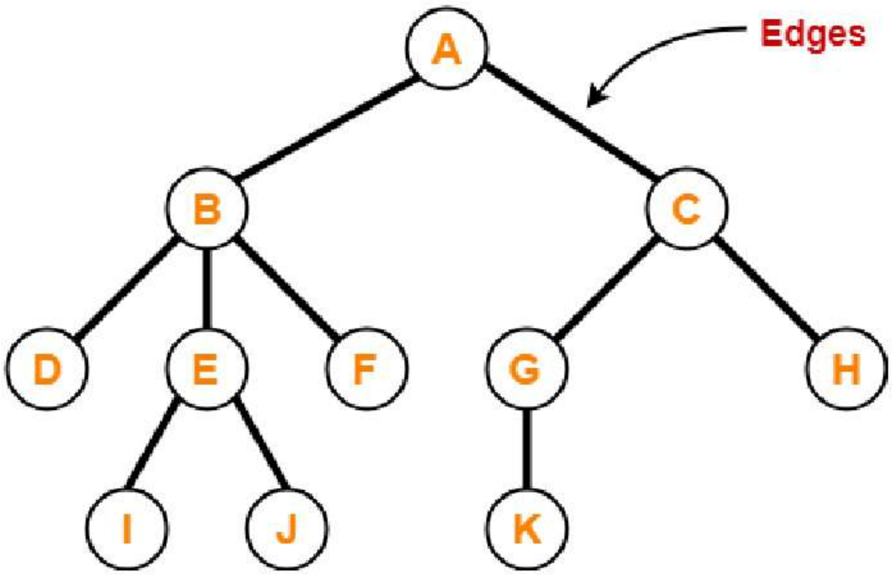
Degree of a tree: ***Is the highest degree of a node among all the nodes in the tree.***



Level: ***In a tree, each step from top to bottom is called as level of a tree. The level count starts with 0 and increments by 1 at each level or step.***



Edge: ***The connecting link between any two nodes is called as an edge. In a tree with n number of nodes, there are exactly (n-1) number of edges.***

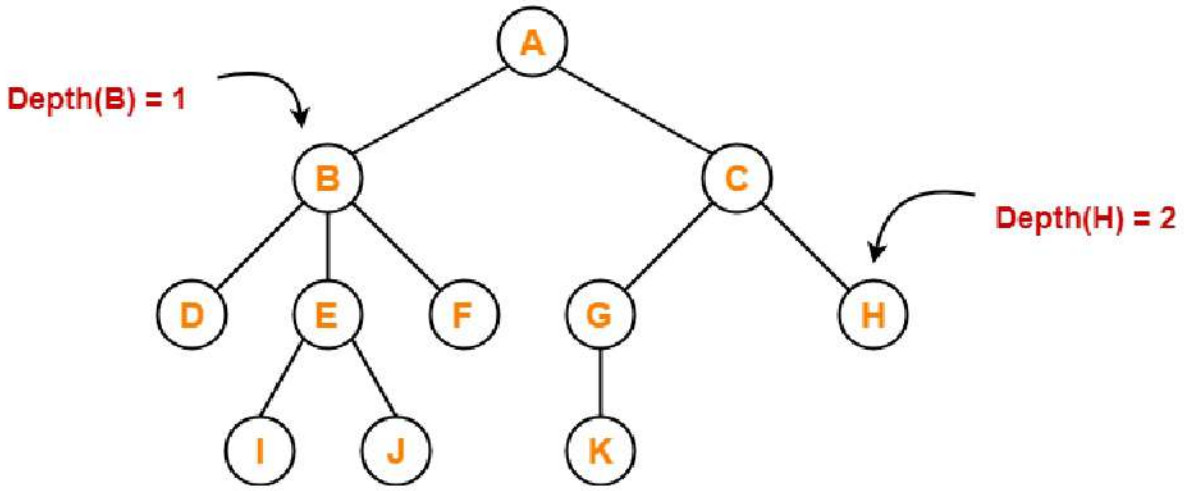


Path: ***Path refers to the sequence of nodes along the edges of a tree. Root – The node at the top of the tree is called root. There is***

