Data Structures

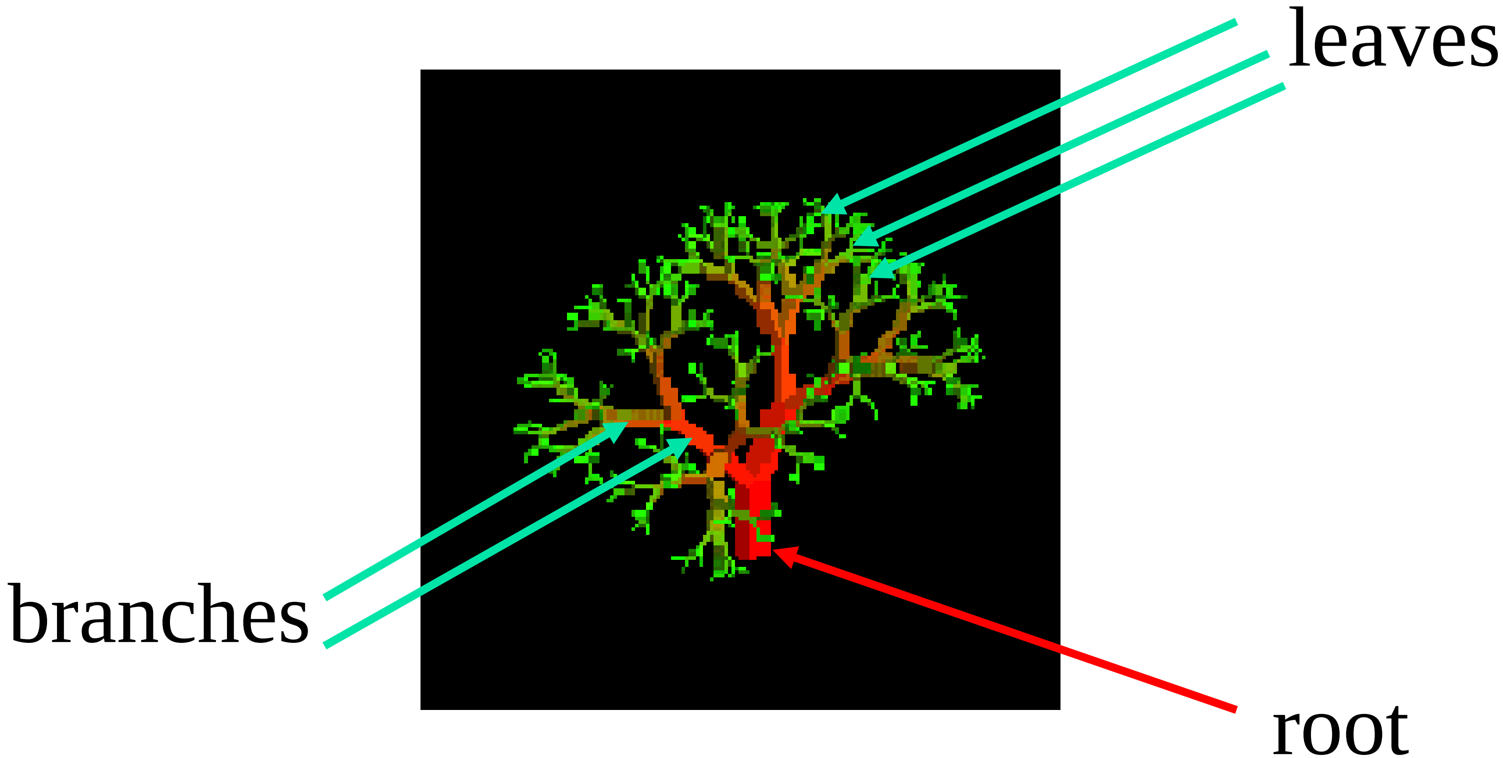
IT 205

Dr. Manish Khare

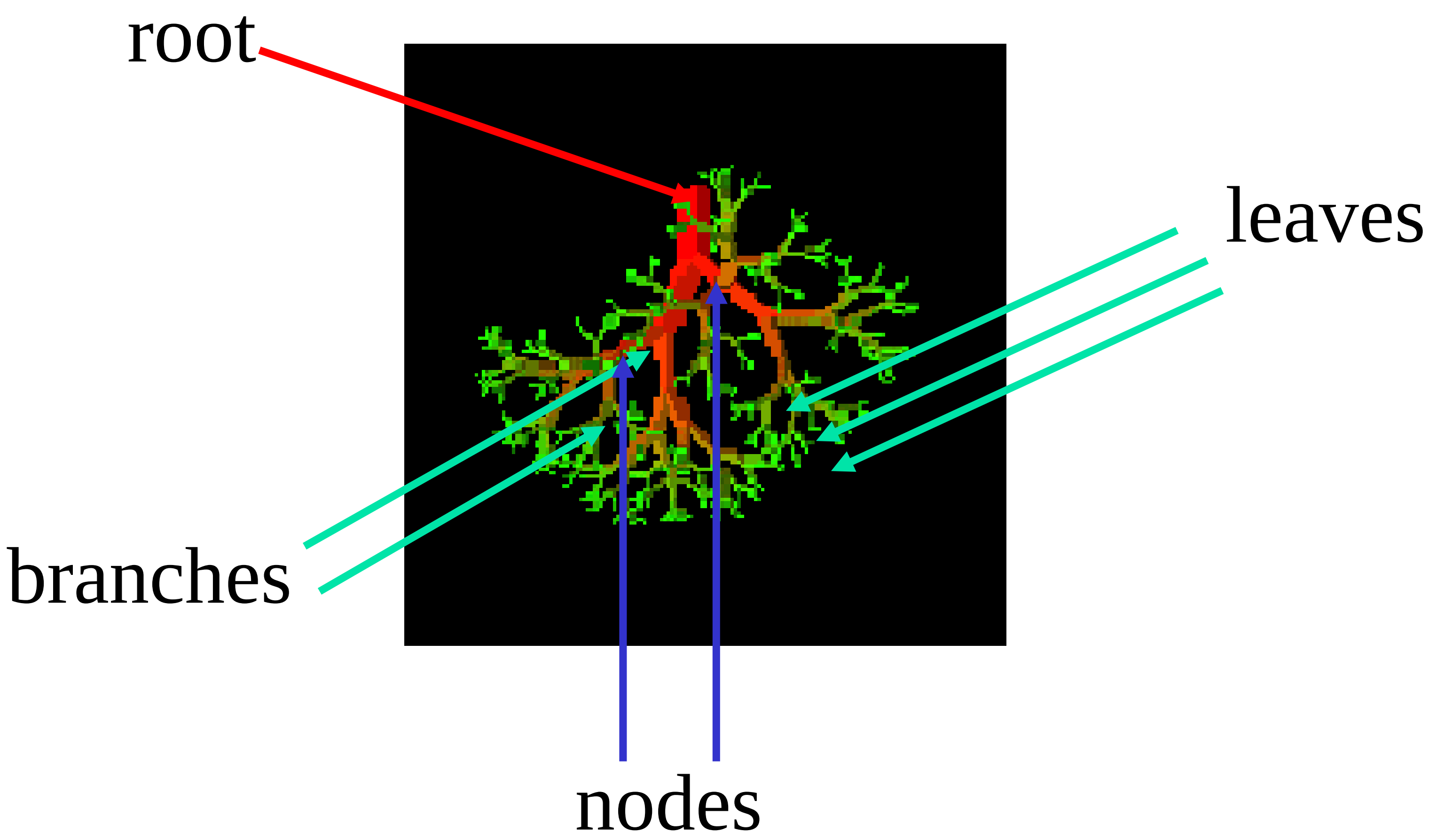
Lecture – 15&16
15-Feb-2018

Tree

Nature View of a Tree



Computer Scientist's View



Tree

- In linear data structure, data is organized in sequential order and in non-linear data structure, data is organized in random order.
- Tree is a very popular data structure used in wide range of applications. A tree data structure can be defined as follows...

- **Tree is a non-linear data structure which organizes data in hierarchical structure and this is a recursive definition.**

- **Tree data structure is a collection of data (Node) which is organized in hierarchical structure and this is a recursive definition**

Definition of Tree

- A tree is a finite set of one or more nodes such that:

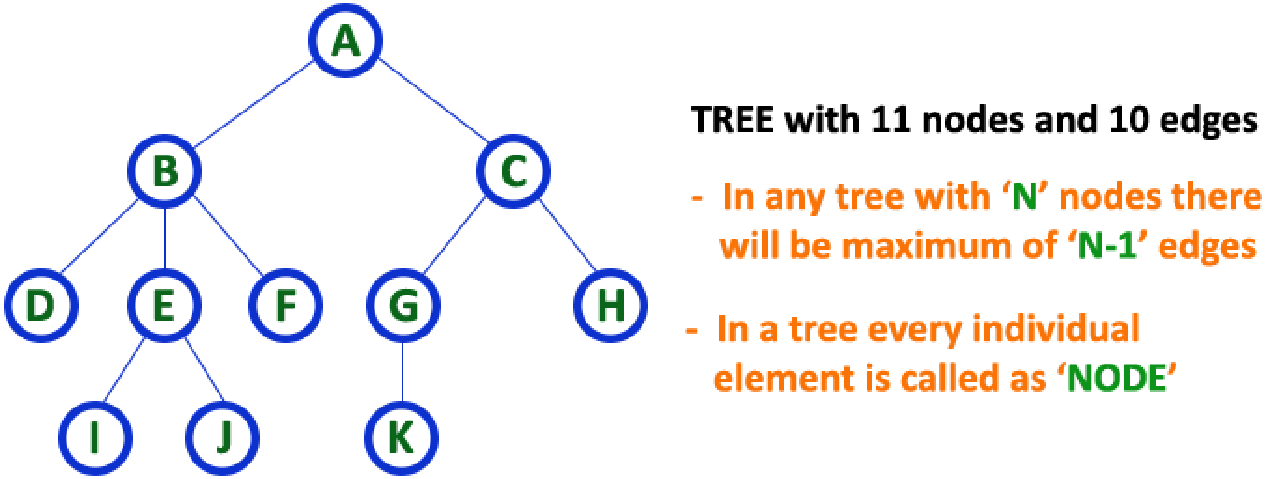
  - There is a specially designated node called the root.

  - The remaining nodes are partitioned into n>=0 disjoint sets T_1, ..., T_n, where each of these sets is a tree. We call T_1, ..., T_n the subtrees of the root.

Tree Terminology

- In tree data structure, every individual element is called as **Node**. Node in a tree data structure, stores the actual data of that particular element and link to next element in hierarchical structure.

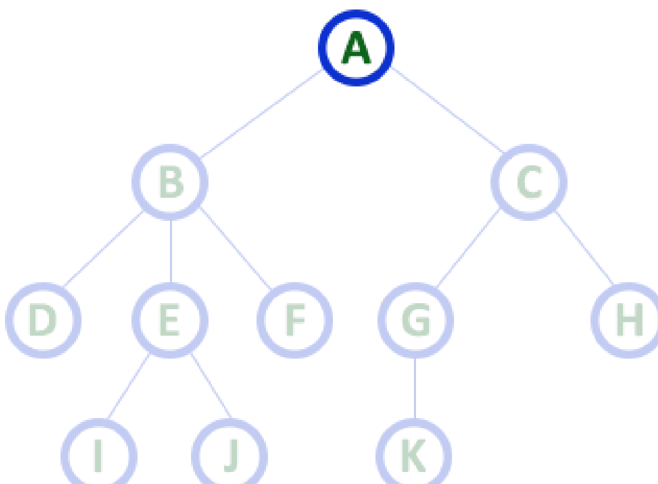
- In a tree data structure, if we have **N** number of nodes then we can have a maximum of **N-1** number of links.



Tree Terminology

➢ Root

In a tree data structure, the first node is called as **Root Node**. Every tree must have root node. We can say that root node is the origin of tree data structure. In any tree, there must be only one root node. We never have multiple root nodes in a tree.



