Book

A Simplified Approach to

Data Structures

Prof.(Dr.)Vishal Goyal, Professor, Punjabi University Patiala

Dr. Lalit Goyal, Associate Professor, DAV College, Jalandhar

Mr. Pawan Kumar, Assistant Professor, DAV College, Bhatinda

Shroff Publications and Distributors

Edition 2014

BINARY TREE

CONTENTS

- Introduction
- Tree
- Binary Tree
    - Properties of binary tree.
    - Memory representation of binary tree.
    - Operations performed on binary tree.

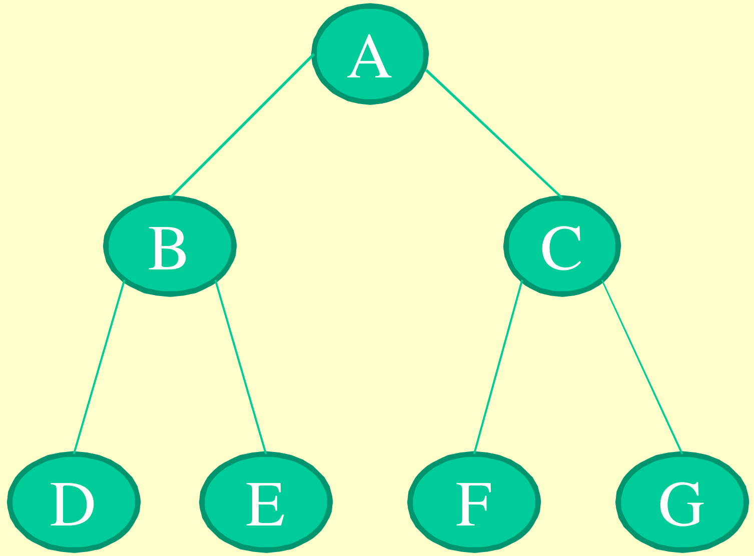
INTRODUCTION

TREE

Tree is a finite non-empty set of elements in which first element is called **Root** and remaining elements are partitioned into a number of disjoint subsets each of which is itself a tree. Each element in a tree is called **Node.**

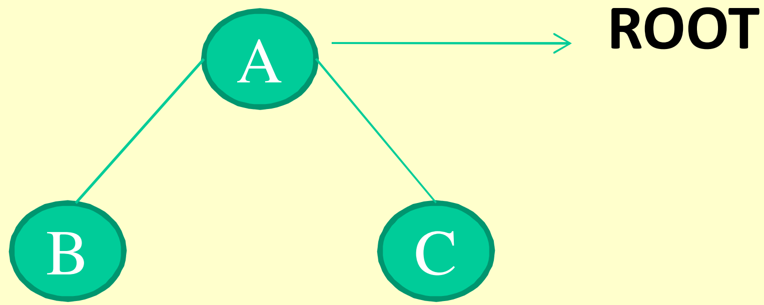
GENERAL REPRESENTATION OF TREE

In tree, **A** is **root node** and remaining elements are **nodes.** The subset to the left of the root node is called **Left subtree** and node to the right of the root is called **Right subtree.** The elements of the tree have the **parent - child** relationship.



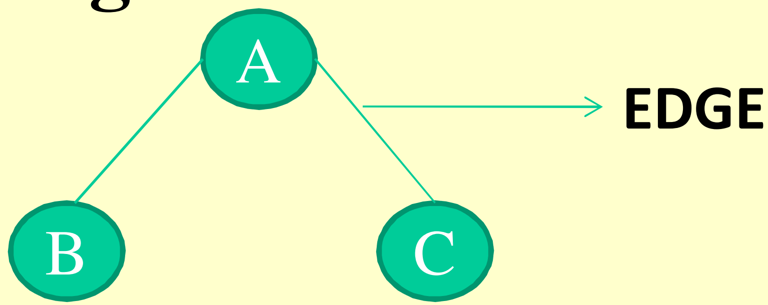
ROOT

The root of a tree is the origin of the tree form where the tree origins or starts . Node **A** is the root of the tree.



EDGE

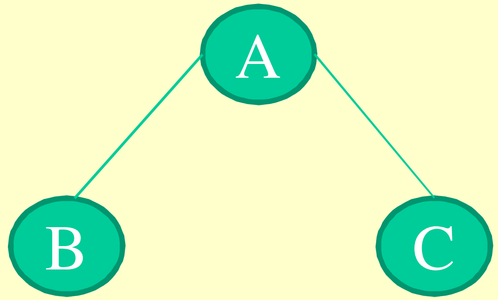
A line which connect a parent node to its child node is called **Edge** or **Branch.**



CONTINUED….

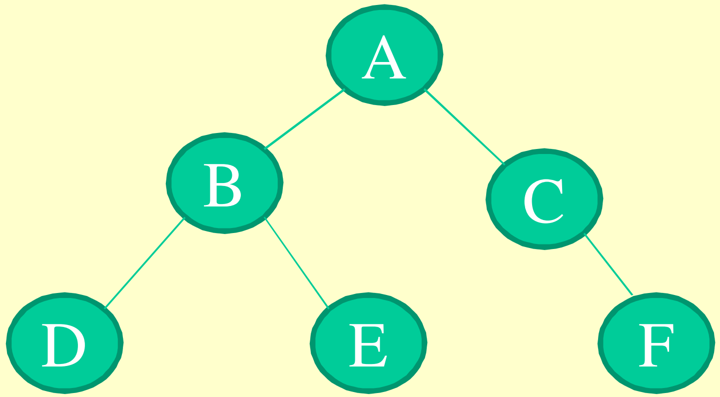
- **SUCCESSOR**

Left and right subtree of tree are called as successors or child of a node. A is having two successors as B and C.



- **TERMINAL NODE**

A node is called as terminal node if it has no children.



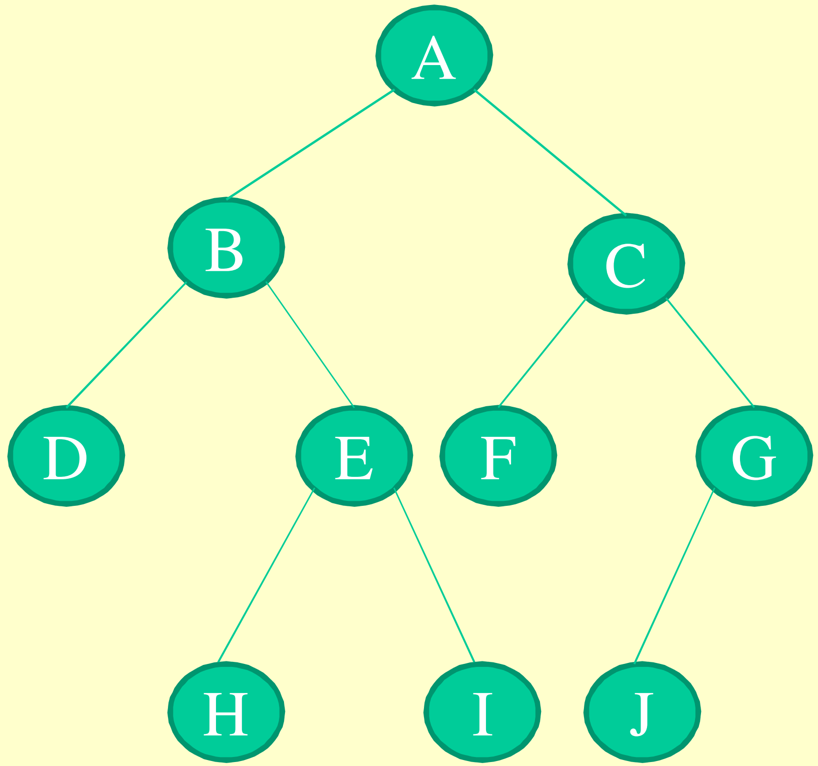
CONTINUED…

- **PARENT NODE**

  Node **A** is said to be parent of **B** and **C**. Similarly **B** is the parent of **D** and **E**.

- **SIBLINGS**

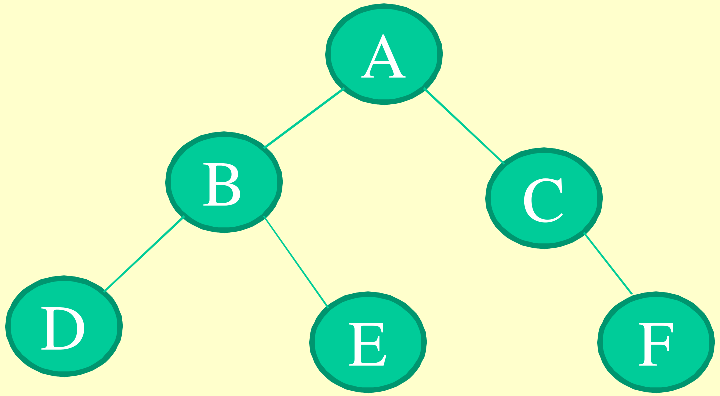
  The nodes which are having same parent are known as siblings. **B** and **C** are the siblings as they are children of same parent node **A**.



TREE TERMINOLOGIES

- **PATH**

  A path between any two nodes in tree is a sequence of nodes in which successive nodes are connected by edges.



Path from A to D
A → B → D
Path from A to E
A → B → E
Path from A to F
A → C → F

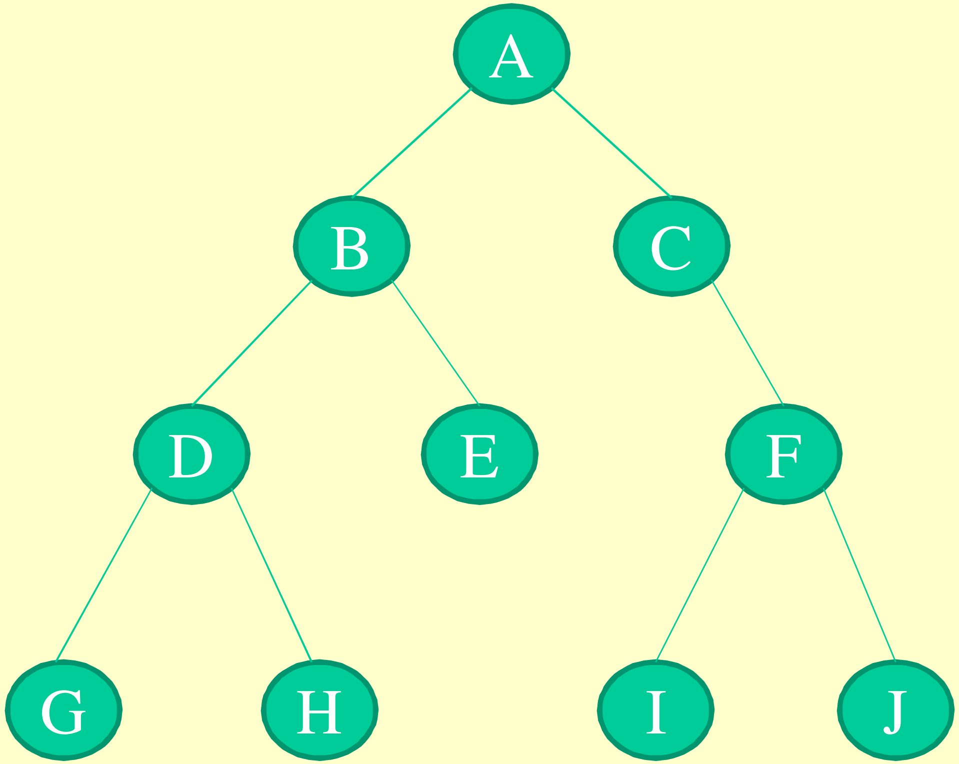
LENGTH

The length of a path in a tree is total number of edges which come across that path.

Length of Path A to B:1
Length of Path A to C:1
Length of Path A to D:2
Length of Path A to E:2
Length of Path A to F:2

- **HEIGHT**

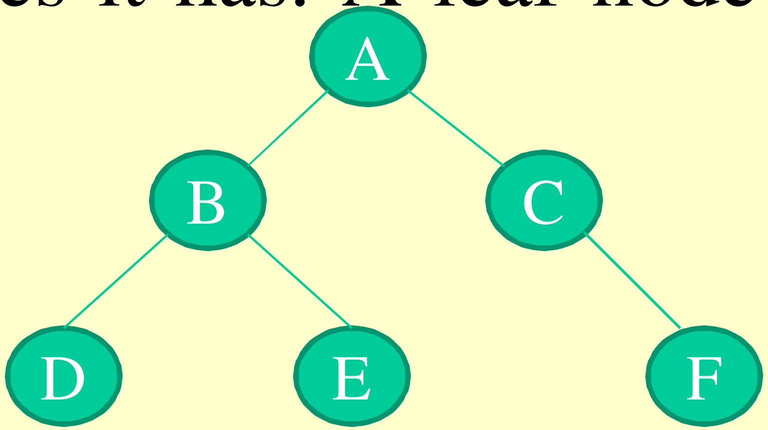
The height of any node in the tree is length of the longest path from that node to a terminal node . The height of the root is treated as the height of tree.



Height of A:3
Height of B:2
Height of C:2
Height of D:1
Height of E:1
Height of F:1
Height of G:0
Height of H:0
Height of I:0
Height of J:0

- **DEGREE**

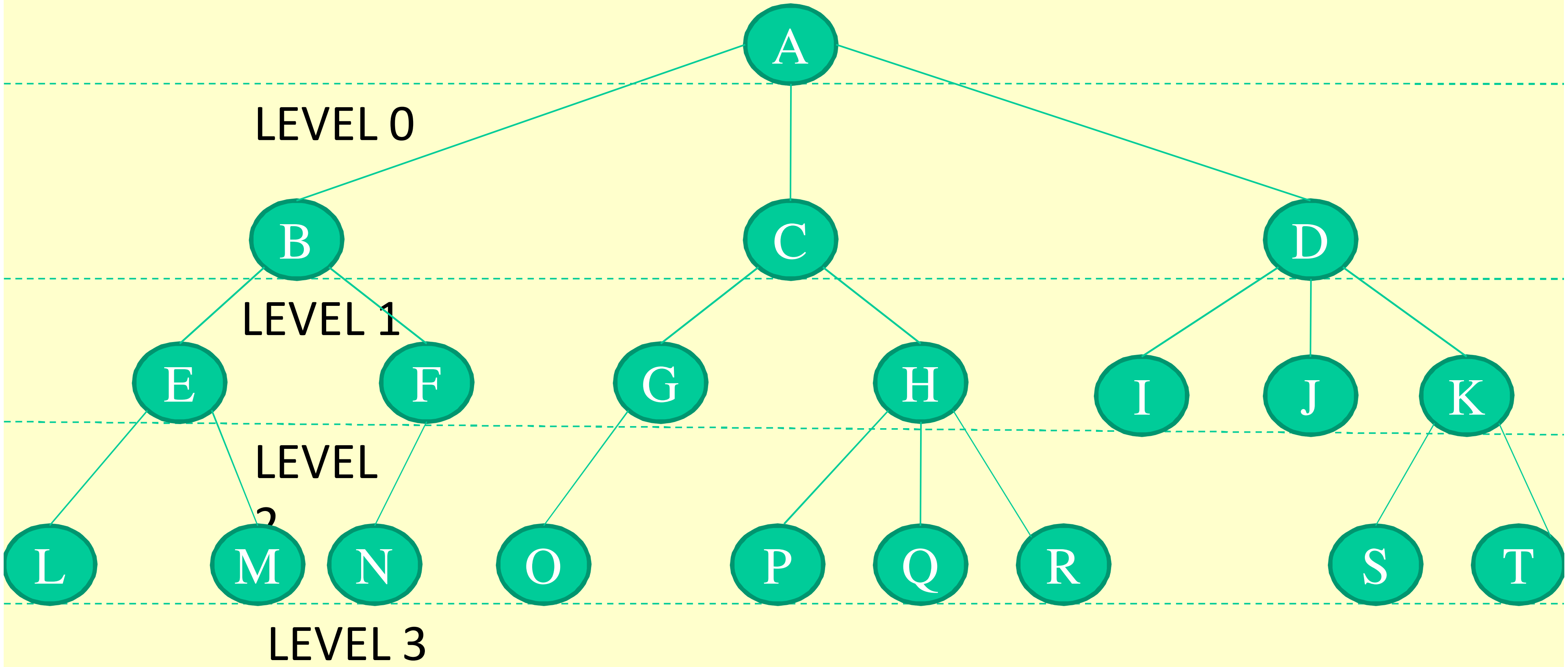
The degree of node can be defined as number of child nodes it has. A leaf node always has degree zero.



Degree of A = 2
Degree of B = 2
Degree of C = 1
Degree of D = 0
Degree of E = 0
Degree of F = 0

- **LEVEL**

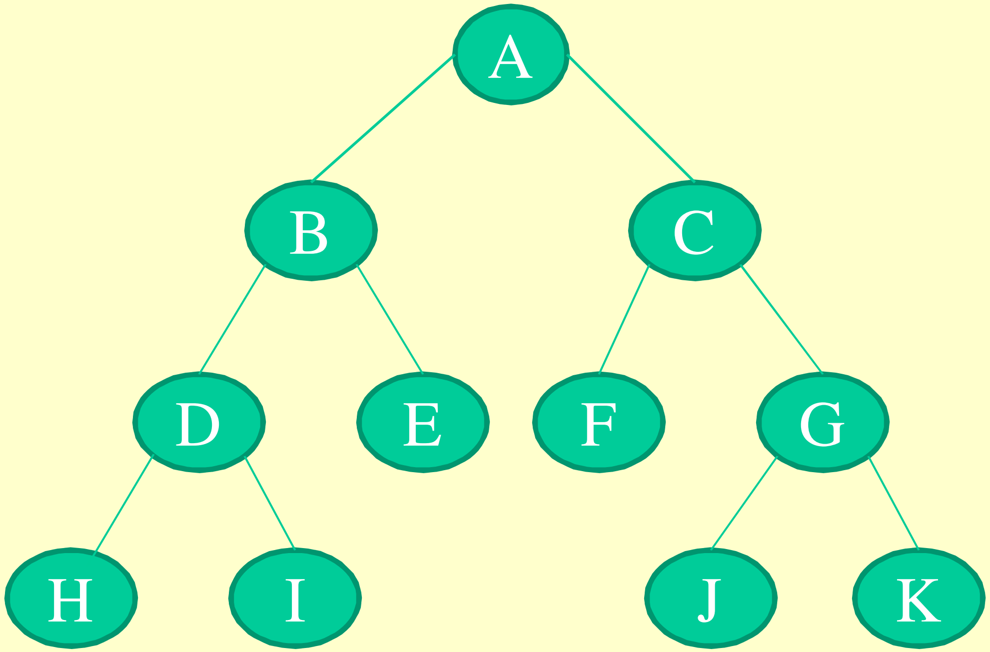
The level of node is the length or path from the root. The root node of the tree has level 0,and the level of any node in the tree is one more than the level of its father.



LEVELS OF NODES IN A TREE

BINARY TREE

A binary tree can be defined as a finite collection of nodes where each node n is having the property that it can have 0,1 or 2 children.



CONTINUED….

- A binary tree can be defined as a finite collection of nodes where each node n is having the property that it can have 0,1 or 2 children.

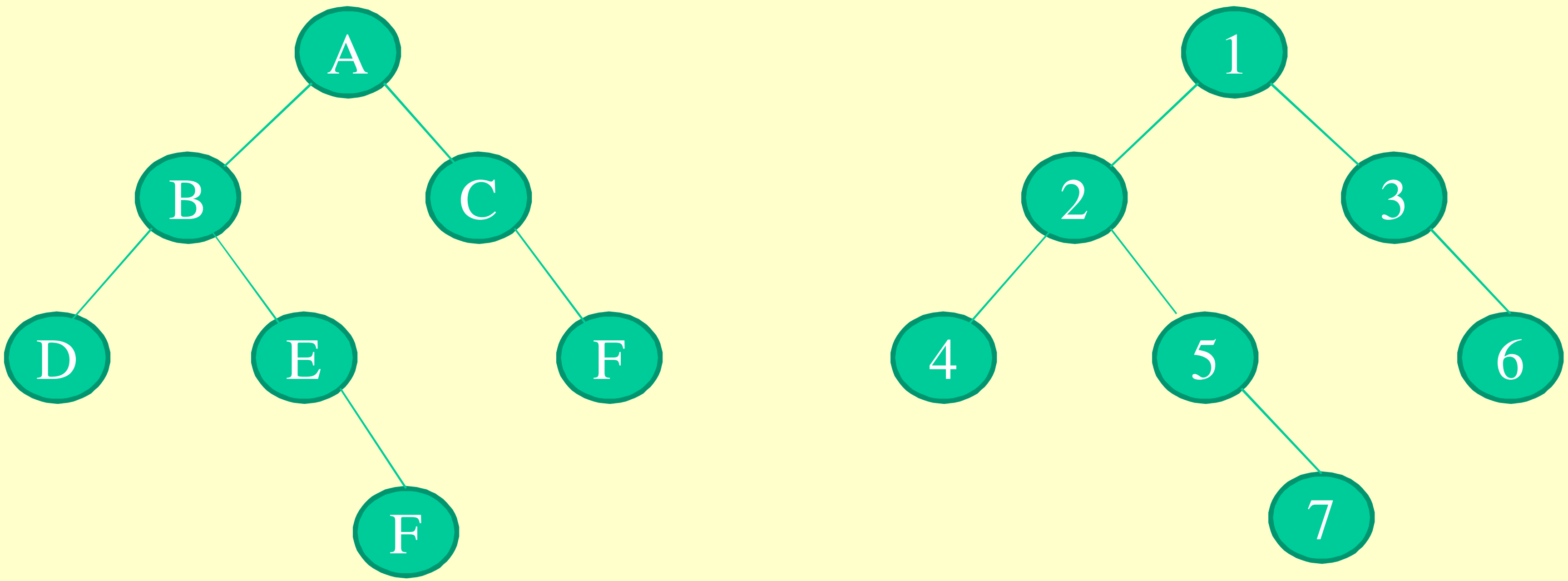
- A binary tree may be empty known as NULL tree or it contains a special node called **root** of the tree and remaining nodes in the tree form the **left right binary sub trees.**

TYPES OF BINARY TREE

- **Similar Binary Trees**
- **Equivalent Binary Trees**
- **Complete Binary Trees**
- **Strictly Binary Trees**

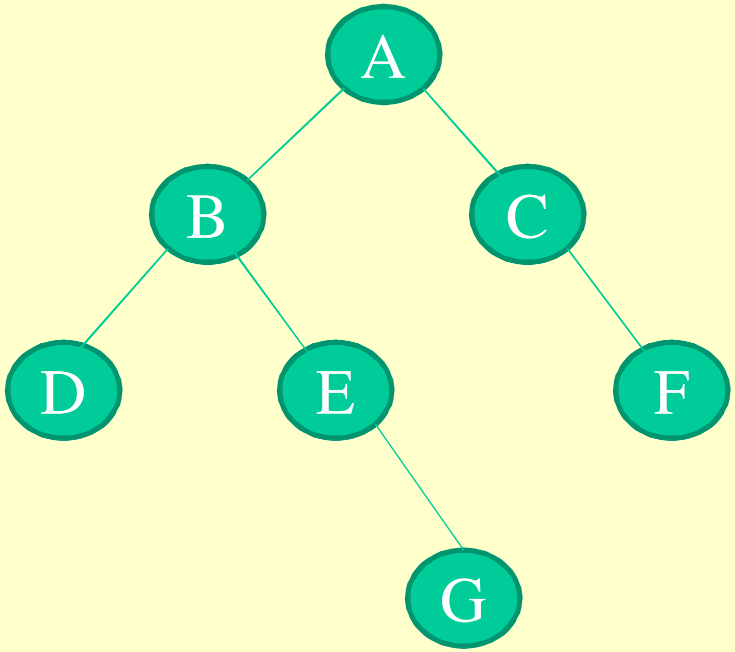
SIMILAR BINARY TREE

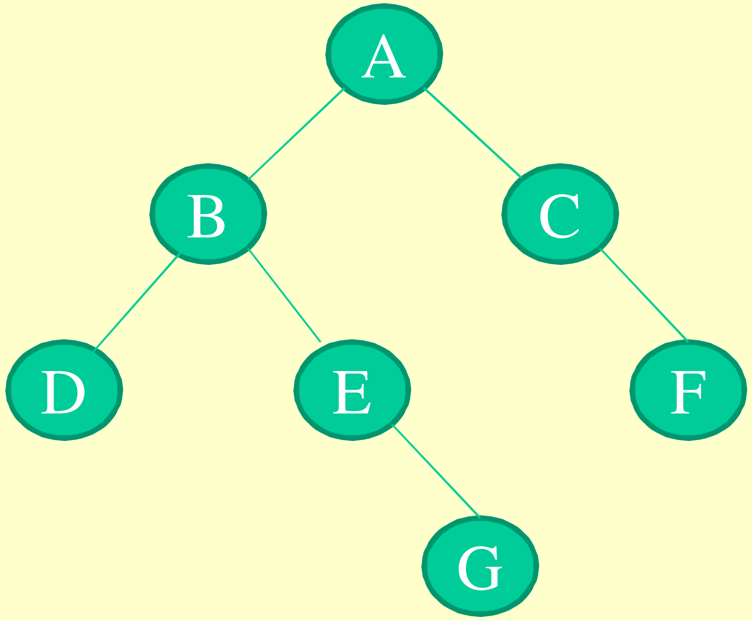
Two binary tree are called similar if both are having similar structure but the elements in both the tree can be different.



EQUIVALENT BINARY TREE

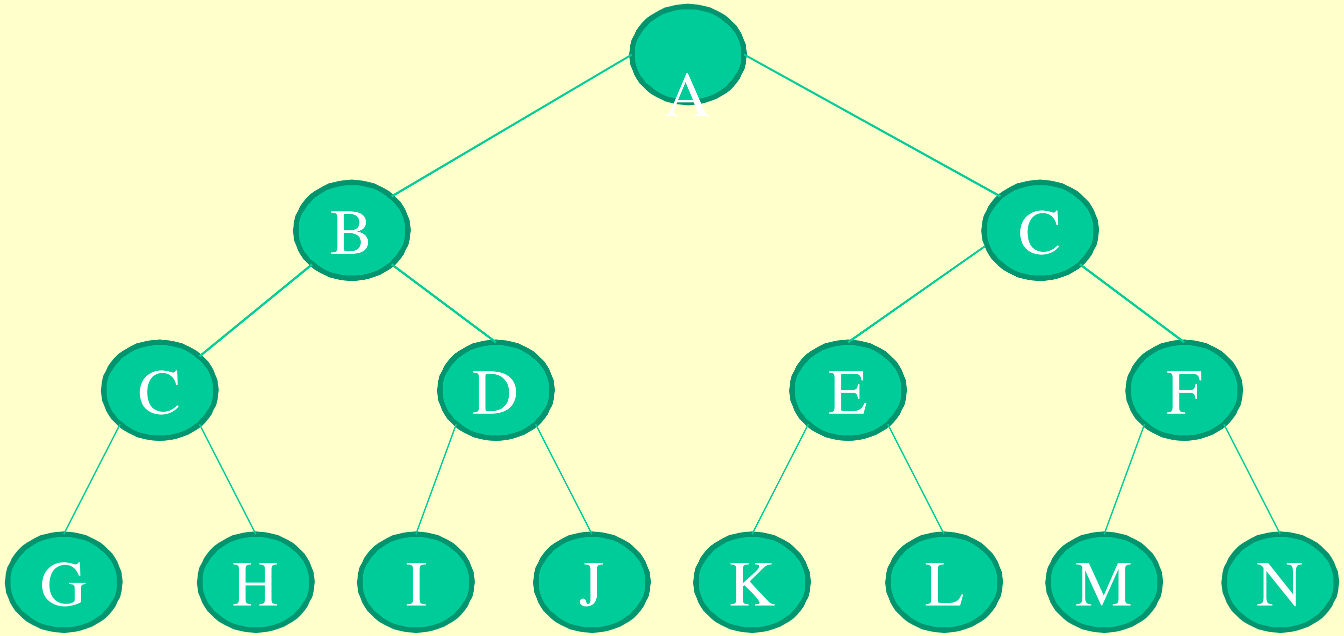
Two binary trees are said to be equivalent or copies if they are similar & are having the same contents in their respective nodes.





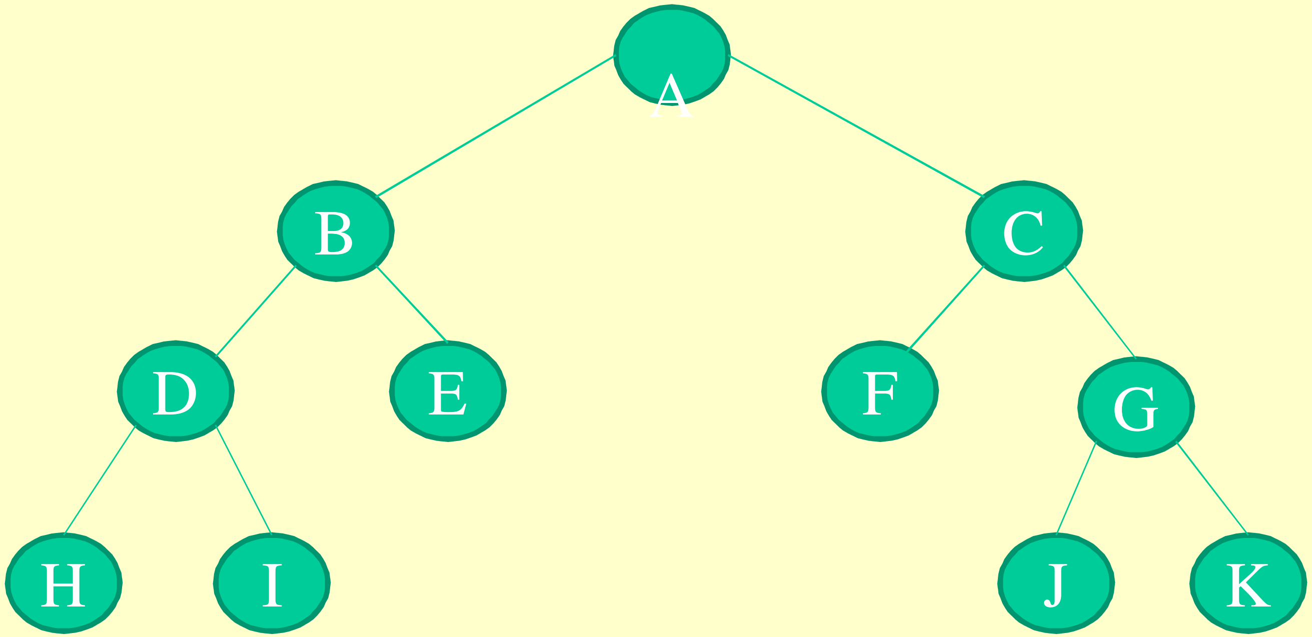
COMPLETE BINARY TREE

A binary tree is said to be complete if it contains the maximum number of nodes at each level except the last level.



STRICTLY BINARY TREE

A binary tree is called Strictly binary tree if all non leaf nodes of tree contains exactly two children. Every non leaf node of the binary tree contains left right subtree.



PROPERTIES OF BINARY TREE

- A Binary Tree with **n** nodes has exactly **n-1** edges.
- In a binary tree every node except the root node has exactly one parent.
- In a binary tree there is exactly one path connecting any two nodes in the tree.
- The minimum number of nodes in a binary tree of height **h** is **h+1**.

CONTINUED….

- The maximum number of nodes in a binary tree of height **h** is

  $\mathbf{2^{(h+1)}-1.}$
- Number of leaf nodes in a complete binary tree is **(n+1)/2**.
- In a complete binary tree,

  Number of external nodes = Number of internal nodes+1.

MEMORY REPRESENTATION OF BINARY TREE

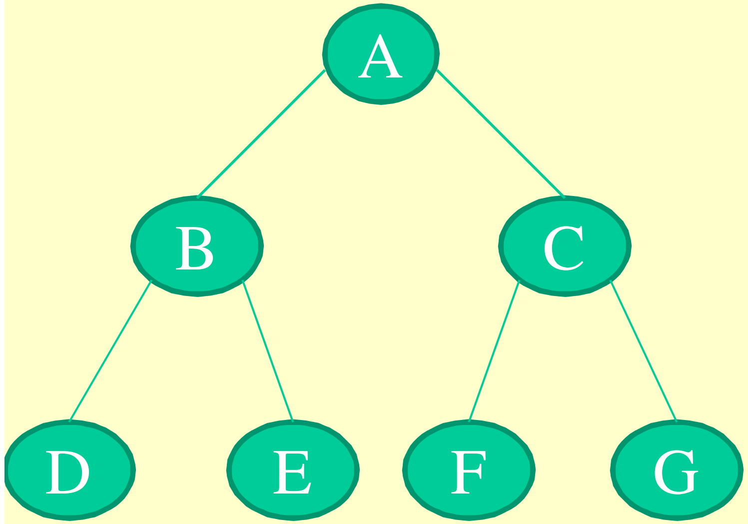
A binary tree can be represented into computer memory
using two ways:-

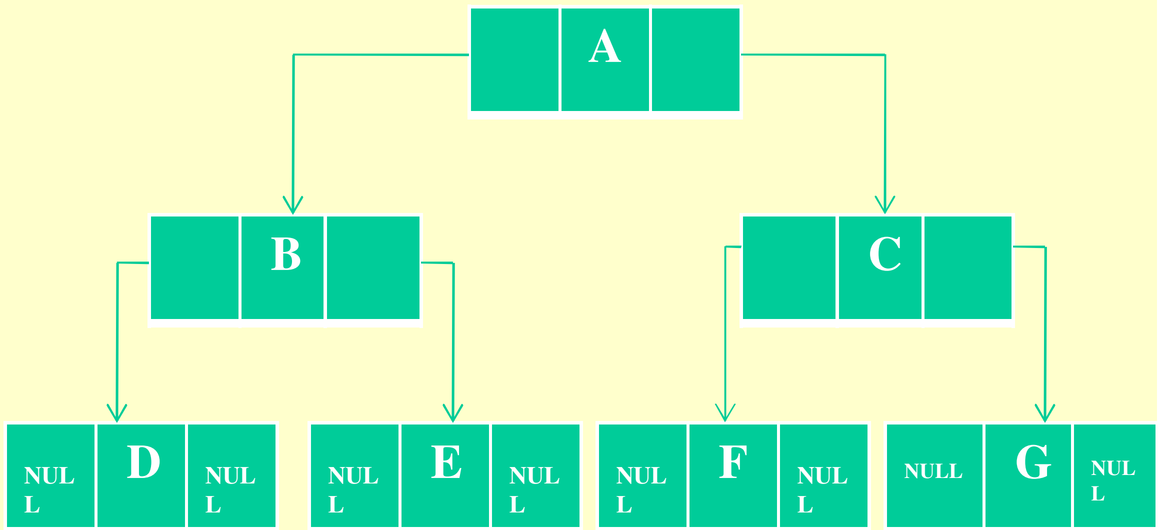
1) **Linked Representation**
2) **Sequential Representation**

Linked Representation Of Binary Tree

Each element of tree is represented by a node having three parts.

- 1st Part (Info)- Which stores the element.
- 2nd Part (left)- Stores the address of left child node.
- 3rd Part (Right)- Stores the address of right child node.





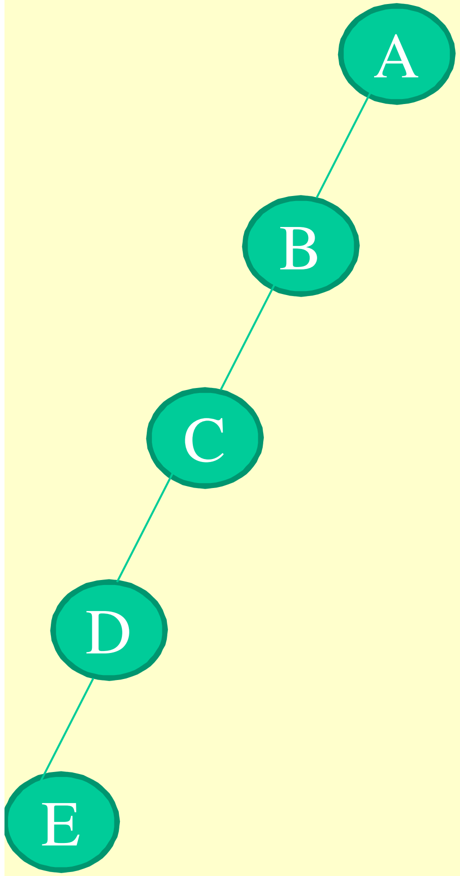
SEQUENTIAL REPRESENTATION

- Root of tree is always stored at the 1st array index & its left and right child will be stored at 2nd and 3rd index respectively.

- If a node occupies the Kth index of array then its:-
  - Left child will be stored at $(2 \times k)^{th}$ array index.
  - Right child will be stored at $(2 \times (k + 1))^{th}$ array index.

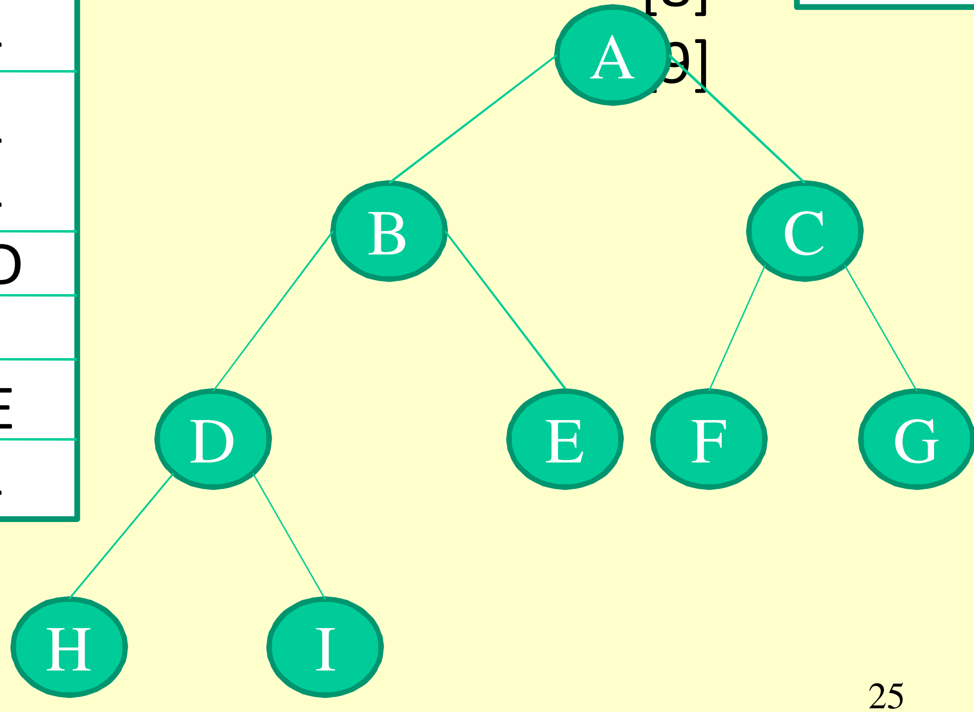
- Sequential representation of binary tree of height **h** will require an array size $2^{(h+1)} - 1$

CONTINUED…



[1]	A
[2]	B
[3]	-
[4]	C
[5]	-
[6]	-
[7]	-
[8]	D
[9]	
[10]	E
[11]	-

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



OPERATIONS PERFORMED ON BINARY TREES

Various operations performed on Binary Tree are:

1) **Traversing.**
2) **Finding the number of external and internal nodes.**

TRAVERSING BINARY TREE

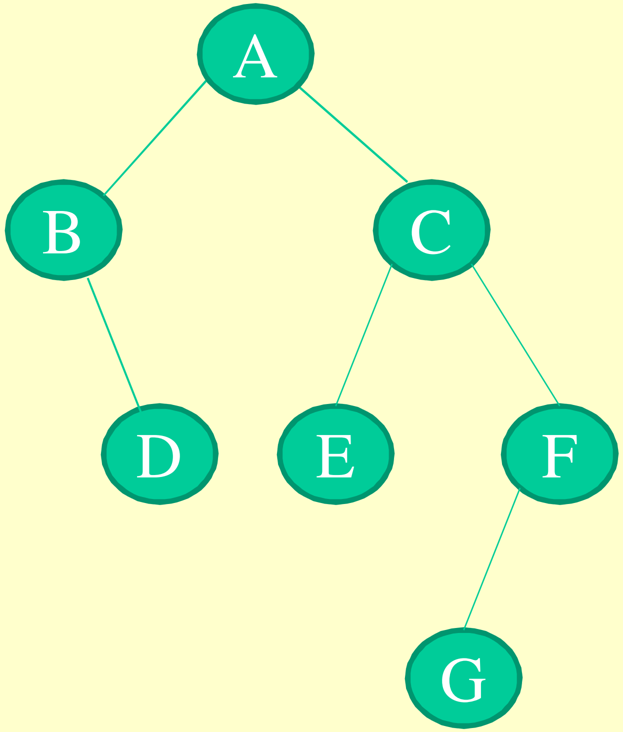
- Traversing is the process of visiting each node in the tree.
- There are standard methods for traversal of Binary Tree:
  - **Pre-Order Traversal.**
  - **In-Order Traversal.**
  - **Post-Order Traversal.**

- While traversing, three main activities take place:
  - **Visiting the Root.**
  - **Traversing the left subtree.**
  - **Traversing the right subtree.**

PRE ORDER TRAVERSAL

- This is also known as **depth-first order or Root-Left-Right traversal.**
- In this method, traversal order followed is:
  - **Visit the root.**
  - **Traverse the left sub tree .**
  - **Traverse the right sub tree**

PRE ORDER TRAVERSAL



Nodes are visited in preorder as: **A B D C E F G**

ALGORITHM

Traverse Binary Tree In Pre Order Manner.

Step1: If **Root=Null** Then

Print ”Tree is Empty”

Exit

Else

Set **Pointer=Root**

[End If]

Step2: Initialize an empty Stack by pushing **Null** into it and setting the Stack variable **Top** to 1

Step3: Repeat While **Pointer** $\neq$ **Null**

Print **Pointer** $\rightarrow$ **Info**

If **Pointer** $\rightarrow$ **Right** $\neq$ **Null** Then

CONTINUED….

Push **Pointer → Right** onto the Stack by incrementing stack's variable **Top**.

If **Pointer → Left ≠ Null** Then

Set **Pointer=Pointer → Left**

Else

Set **Pointer=Stack → Top**

Decrement the Stack's variable **Top** by 1

[End If]

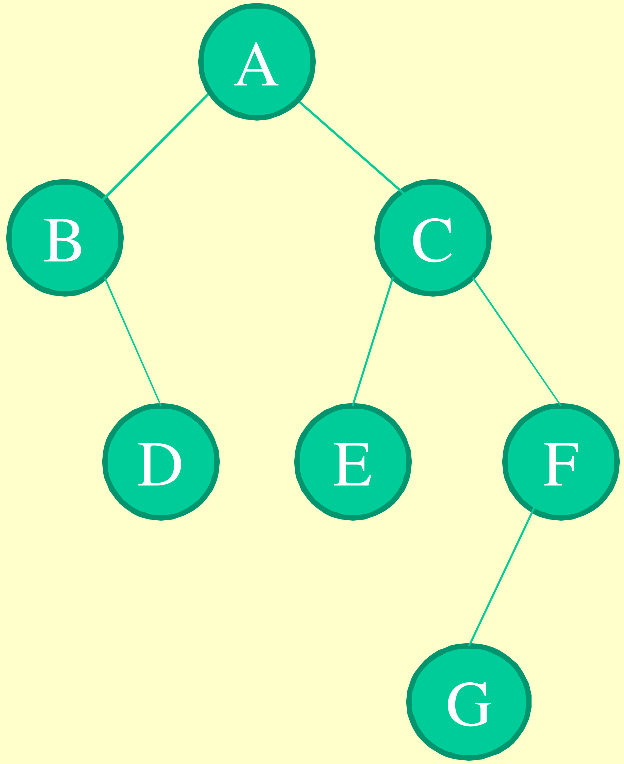
[End Loop]

Step4: Exit

IN ORDER TRAVERSAL

- This is also known as **symmetric order or Left-Root-Right traversal.**
- In this method, traversal order followed is:
  - **Traverse the left subtree.**
  - **Visit the root.**
  - **Traverse the right subtree .**

IN ORDER TRAVERSAL



The nodes are visited in order as : **B D A E C G F**

ALGORITHM

Traverse Binary Tree In In Order Manner.

Step1: If **Root=Null** Then

Print ”Tree is empty”

Exit

Else

Set **Pointer=Root**

[End If]

Step2: Initialize an empty Stack by pushing **Null** into it and setting the Stack variable **Top** to 1

Step3: Set **Flag=True**

<u>Step 4:</u> Repeat Steps 5 to 10 while **Flag = True**

<u>Step 5:</u> Repeat while **Pointer ≠ Null**

Push **Pointer** onto the stack and Increment the stack variable **Top** by 1

Set **Pointer = Pointer → Left**

[End Loop]

<u>Step 6:</u> Set **Pointer = Stack → Top**

<u>Step 7:</u> Decrement the stack variable **Top** by 1

<u>Step 8:</u> Set **Flag= False**

<u>Step 9:</u> Repeat while **Pointer ≠ Null** and **Flag = False**

Continued…

Set **Pointer = Pointer → Right**

Set **Flag = True**

Else

Pointer=Stack → Top

decrement the stack variable **Top** by 1

[End If]

[End Loop]

<u>Step 10</u>: If **Pointer = Null** Then

Set **Flag = False**

[End If]

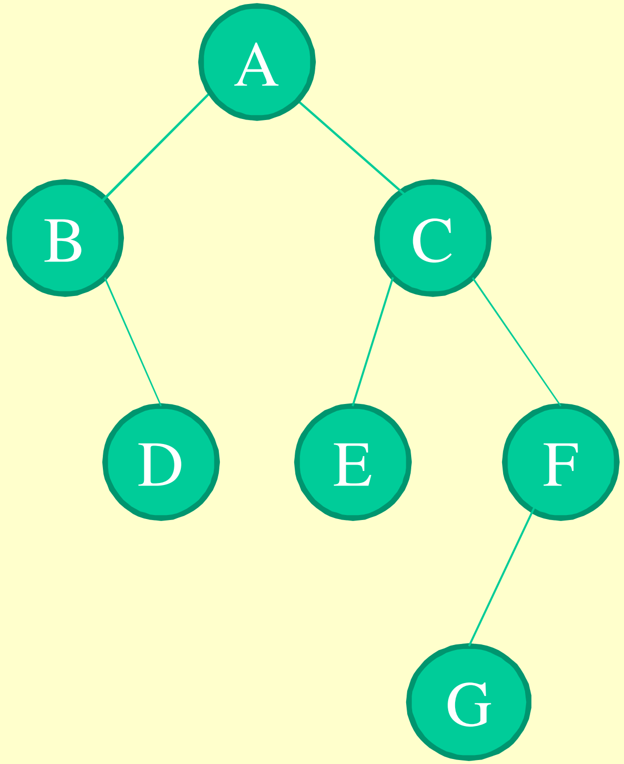
[End Loop]

<u>Step11</u>: Exit

POST ORDER TRAVERSAL

- This is also known as **Left-Right-Root traversal.**
- In this method, traversal order followed is:
  - **Traverse the left subtree in Post-order traversal.**
  - **Traverse the right subtree in Post-order traversal.**
  - **Visit the root.**

POST ORDER TRAVERSAL



The nodes are visited in Post Order as : **D B E G F C A**

ALGORITHM

Traverse Binary Tree In In Order Manner.

Step1: If **Root=Null** Then

Print "Tree is empty"

Exit

Else

Set **Pointer=Root**

[End If]

Step2: Initialize an empty stack by pushing **Null** into it and setting the stack variable **Top** to 1

Step 3: Set **Flag=True**

CONTINUED….

Step 4: Repeat Steps 5 to 10 while **Flag = True**

Step 5: Repeat while **Pointer ≠ Null**

Push **Pointer** onto the Stack and
Increment the stack variable **Top** by 1

Set **Pointer = Pointer → Left**

[End Loop]

Step 6: Set **Pointer = Stack → Top**

Step 7: Decrement the stack variable **Top** by 1

Step 8: Set **Flag = False**

CONTINUED….

<u>Step 9:</u> Repeat while **Pointer ≠ Null** and **Flag = False**

If **Pointer** > 0 Then

Print: **Pointer → Info**

Set **Pointer = Stack → Top**

decrement the stack variable **Top** by 1

Else

Set **Pointer = Pointer**

Set **Flag = True**

[End If]

CONTINUED….

Step 10: If **Pointer = Null** Then

Set **Flag = False**

[End If]

[End Loop]

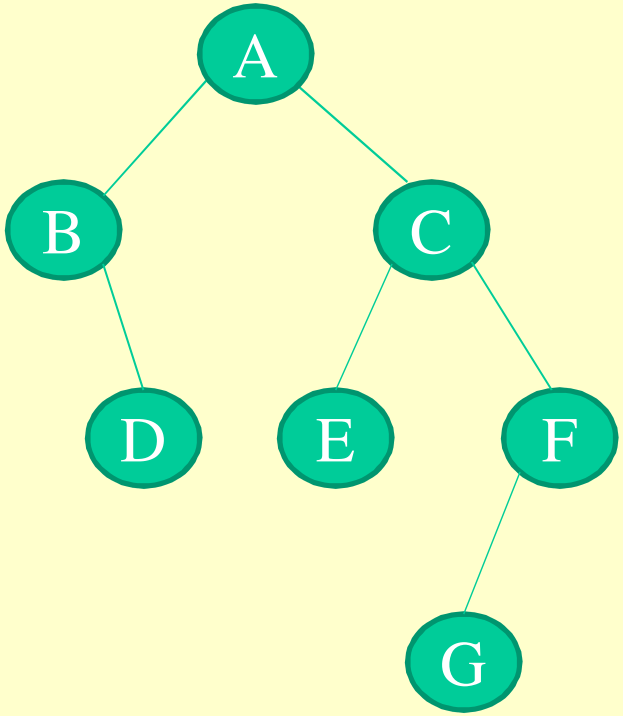
Step 11: Exit

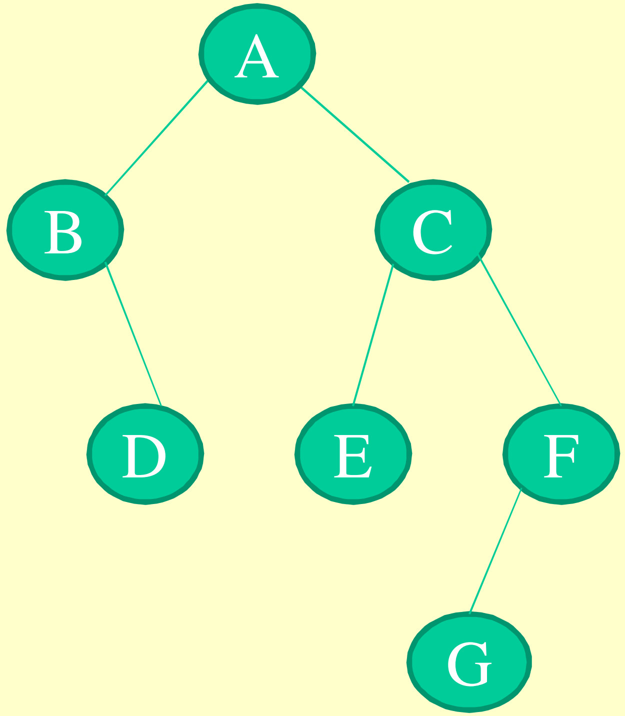
TRAVERSING USING RECURSION

- We can also do traversing using recursive in binary tree.
- In recursion every node is traversed and creates a copy of every call just as factorial program through recursion.
- There are standard methods for the traversal of Binary Tree through recursion, these are:

  - **Pre-Order Traversal**
  - **In-Order Traversal**
  - **Post-Order Traversal**

PRE ORDER TRAVERSAL





The nodes are visited in Pre Order as: A B D C E F G

ALGORITHM

Traversing a binary tree in Pre order manner recursively.

RecPreTraversal (Root)

Step 1: If **Root = Null**

Print “Tree is Empty”

Return

Else

Print **Root → Info**

Continued…

<u>Step 2:</u> If **Root → Left** ≠ **Null** Then

Call RecPreTraversal (**Root → Left**)

[End If]

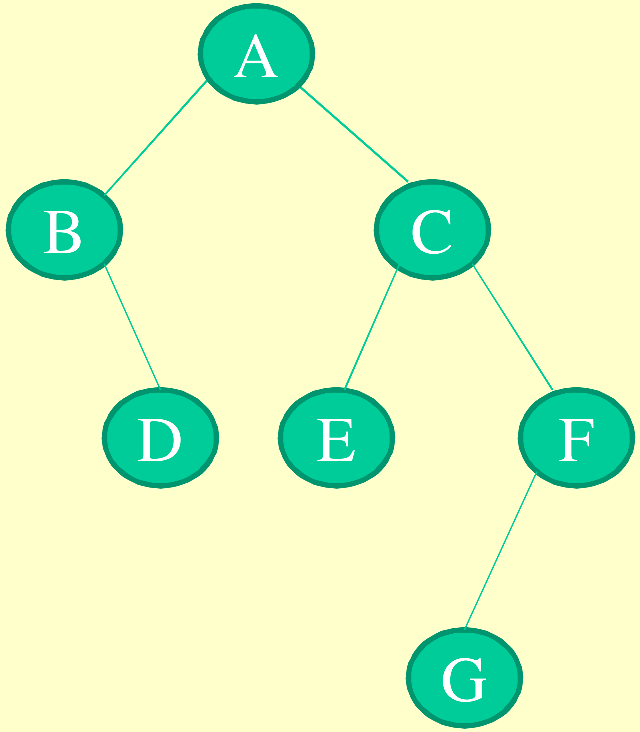
<u>Step 3:</u> If **Root → Right** ≠ **Null** Then

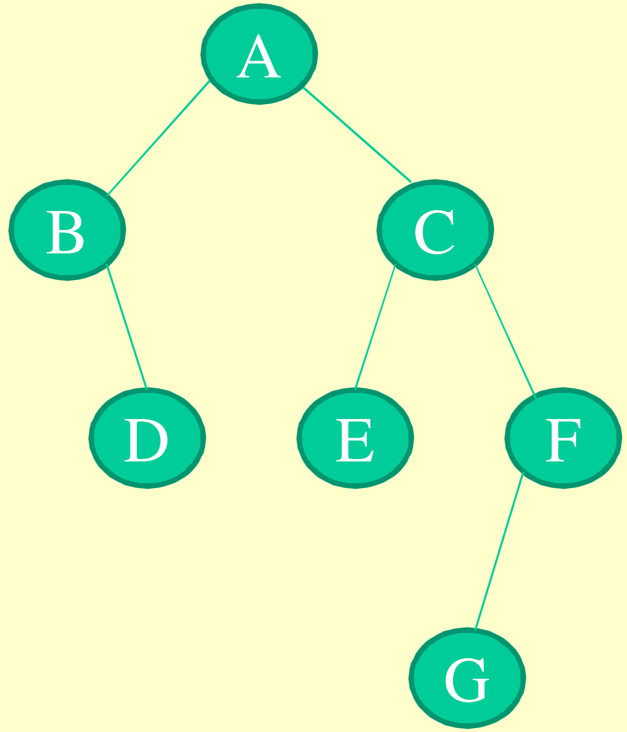
Call RecPreTraversal (**Root → Right**)

[End If]

<u>Step 4:</u> Return

IN ORDER TRAVERSAL





The nodes are visited in In Order as: **B D A E C G F**

Traversing a binary tree in In order manner recursively.

Call RecInTraversal(Root)

Step 1: If **Root = Null**

Print "Tree is Empty"

Return

[End If]

Step 2: If **Root → Left ≠ Null** Then

Call RecInTraversal (**Root → Left**)

[End If]

Step 3: Print **Root → Info**

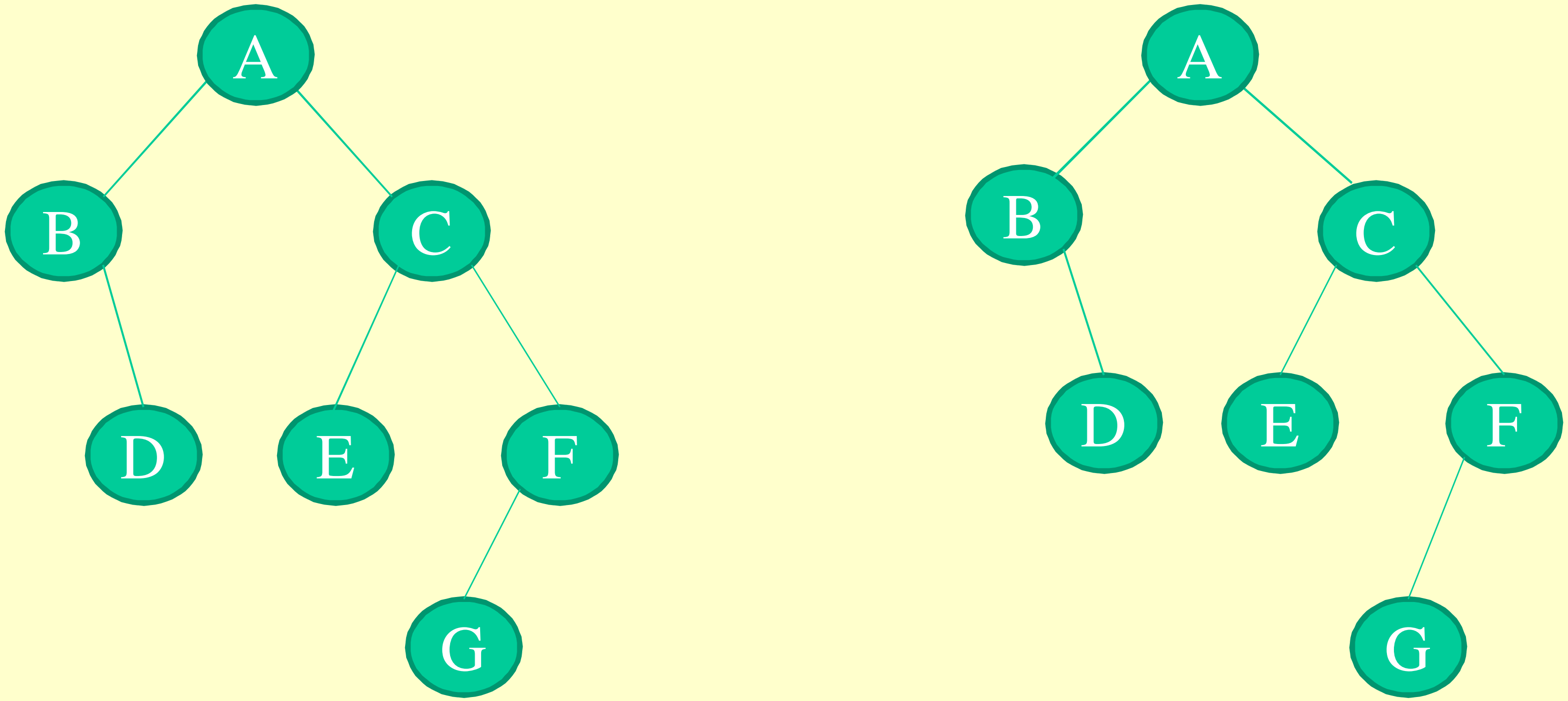
Step 4: If Root **Right** ≠ **Null** Then

Call RecInTraversal (**Root → Right**)

[End if]

Step 5: Return

POST ORDER TRAVERSAL



The nodes are visited in Post Order as **D B E G F C A**

ALGORITHM

Traversing a binary tree in Postorder manner recursively.

RecPostTraversal (**Root**)

Step 1: If **Root = Null**

Print "Tree is Empty"

Return

[End if]

Step 2: If **Root → Left** ≠ **Null** Then

Call RecPostTraversal (**Root → left**)

[End if]

CONTINUED….

Step 3: If **Root → Right** ≠ **Null** Then

Call RecPostTraversal (**Root → Right**)

[End if]

Step 4: Print **Root → Info**

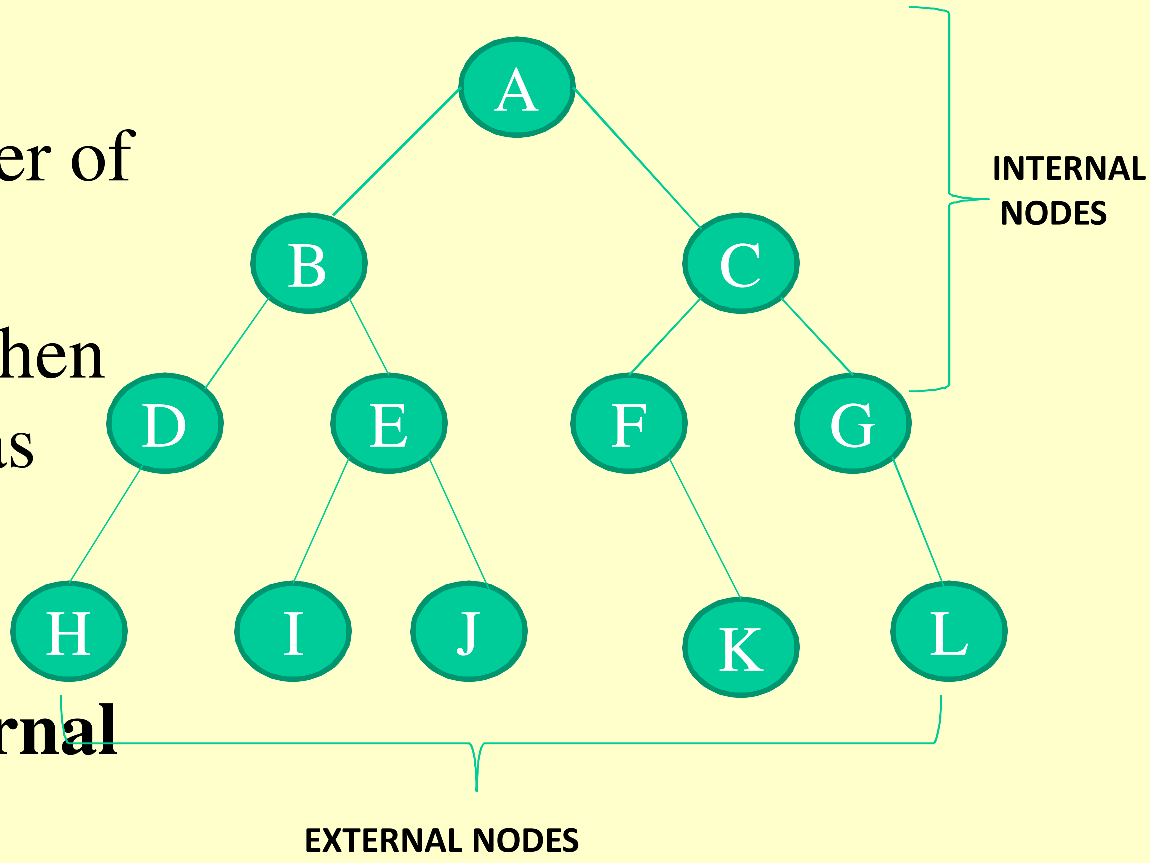
Step 5: Return

TO FIND INTERNAL AND EXTERNAL NODES

- The nodes which do not have any left and right child are said to be external nodes or leaf nodes.
- All other nodes having one or two children are said to be internal nodes.
- To find the number of external/internal nodes in a tree we have to traverse it and we can traverse the tree using any of the three traversal methods.

CONTINUED….

- **During traversing the tree**
  - Each node will be tested for its number of children.
  - If it has any child then it will be counted as **internal node** Otherwise, It will be counted as **external node**



ALGORITHM

Count the number of external and internal nodes in a Binary Tree using the pre-order traversal

Step 1: If **Root = Null** Then

Print : "Tree is Empty"

Exit

Else

Set **Pointer = Root**

[End If]

Step 2: Initialize an empty stack by pushing **Null** into it and setting the stack variable **Top** to 1

<u>Step 3:</u> Initialize the variable **Internal**=0 and **External**=0

<u>Step 4:</u> Repeat while **Pointer** ≠ **Null**

1) If **Pointer → Right** ≠ **Null** Then

   Push **Pointer → Right** onto the stack by incrementing stack variable **Top**

   [End If]

2) If **Pointer → Left** ≠ **Null** Then

   Set **Pointer = Pointer → Left**

   Set **Internal = Internal + 1**

   Else If

   Pointer → Right = Null Then

   External = External + 1

CONTINUED….

Else

Internal = Internal + 1

[End If]

Set **Pointer = Stack → Top**

Decrement the Stack's variable **Top** by 1

[End If]

[End Loop]

Step 5: Print : "Number of Internal and External nodes are:" Internal , External.

Step 6: Exit