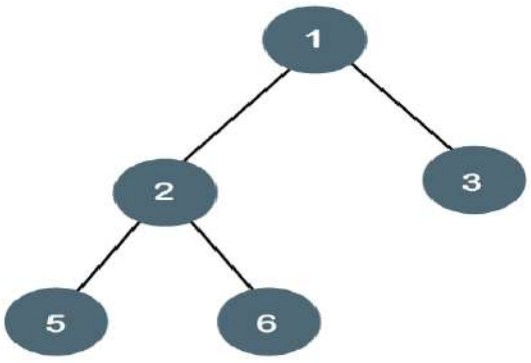
only one root per tree and one path from the root node to any node.

Depth: ***Total number of edges from root node to a particular node is called as depth of that node.***

- Total number of edges from root node to a particular node is called as depth of that node.
- Depth of a tree is the total number of edges from root node to a leaf node in the longest path.
- Depth of the root node = 0
- The terms "level" and "depth" are used interchangeably.



Binary Tree: The Binary tree means that the node can have maximum two children. Here, binary name itself suggests that 'two'; therefore, each node can have either 0, 1 or 2 children.



The above tree is a binary tree because each node contains the utmost two children.

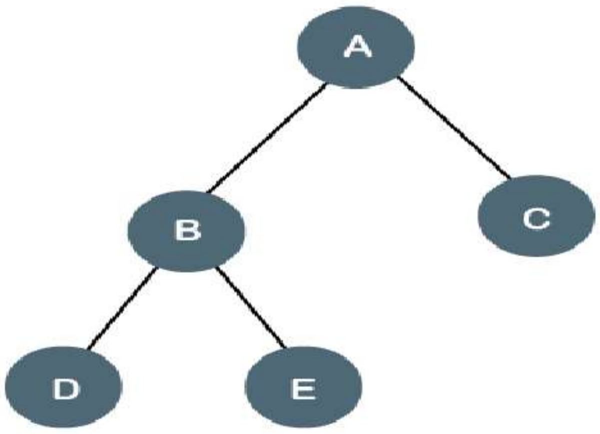
Types Of Binary Trees:

1. **Strict Binary Tree**
2. **Complete Binary Tree**
3. **Binary Search Tree**
4. **Heap Tree**

1. **Full/Proper/Strict Binary Tree:** The full binary tree is also known 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.

Example:

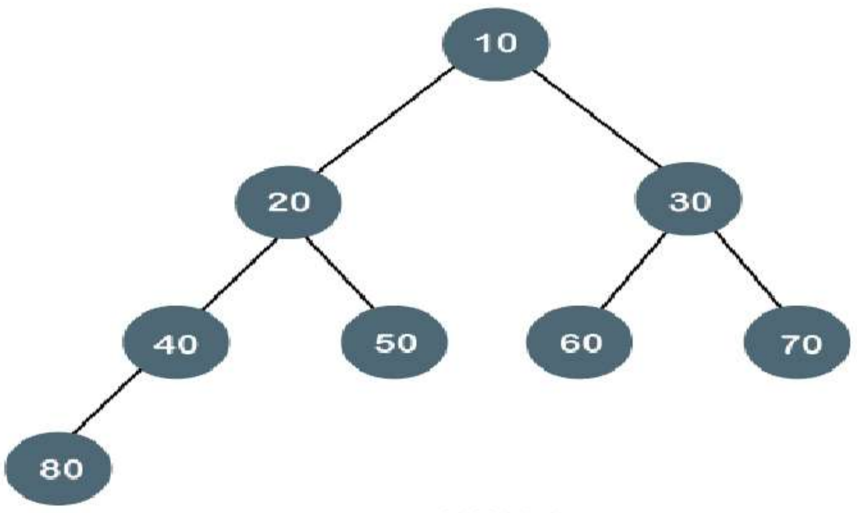


2. **Complete Binary Tree:** The complete binary tree is a tree in which all

the nodes are 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.