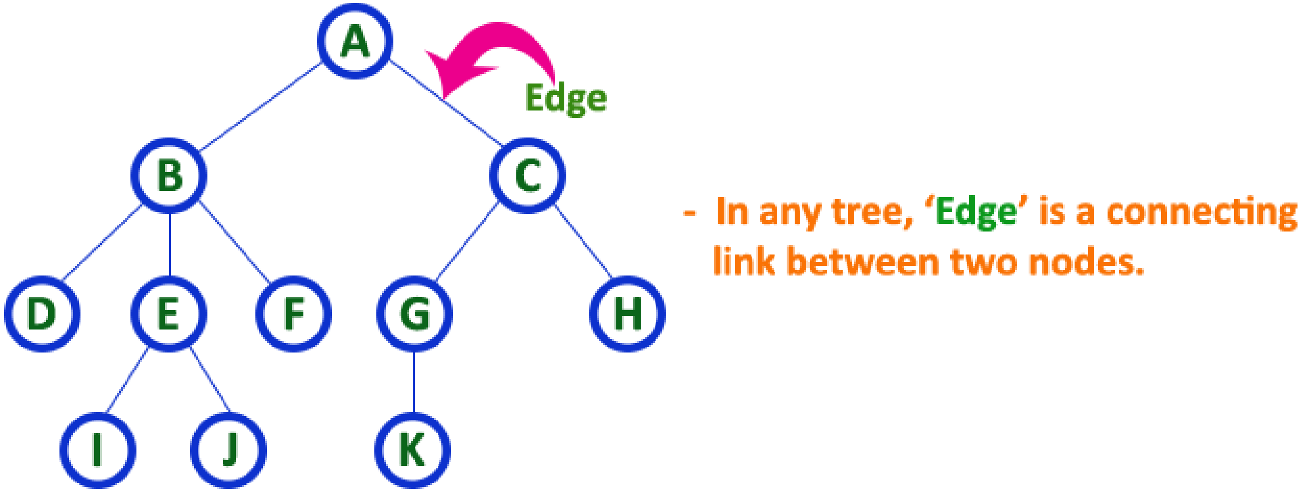
Here 'A' is the 'root' node

- In any tree the first node is called as ROOT node

Tree Terminology

➢ Edge

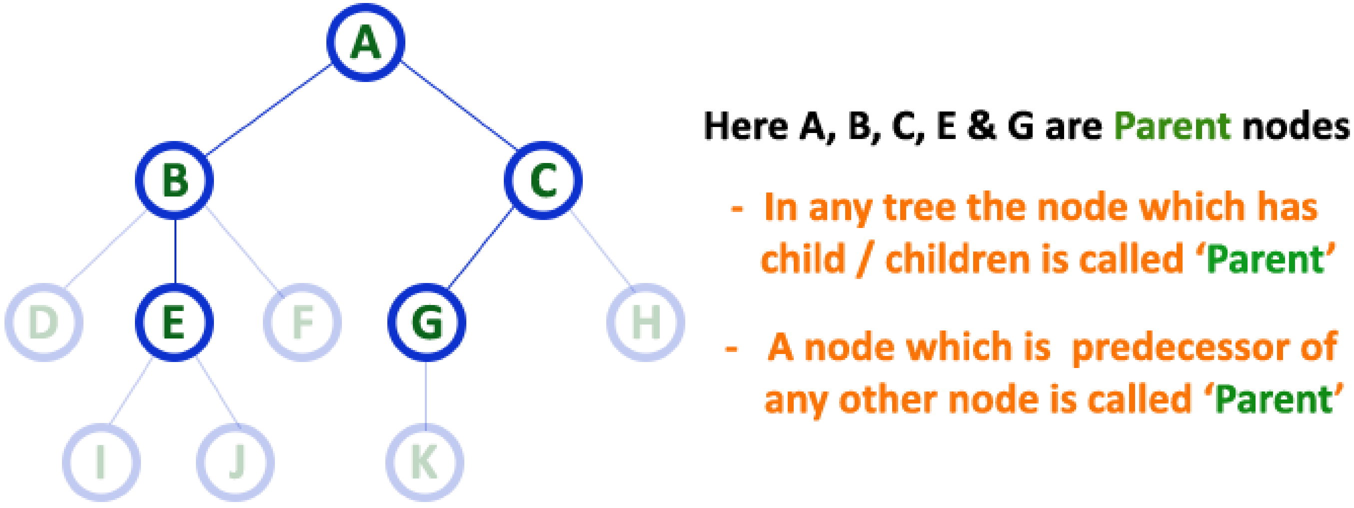
In a tree data structure, the connecting link between any two nodes is called as **EDGE**. In a tree with '**N**' number of nodes there will be a maximum of '**N-1**' number of edges.



Tree Terminology

Parent

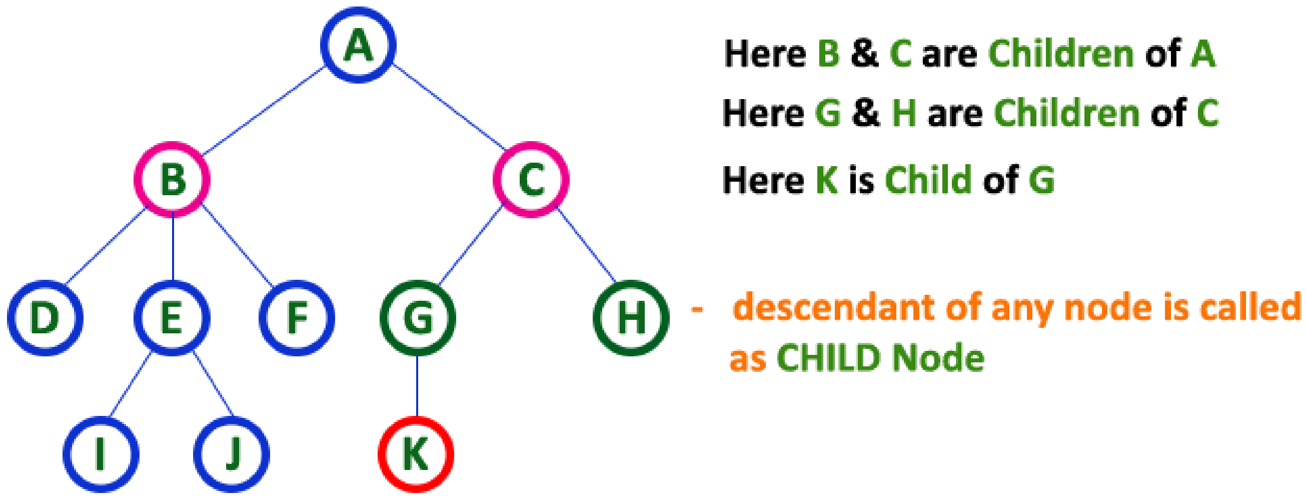
In a tree data structure, the node which is predecessor of any node is called as **PARENT NODE**. In simple words, the node which has branch from it to any other node is called as parent node. Parent node can also be defined as "**The node which has child / children**".



Tree Terminology

➢ Child

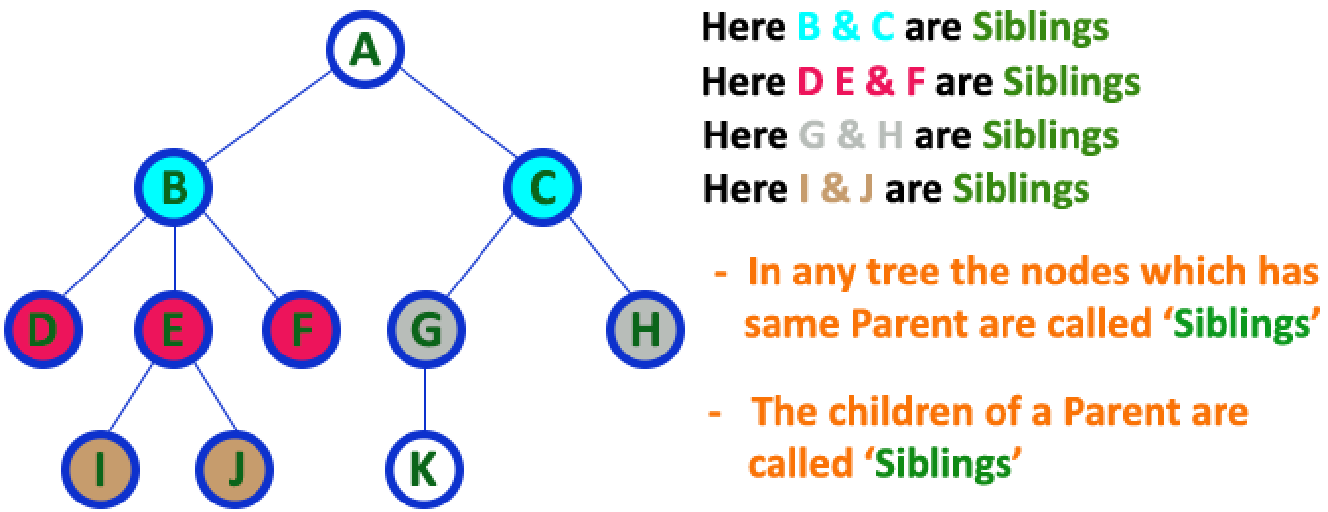
In a tree data structure, the node which is descendant of any node is called as **CHILD Node**. In simple words, the node which has a link from its parent node is called as child node. In a tree, any parent node can have any number of child nodes. In a tree, all the nodes except root are child nodes.



Tree Terminology

➢ Siblings

In a tree data structure, nodes which belong to same Parent are called as **SIBLINGS**. In simple words, the nodes with same parent are called as Sibling nodes.

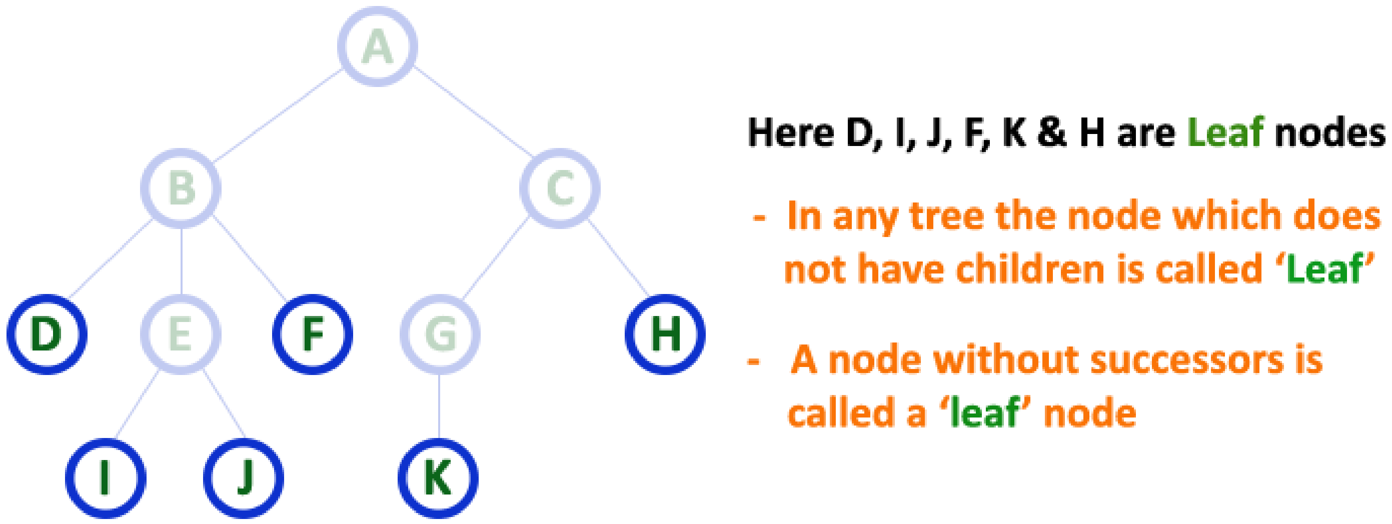


Tree Terminology

➢ Leaf

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.

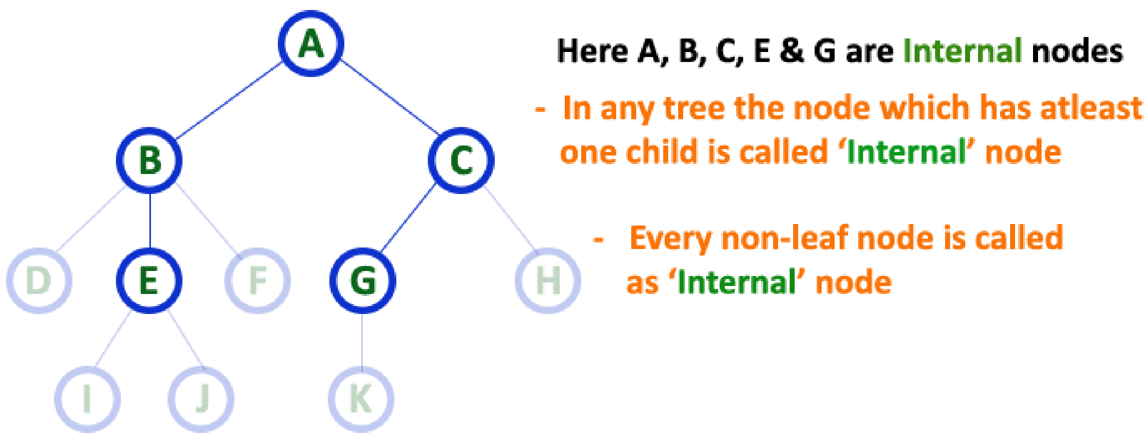


Tree Terminology

➢ Internal Nodes

In a tree data structure, the node which has atleast one child is called as **INTERNAL Node**. In simple words, an internal node is a node with atleast one child.

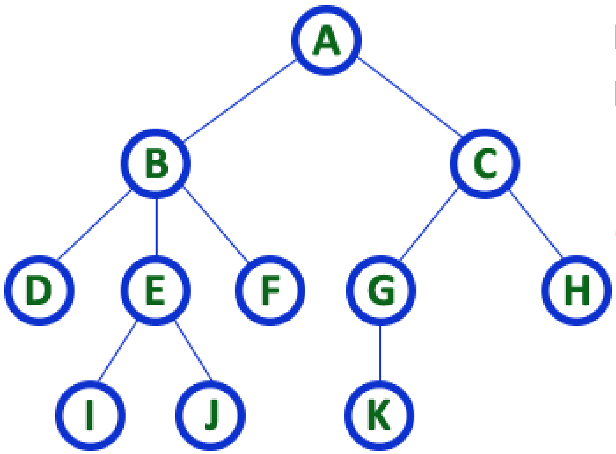
In a tree data structure, nodes other than leaf nodes are called as **Internal Nodes**. **The root node is also said to be Internal Node** if the tree has more than one node. Internal nodes are also called as '**Non-Terminal**' nodes.



Tree Terminology

➢ 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 of a node among all the nodes in a tree is called as '**Degree of Tree**'



Here Degree of B is 3

Here Degree of A is 2

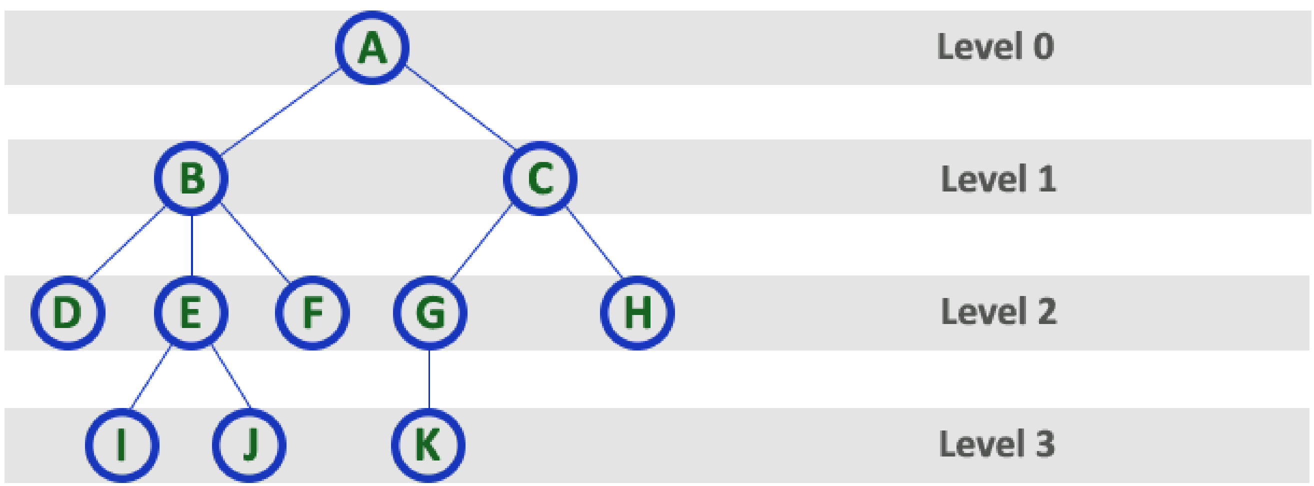
Here Degree of F is 0

- In any tree, 'Degree' a node is total number of children it has.

Tree Terminology

➢ Level

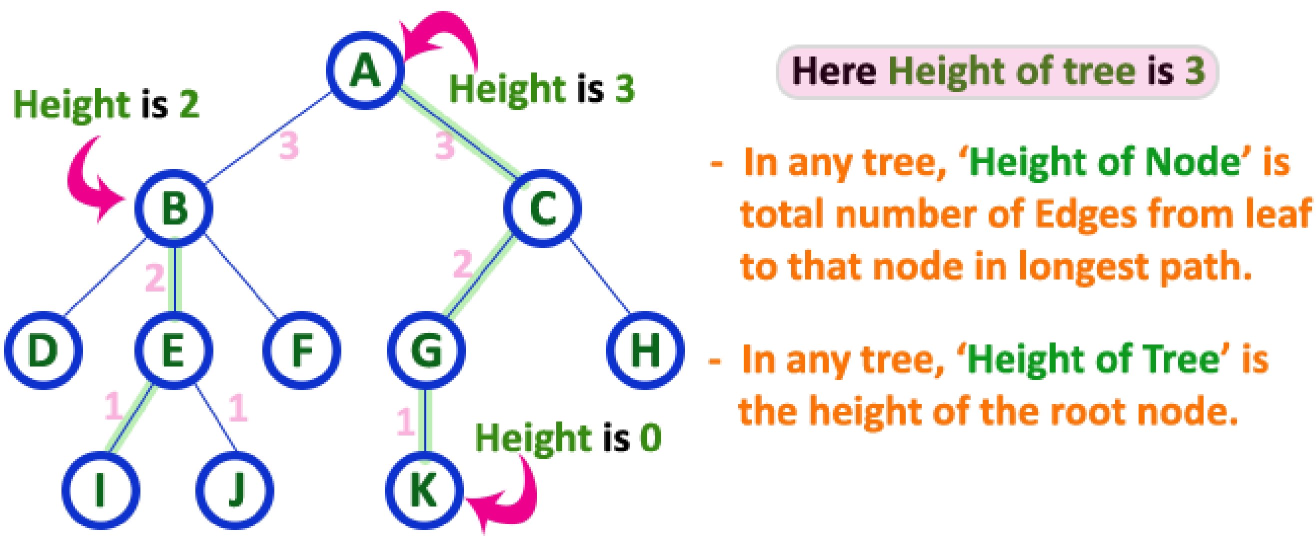
In a tree data structure, the root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2 and so on... In simple words, in a tree each step from top to bottom is called as a Level and the Level count starts with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



Tree Terminology

➤ Height

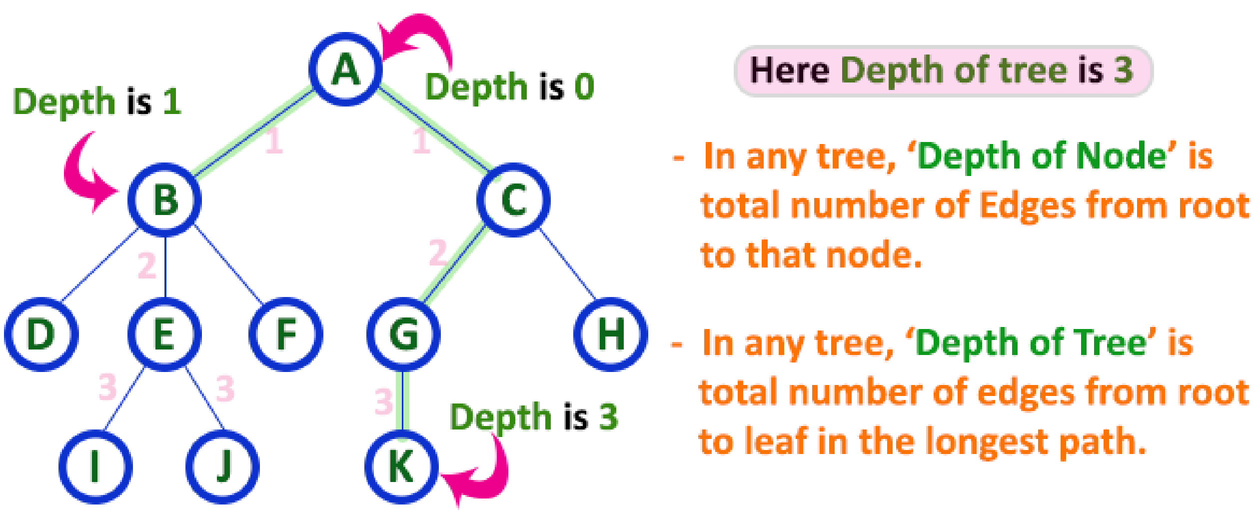
In a tree data structure, the total number of egdes from leaf node to a particular node in the longest path is called as **HEIGHT** of that Node. <u>In a tree, height of the root node is said to be **height of the tree**</u>. In a tree, <u>**height of all leaf nodes is '0'.**</u>



Tree Terminology

➢ Depth

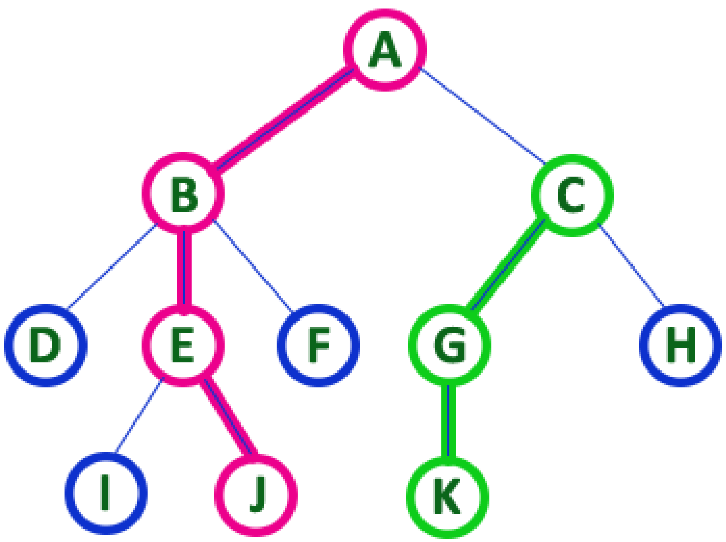
In a tree data structure, the total number of edges from root node to a particular node is called as **DEPTH** of that Node. <u>In a tree, the total number of edges from root node to a leaf node in the longest path is said to be **Depth of the tree**</u>. In simple words, the highest depth of any leaf node in a tree is said to be depth of that tree. In a tree, <u>**depth of the root node is '0'.**</u>



Tree Terminology

➢ 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. **<u>Length of a Path</u>** <u>is total number of nodes in that path.</u> In below example **the path A - B - E - J has length 4**.



- In any tree, 'Path' is a sequence of nodes and edges between two nodes.

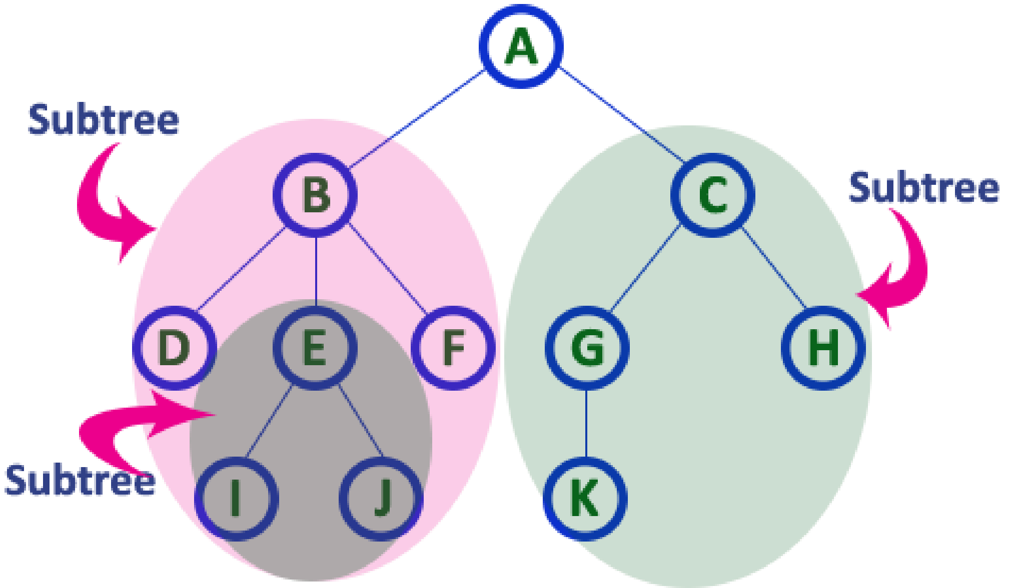
Here, 'Path' between A & J is
A - B - E - J

Here, 'Path' between C & K is
C - G - K

Tree Terminology

➢ 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.



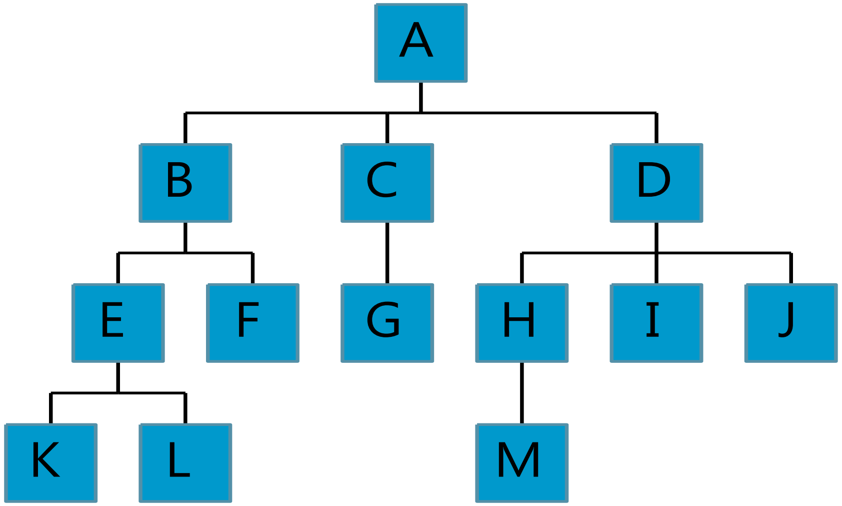
Tree Representations

- A tree data structure can be represented in two methods.

- Those methods are as follows...
  - **List Representation**
  - **Left Child - Right Sibling Representation**
  - **Representation as a degree-Two Tree**

Tree Representations

Consider the following tree...



List Representation

- In this representation, we use two types of nodes one for representing the node with data and another for representing only references. We start with a node with data from root node in the tree. Then it is linked to an internal node through a reference node and is linked to any other node directly. This process repeats for all the nodes in the tree.

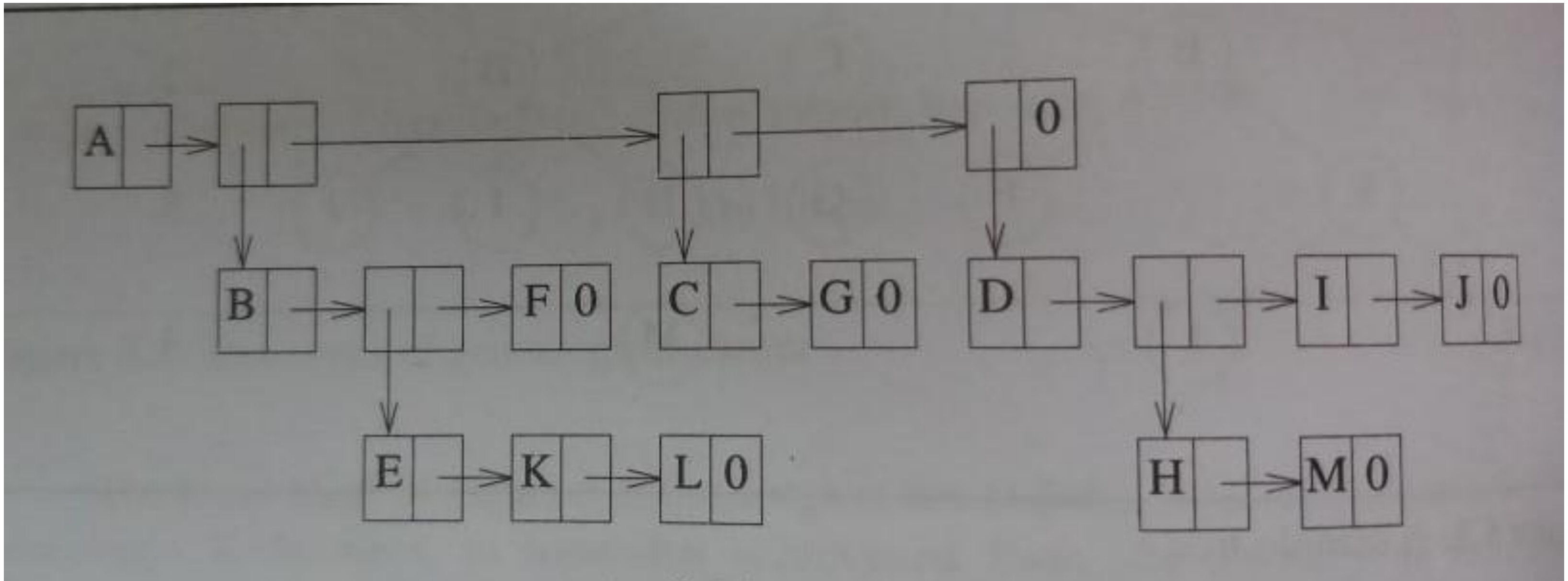
- The tree of figure could be written as the list

(A (B (E (K, L), F), C (G), D (H (M), I, J)))

data	link 1	link 2	...	link n

How many link fields are needed in such a representation?

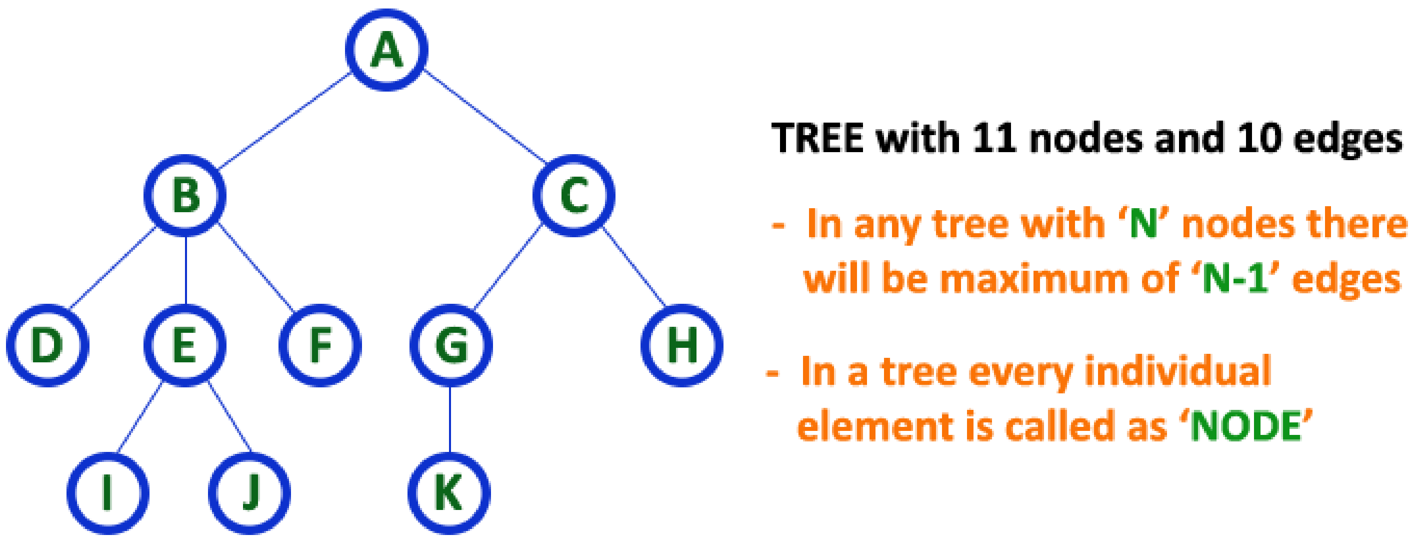
List Representation



tag fields not shown

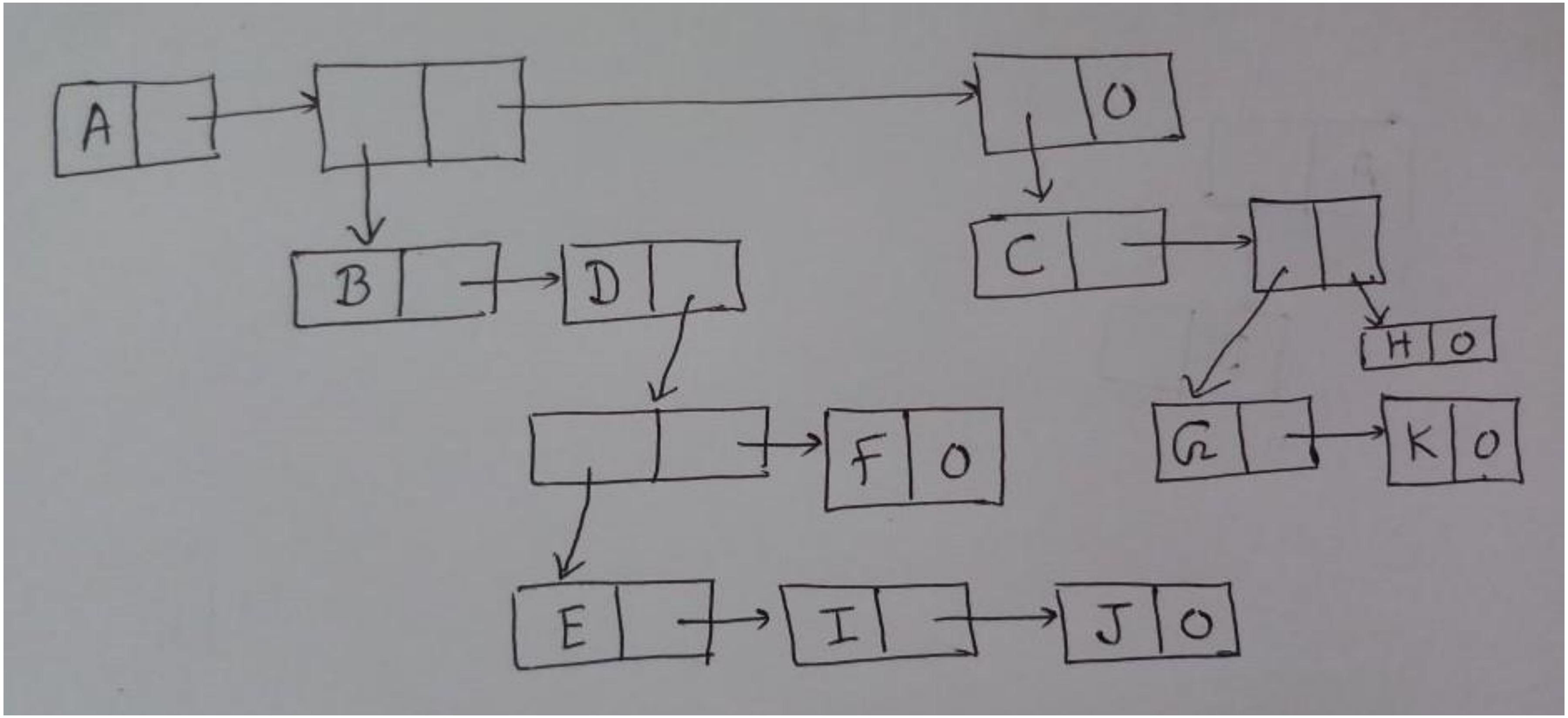
List Representation

- Consider following tree



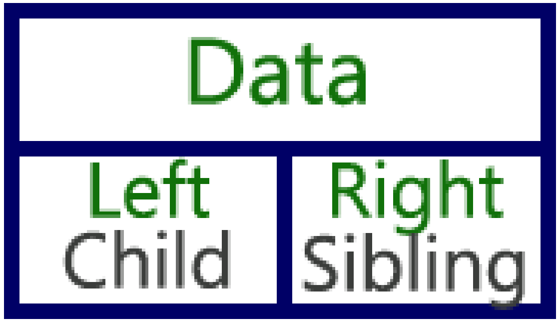
(A(B(D,E(I,J),F),C(G(K),H)))

List Representation



Left Child - Right Sibling Representation

- In this representation, we use list with one type of node which consists of three fields namely Data field, Left child reference field and Right sibling reference field.
- Data field stores the actual value of a node, left reference field stores the address of the left child and right reference field stores the address of the right sibling node.
- Graphical representation of that node is as follows...

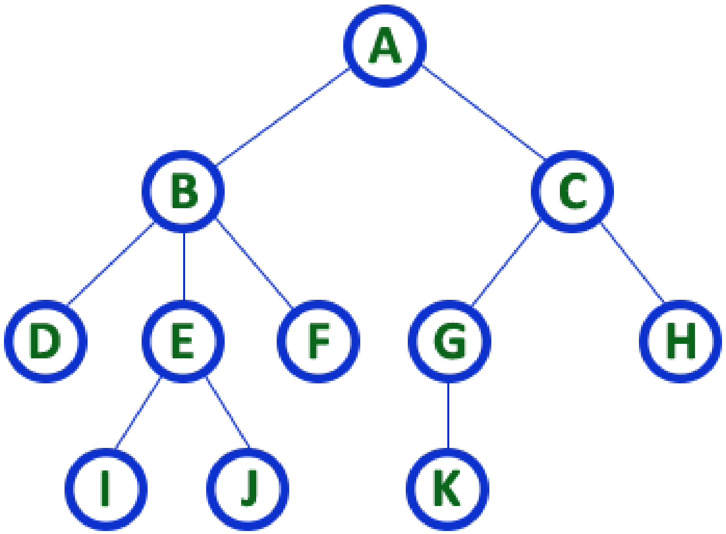


Left Child - Right Sibling Representation

- In this representation, every node's data field stores the actual value of that node.
  - If that node has left child, then left reference field stores the address of that left child node otherwise that field stores NULL.
  - If that node has right sibling then right reference field stores the address of right sibling node otherwise that field stores NULL.

Left Child - Right Sibling Representation

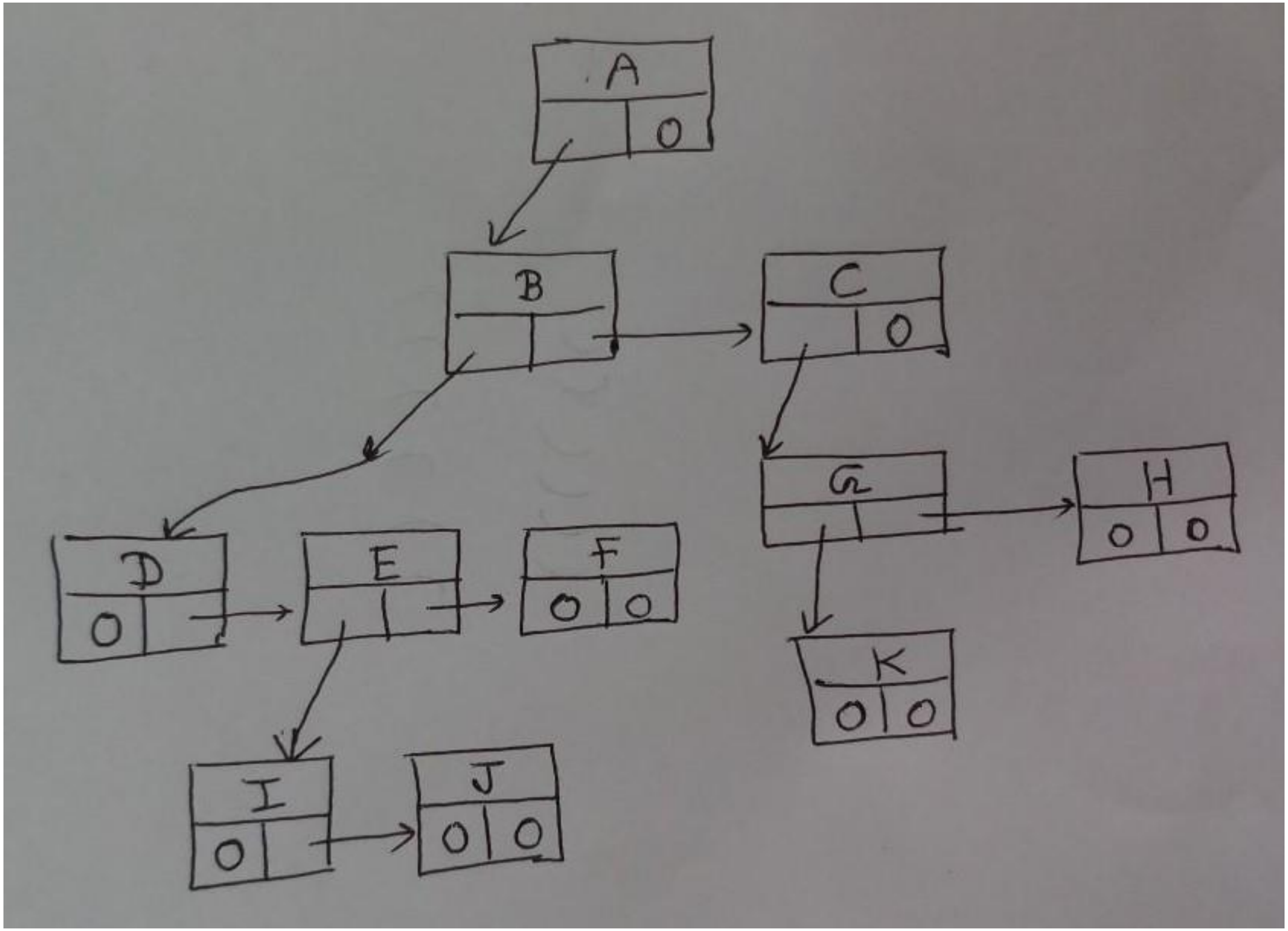
- Consider following tree



TREE with 11 nodes and 10 edges

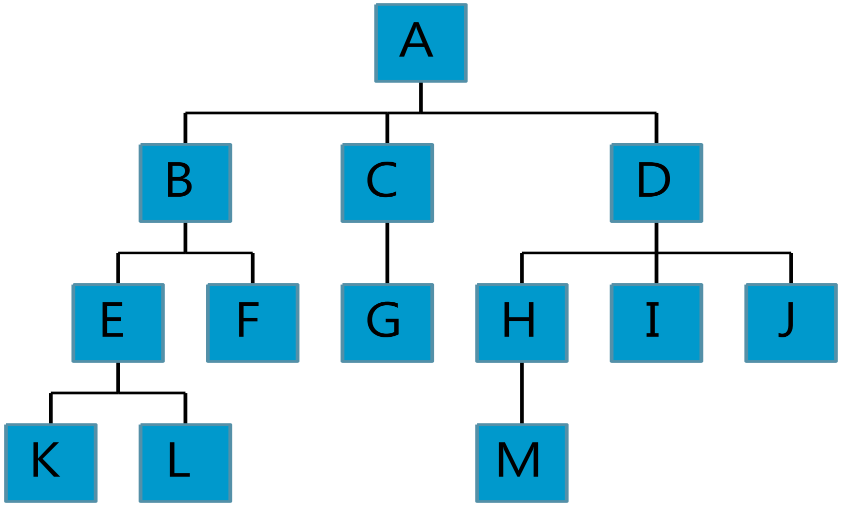
- In any tree with 'N' nodes there will be maximum of 'N-1' edges
- In a tree every individual element is called as 'NODE'

Left Child - Right Sibling Representation

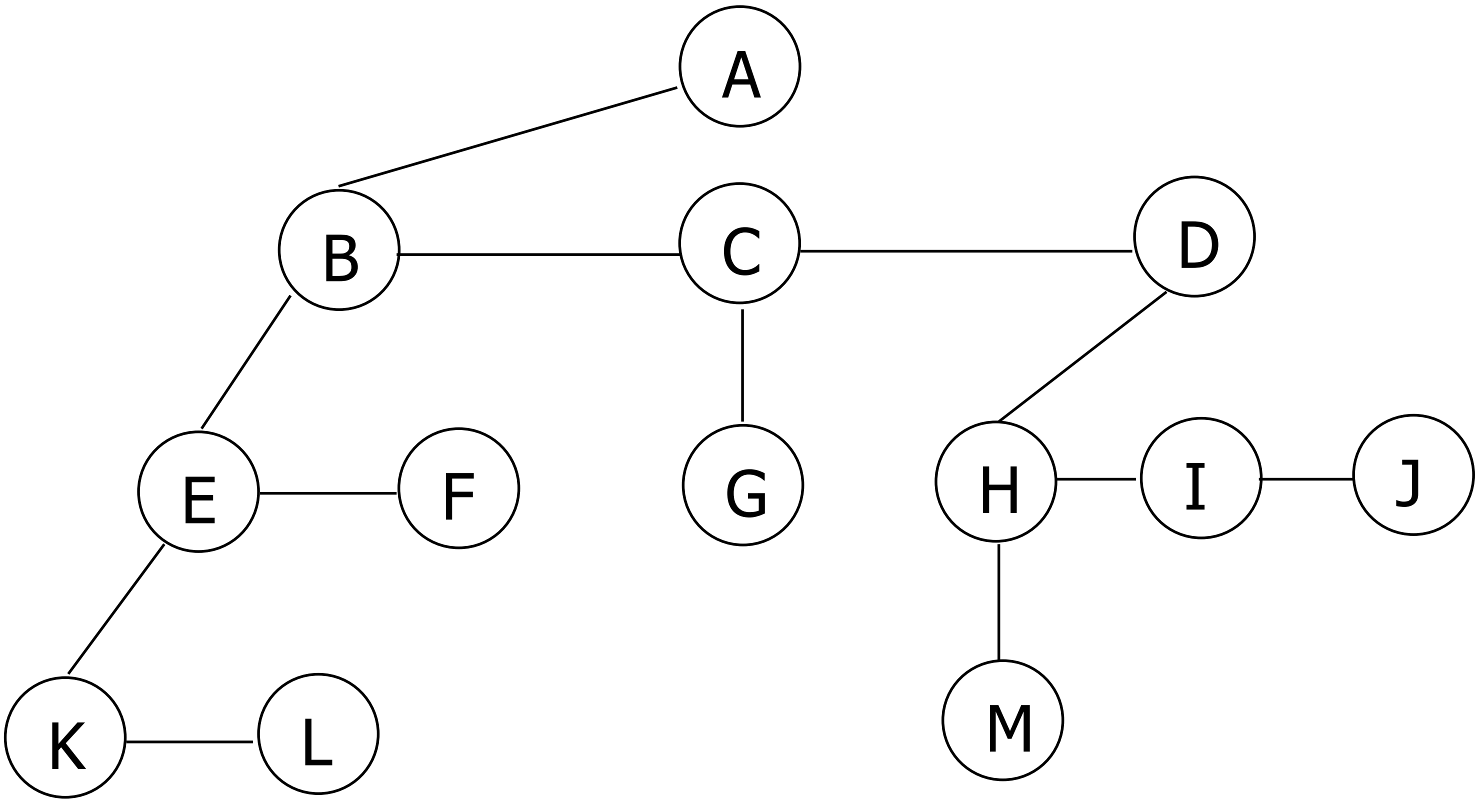


Left Child - Right Sibling Representation

Consider the following tree...

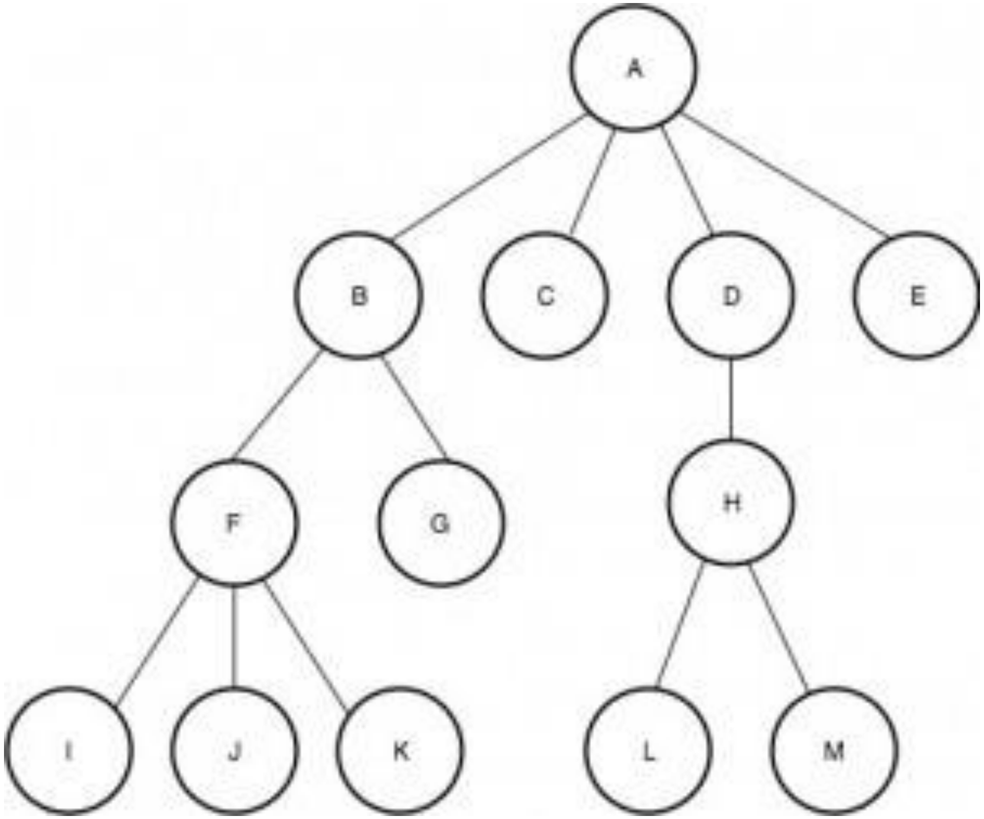


Left Child - Right Sibling Representation

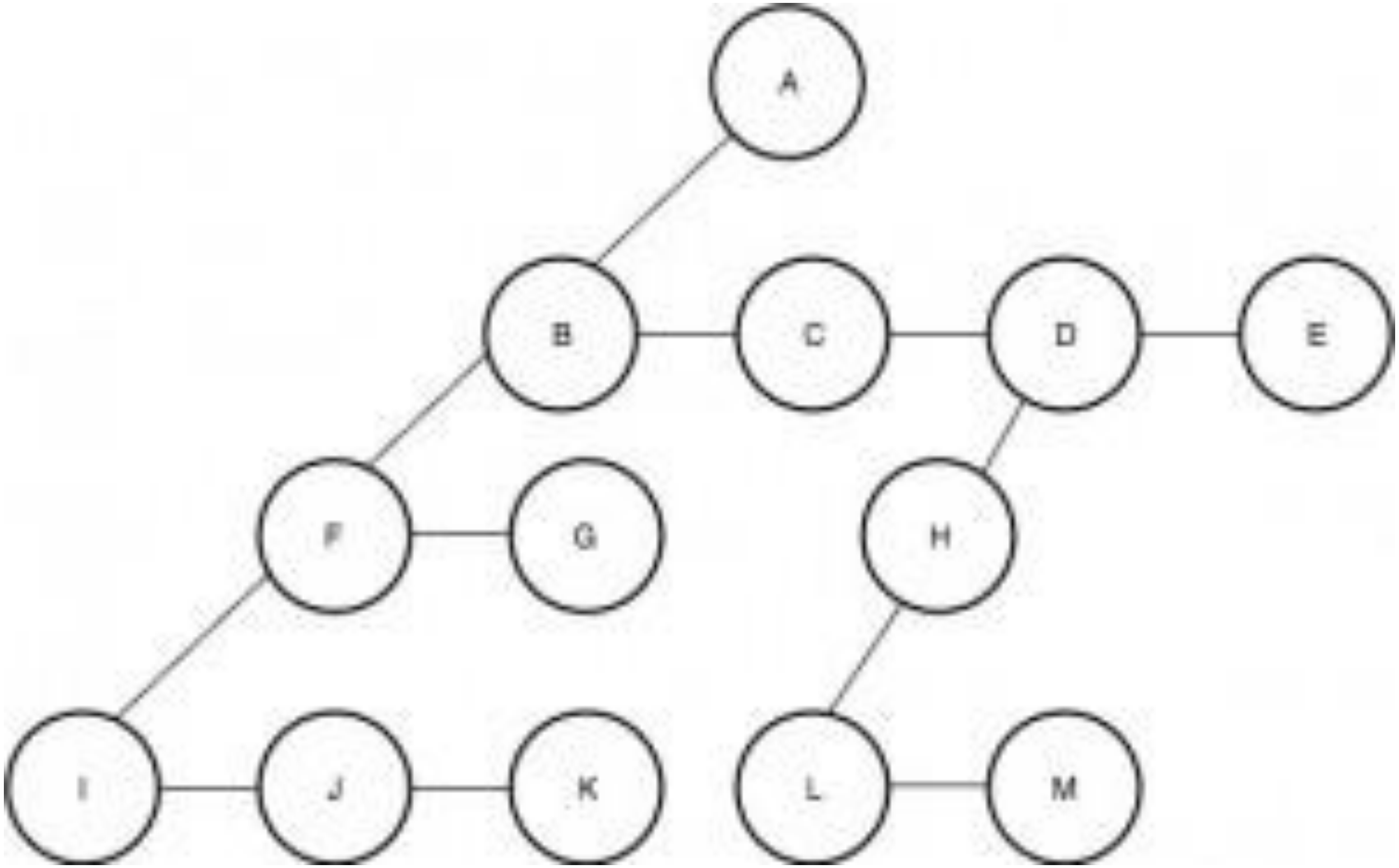


Left Child - Right Sibling Representation

Consider the following tree...



Left Child - Right Sibling Representation

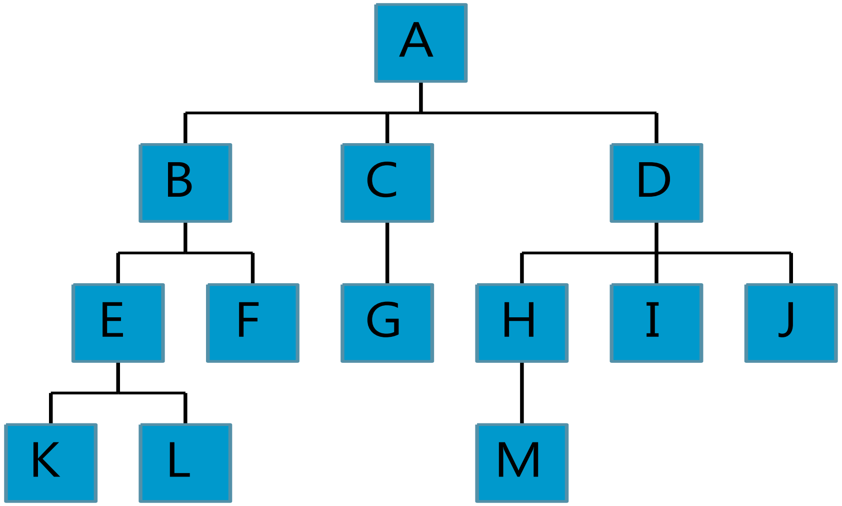


Representation as a Degree-Two Tree

- To obtain the degree-two tree representation of a tree, we simply rotate the right-sibling pointers in a left child-right sibling tree clockwise by 45 degrees.
- In the degree-two representation, we refer to the two children of a node as the left and right children.
- Notice that the right child of the root node of the tree is empty. This is always the case since the root of the tree we are transforming can never have a sibling.
- This type of representation are also known as binary tree.

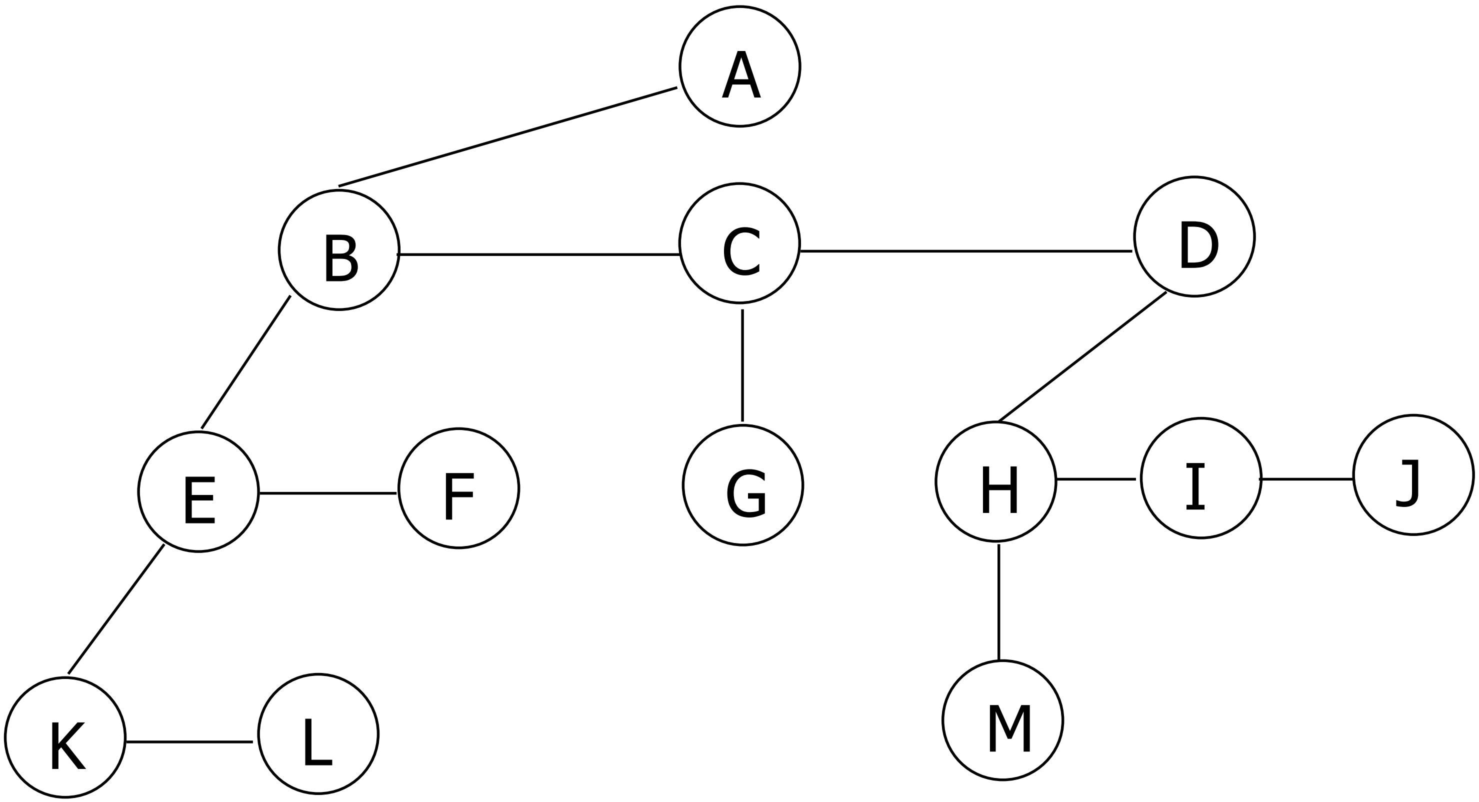
Representation as a Degree-Two Tree

Consider the following tree...



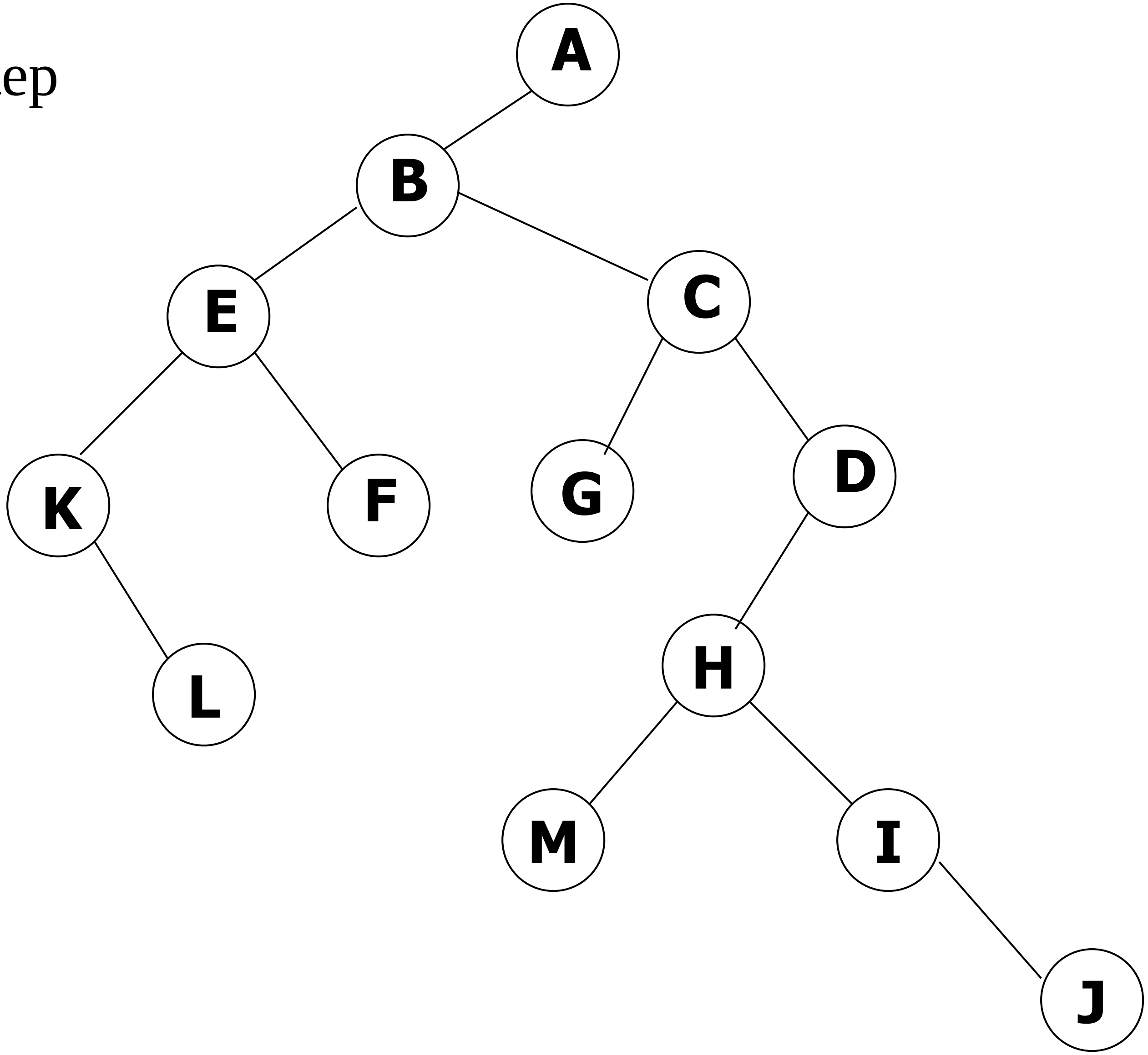
Representation as a Degree-Two Tree

- First Step

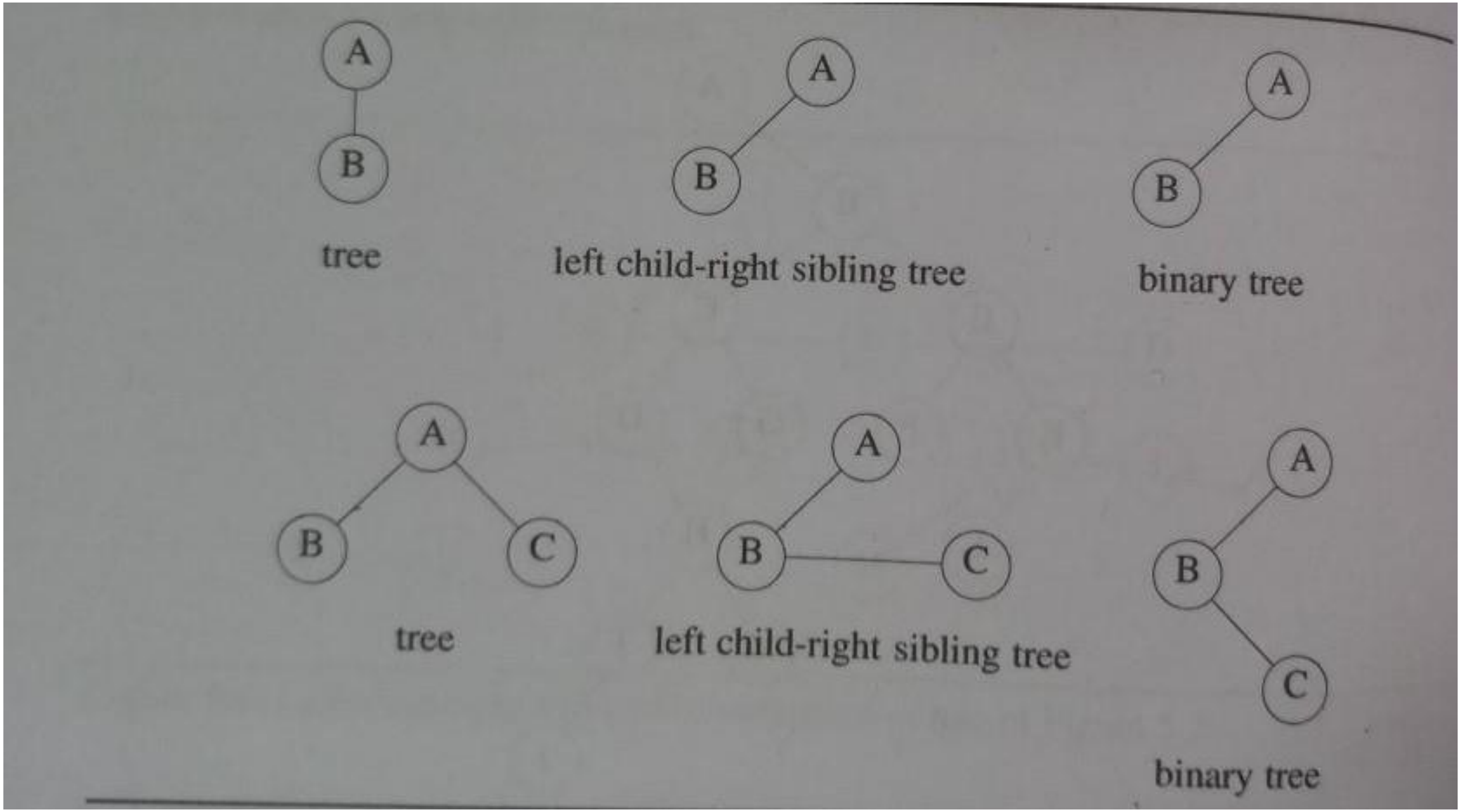


Representation as a Degree-Two Tree

- Final Step

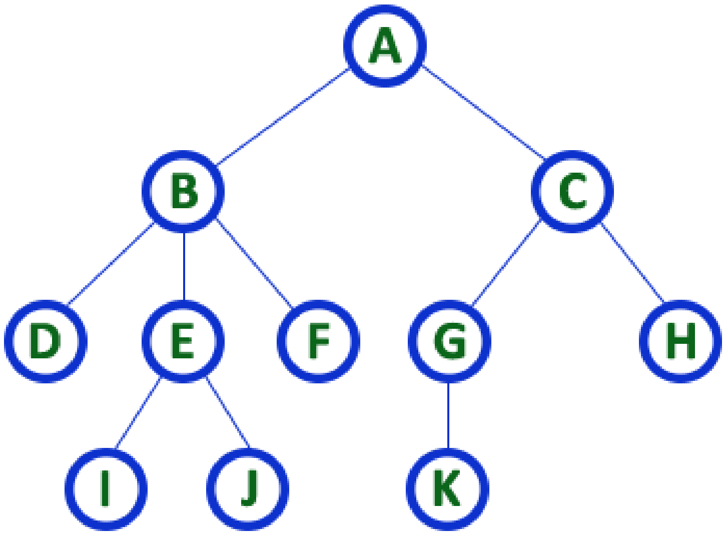


Representation as a Degree-Two Tree



Representation as a Degree-Two Tree

- Consider following tree

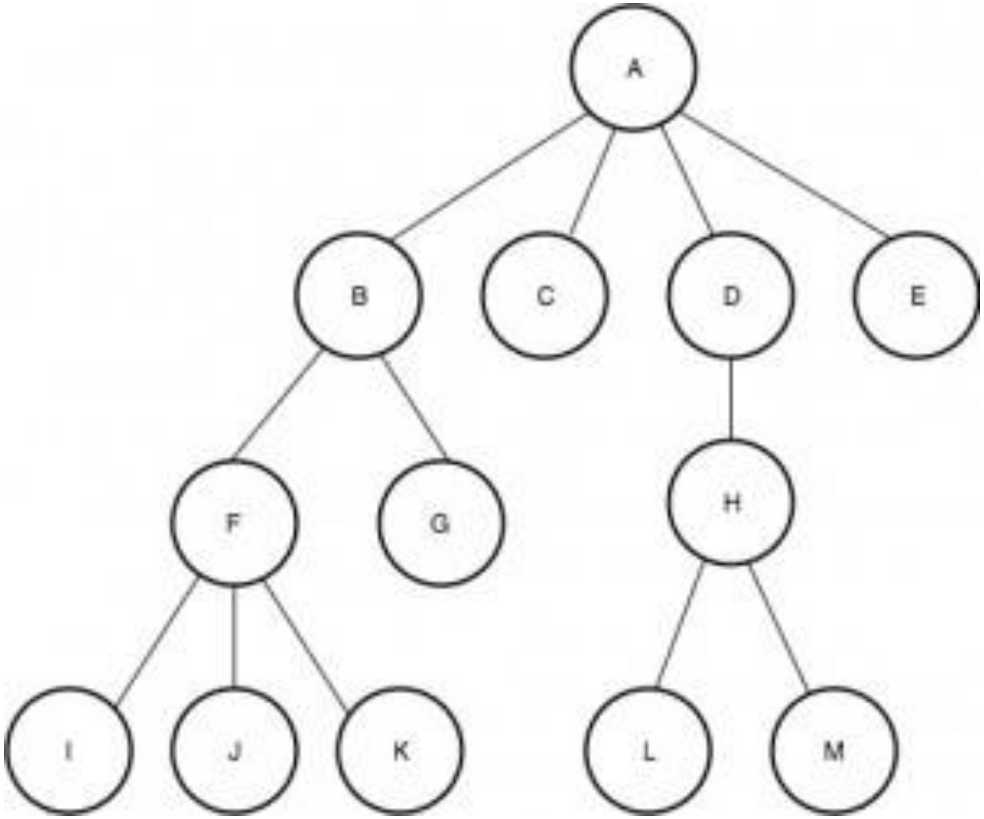


TREE with 11 nodes and 10 edges

- In any tree with 'N' nodes there will be maximum of 'N-1' edges
- In a tree every individual element is called as 'NODE'

Representation as a Degree-Two Tree

Consider the following tree...



Binary Tree

- We have seen that we can represent any tree as a binary tree.
- In fact, binary trees are an important type of tree structure that occurs very often.
- The main characteristics of a binary tree is the stipulation that the degree of any given node must not exceed two.

- A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called *the left subtree* and *the right subtree*.
- Any tree can be transformed into binary tree.
  - by left child-right sibling representation
- The left subtree and the right subtree are distinguished.

Structure of Binary Tree

- The structure defines only as minimal set of operations on binary trees which we use as a foundation on which to build additional operations-

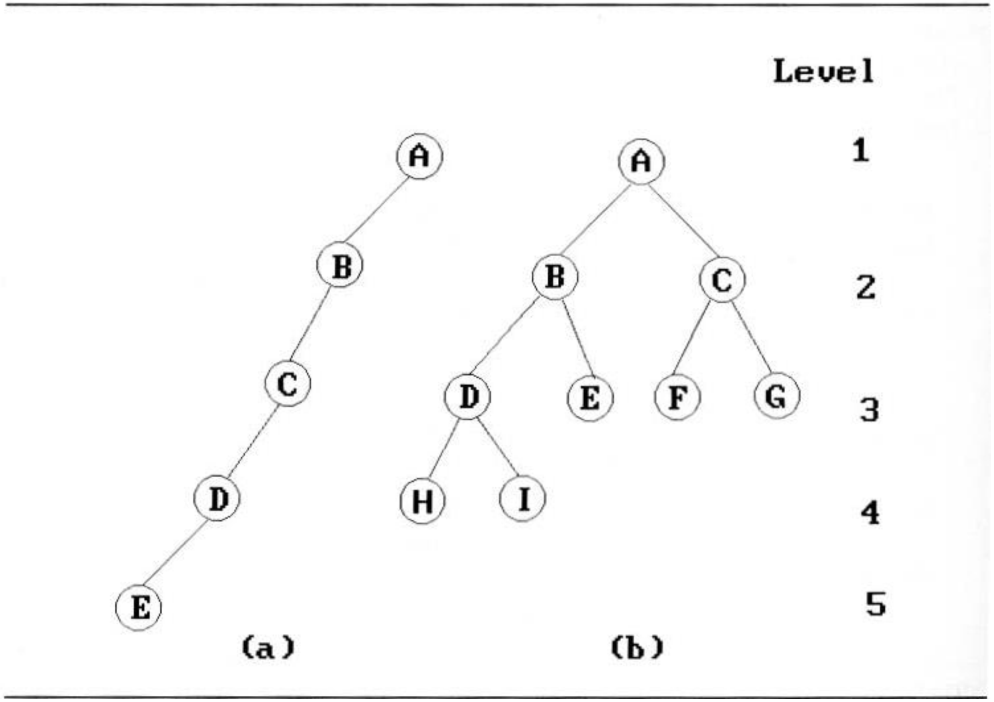
structure *Binary_Tree* (abbreviated *BinTree*) is
objects: a finite set of nodes either empty or consisting of a root node, left *Binary_Tree*, and right *Binary_Tree*.
functions:
for all $bt, bt1, bt2 \in BinTree$, $item \in element$

BinTree Create()	::=	creates an empty binary tree
Boolean IsEmpty(*bt*)	::=	**if** (*bt* == empty binary tree) **return** *TRUE* **else return** *FALSE*
BinTree MakeBT(*bt*1, *item*, *bt*2)	::=	**return** a binary tree whose left subtree is *bt*1, whose right subtree is *bt*2, and whose root node contains the data *item*.
BinTree Lchild(*bt*)	::=	**if** (IsEmpty(*bt*)) **return** error **else return** the left subtree of *bt*.
element Data(*bt*)	::=	**if** (IsEmpty(*bt*)) **return** error **else return** the data in the root node of *bt*.
BinTree Rchild(*bt*)	::=	**if** (IsEmpty(*bt*)) **return** error **else return** the right subtree of *bt*.

Structure 5.1: Abstract data type *Binary_Tree*

Type of Binary trees

- Two special kinds of binary trees:
  (a) *skewed tree*, (b) *complete binary tree*
  - The all leaf nodes of these trees are on two adjacent levels



Properties of Binary Trees

- **Lemma 5.1 [*Maximum number of nodes*]:**
  1. The maximum number of nodes on level i of a binary tree is 2^{i-1}, $i \geq 1$.
  2. The maximum number of nodes in a binary tree of depth k is 2^k-1, $k \geq 1$.

Properties of Binary Trees

- **Lemma 5.2 [*Relation between number of leaf nodes and degree-2 nodes*]:**
  - For any nonempty binary tree, T, if n_0 is the number of leaf nodes and n_2 is the number of nodes of degree 2, then $n_0 = n_2 + 1$.

Properties of Binary Trees

Proof

Let $n0$, $n1$, $n2$ represent the nodes with no children, single child, and two children respectively.

$$n = n0 + n1 + n2 \quad (1)$$

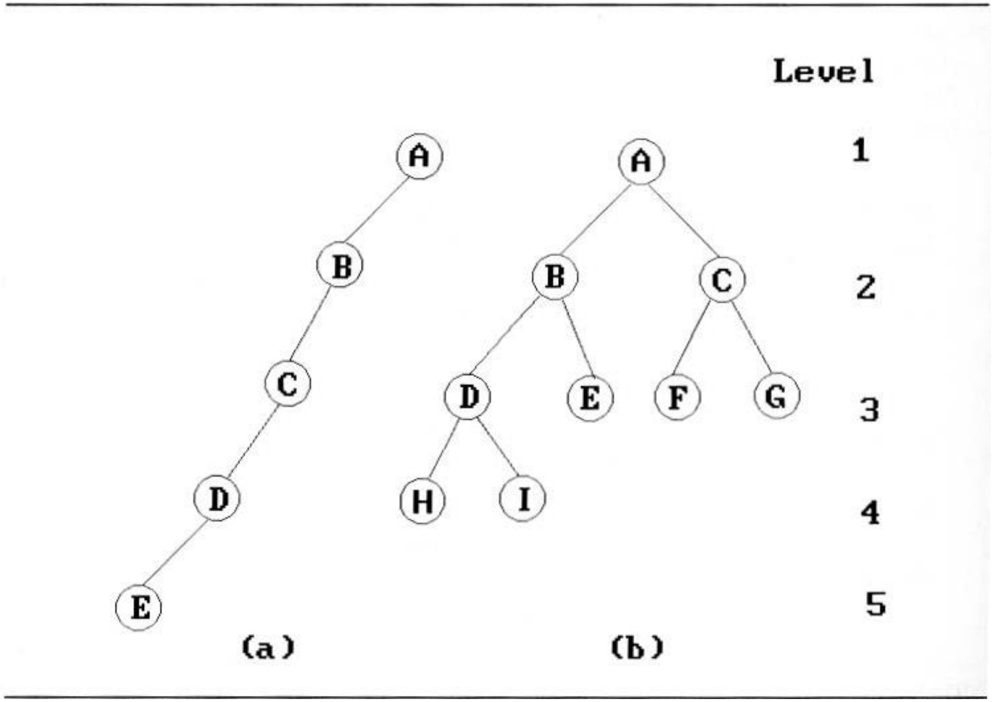
If we count the number of branches in a binary tree, we see that every node except the root has a branch leading into it. If B is the number of branches, then n=B+1. All branches stem from a node of degree one or two. Thus, $B=n_1+2n_2$. Hence, we obtain

$$n = B + 2 = n_1 + 2n_2 + 1 \quad (2)$$

Subtracting Eq. (2) from Eq. (1), and rearranging terms, we get

$$n0 = n2 + 1$$

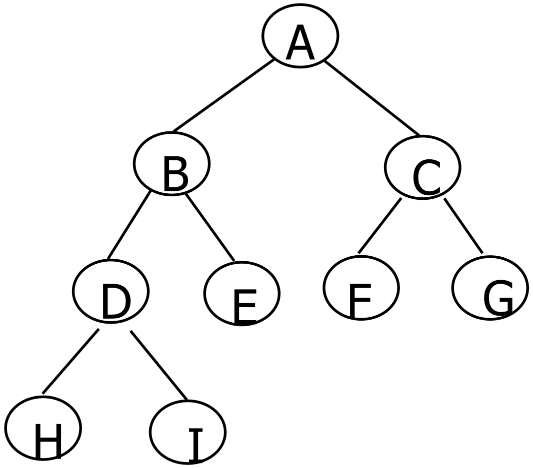
Properties of Binary Trees



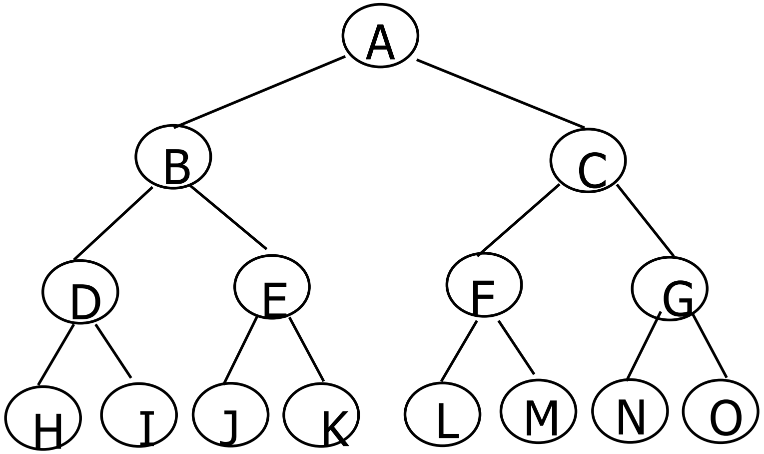
- In fig(a), no=1, n2=0
- In gif(b), n0=5, n2=4

Full BT VS Complete BT

- A full binary tree of depth k is a binary tree of depth k having 2^k -1 nodes, k>=0.
- A binary tree with n nodes and depth k is complete *iff* its nodes correspond to the nodes numbered from 1 to n in the full binary tree of depth k.



Complete binary tree



Full binary tree of depth 4

Binary Tree Representations

- A binary tree data structure is represented using two methods. Those methods are as follows...
  - **Array Representation**
  - **Linked List Representation**