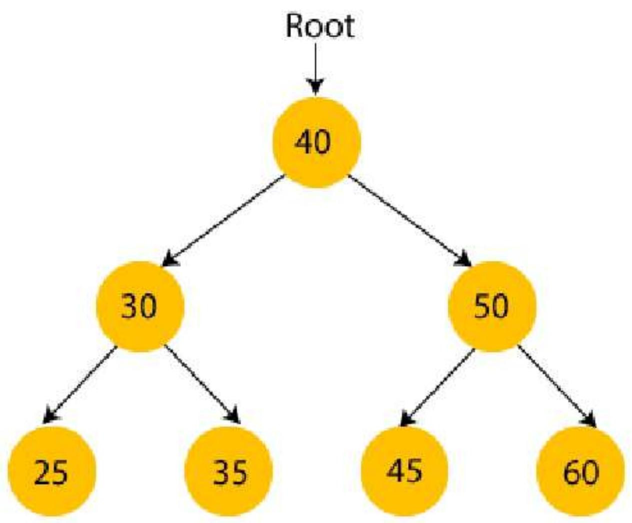
The above tree is a complete binary tree because all the nodes are filled, and all the nodes in the last level are added at the left first.

Example:

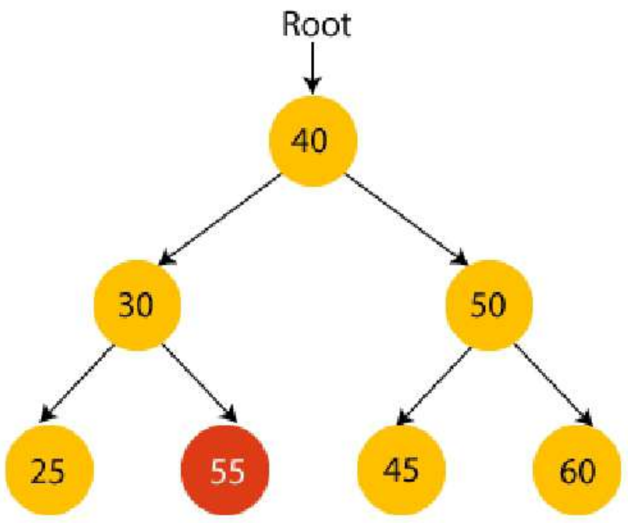


3. Binary Search Tree: A binary search tree follows some order to arrange the elements. In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node.

Example:



Example2:

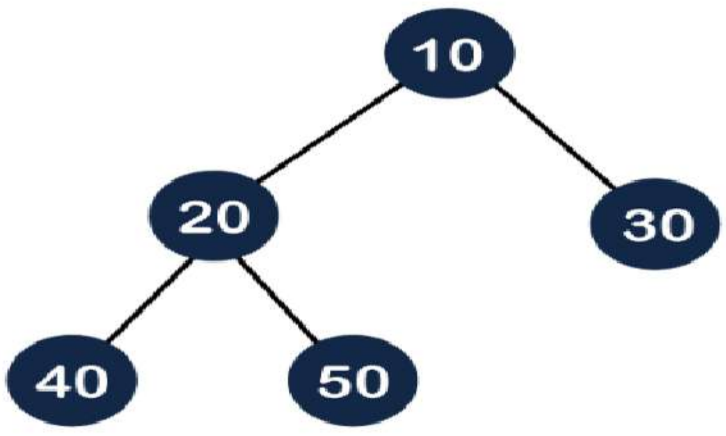


In the above tree, the value of root node is 40, which is greater than its left child 30 but smaller than right child of 30, i.e., 55. So, the above tree does not satisfy the property of Binary search tree. Therefore, the above tree is not a binary search tree.

4. Heap Tree: A Heap is a special Tree-based data structure in which the tree is a complete binary tree.

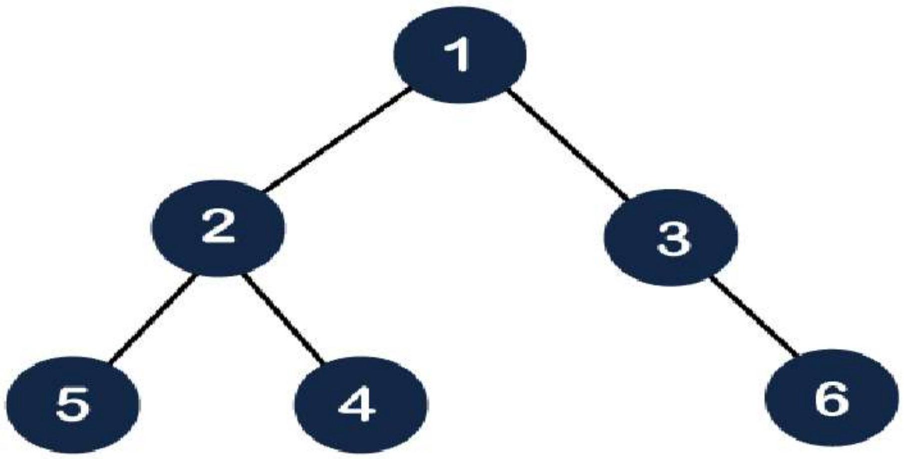
A complete binary tree is a binary tree in which all the levels except the last level, i.e., leaf node should be filled, and all the nodes should be left-justified (Left added).

Example 1: Heap Tree



In the above figure, we can observe that all the internal nodes are filled except the leaf node; therefore, we can say that the above tree is a complete binary tree.

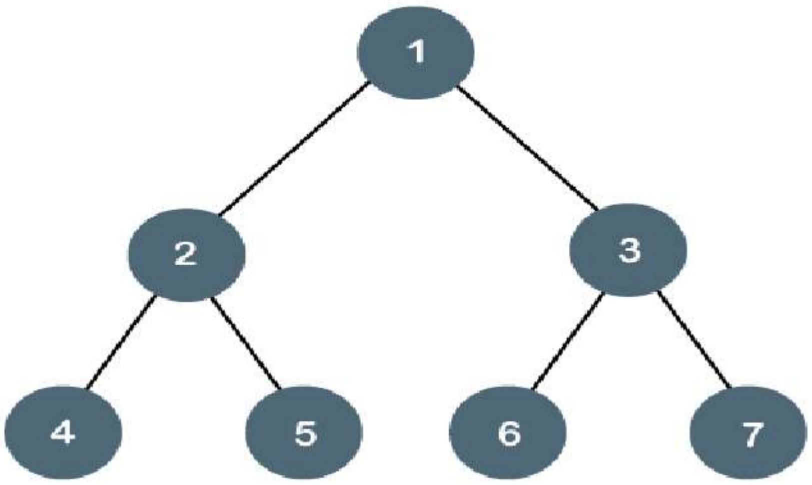
Example 2: Not Heap Tree



The above figure shows that all the internal nodes are completely filled except the leaf node, but the leaf nodes are added at the right part; therefore, the above tree is not a complete binary tree.

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.



Array Representation of Binary Tree:

Traversal Of Binary Tree: Traversal is a technique for visiting all of a tree's nodes and printing their values. Traversing a tree involves iterating over all nodes in some manner.

We always start from the root (head) node since all nodes are connected by edges (links).

As the tree is not a linear data structure, there can be more than one possible next node from a given node, so some nodes must be deferred, i.e., stored in some way for later visiting.

Types of Traversal of Binary Tree

There are three types of traversals of a binary tree.

1. Inorder tree traversal
2. Preorder tree traversal
3. Postorder tree traversal

Preorder: In this traversal method, the root node is visited first, then the left subtree, and finally the right subtree.

In Order:

Post Order Traversal: