Dayananda Sagar University Bangalore

Department of Computer Applications

Data Structures (21CA1203)

Dr. Rekha R. Nair

MODULE - 5

Trees and advance Data Structures

Basic tree concepts, Binary Trees: Properties, Representation of Binary Trees using arrays and linked lists, operations on a Binary tree , Binary Tree Traversals (recursive), Creation of binary tree from in-order and pre (post) order traversals. Adv Data Structures

Introduction to trees

- So far we have discussed mainly linear data structures – strings, arrays, lists, stacks and queues
- Now we will discuss a non-linear data structure called tree.
- Trees are mainly used to represent data containing a hierarchical relationship between elements, for example, records, family trees and table of contents.

- Consider a parent-child relationship

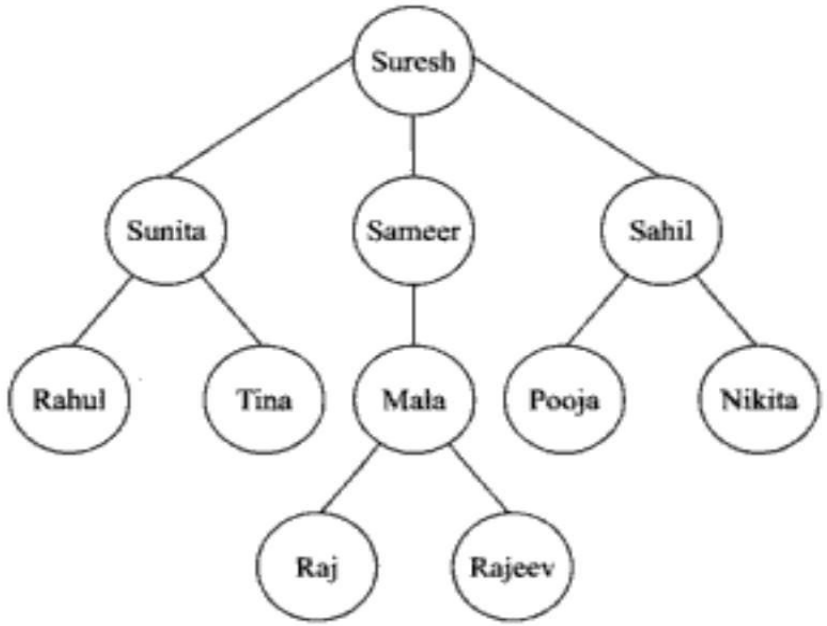


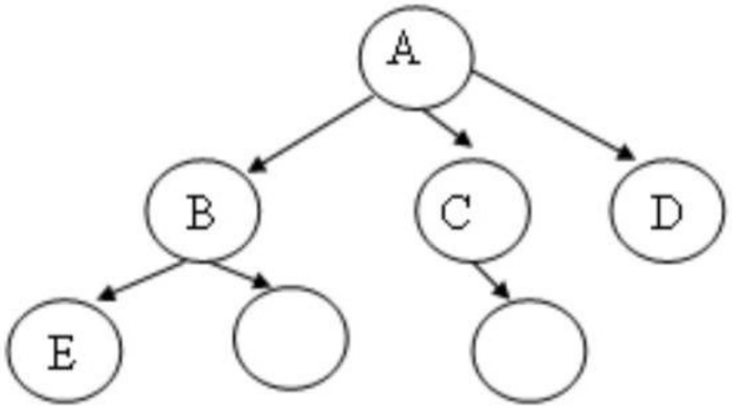
Fig. 8.1 *A Hypothetical Family Tree*

What is a Tree?

A tree is an abstract model of a hierarchical structure that consists of nodes with a parent-child relationship.

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

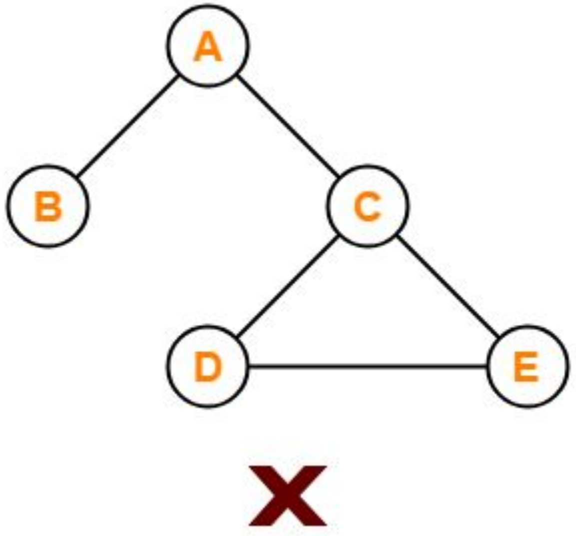
- Tree is a sequence of nodes
- There is a starting node known as a root node
- Every node other than the root has a parent node.
- Nodes may have any number of children



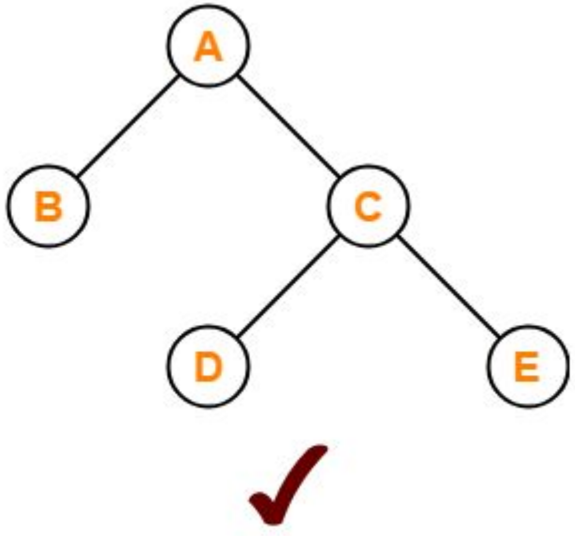
A has 3 children, B, C, D
A is parent of B

What is a Tree?

- A tree is a connected graph without any circuits.
- If in a graph, there is one and only one path between every pair of vertices, then graph is called as a tree.



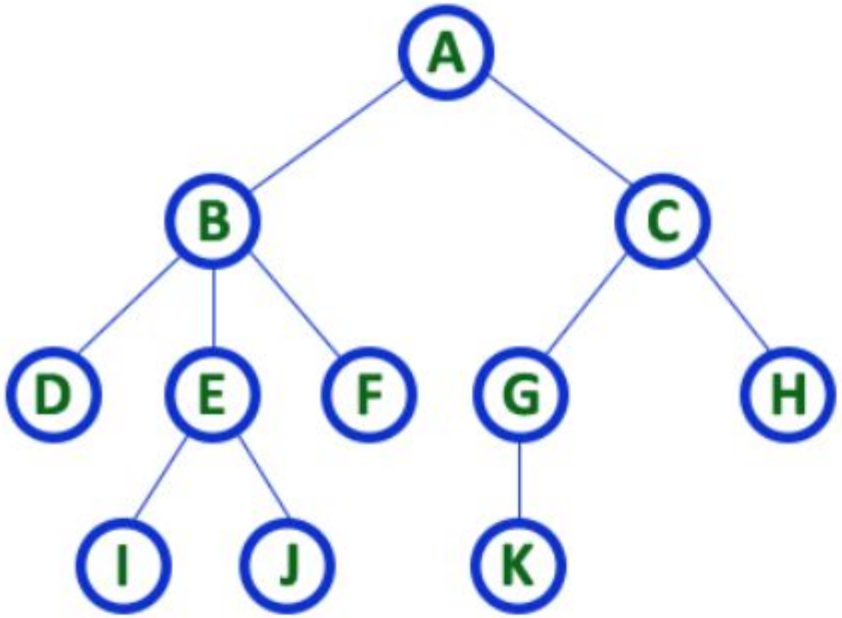
This graph is not a Tree



This graph is a Tree

What is a Tree?

TREE with 11 nodes and 10 edges

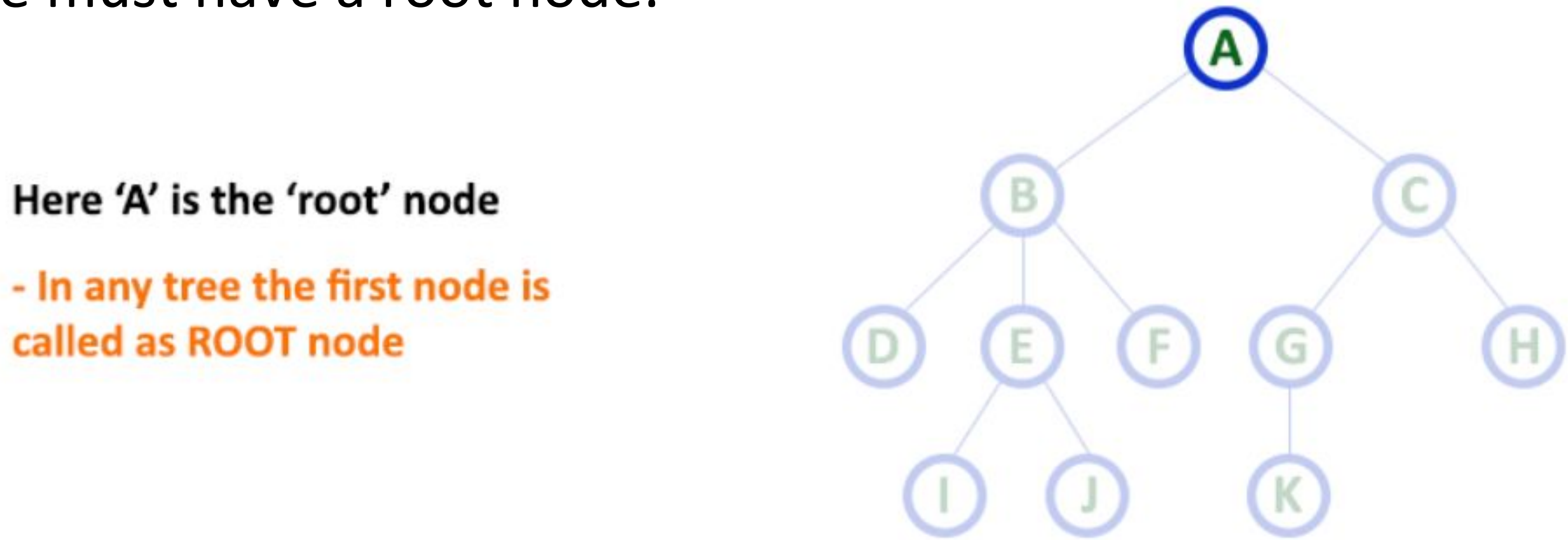


- In any tree with 'N' nodes there will be maximum of 'N-1' edges
- In a tree every individual element is called as 'NODE'

Tree-Basic Terminology

1) **Root** – Node at the top of the tree is called root.

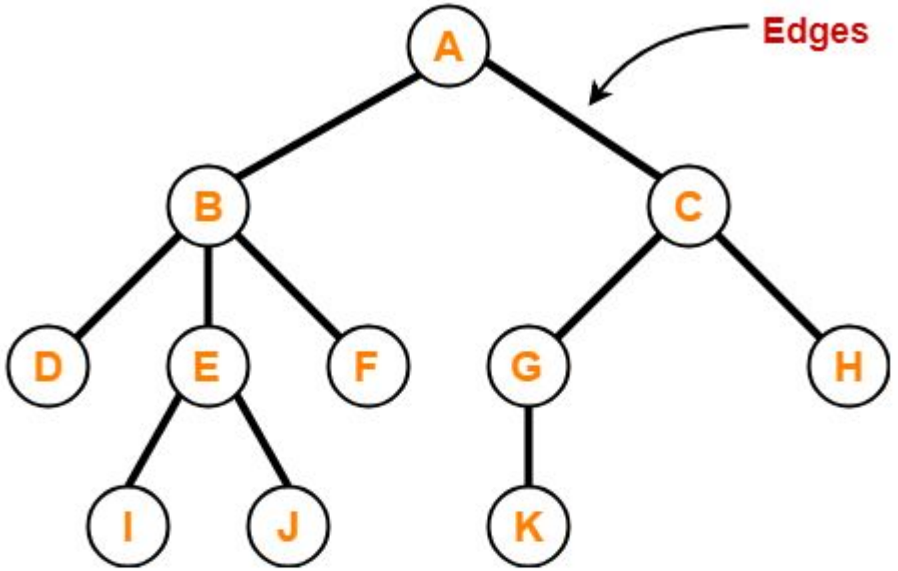
- Every tree must have a root node.



Tree-Basic Terminology

2) 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.

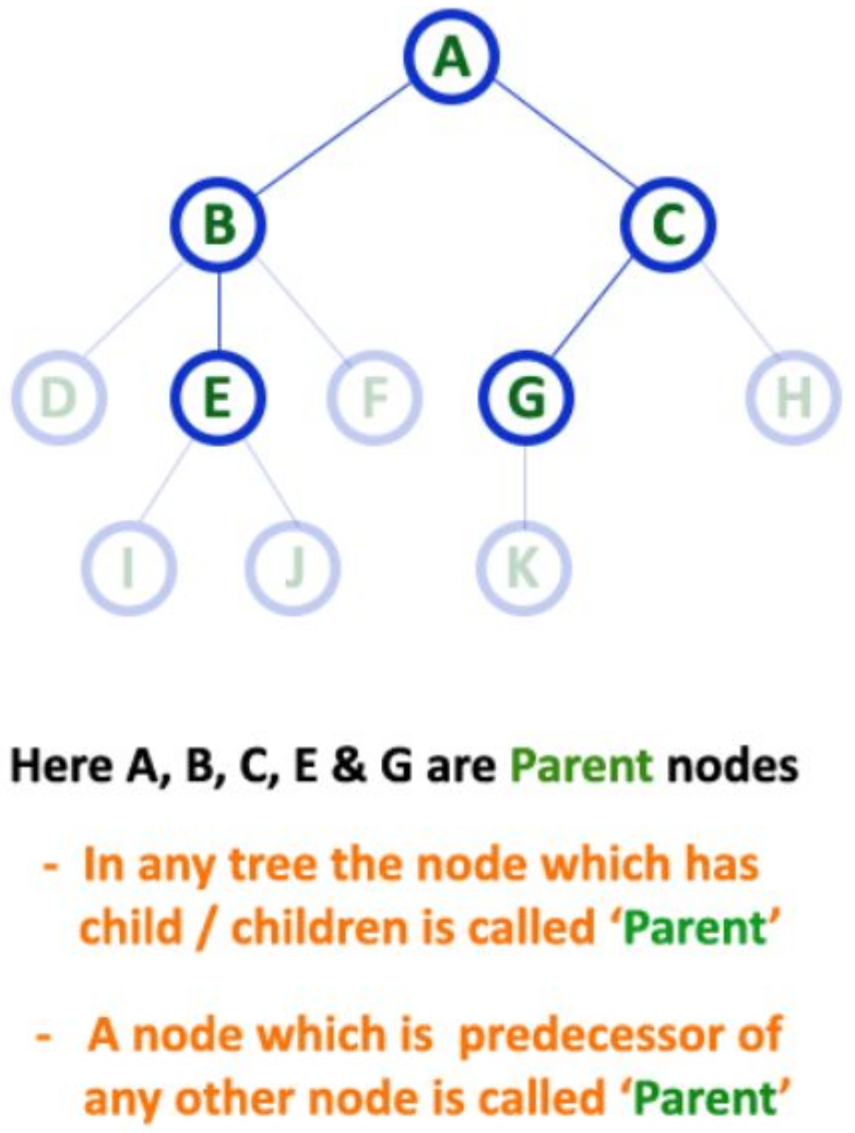


- In any tree, '**Edge**' is a connecting link between two nodes.

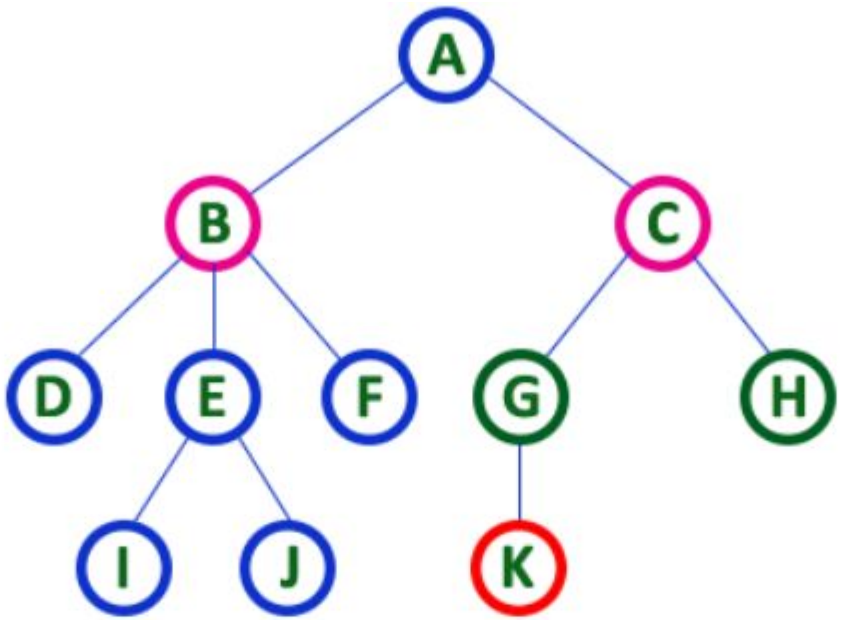
Tree-Basic Terminology

3) Parent – In a tree data structure, the node which is a predecessor of any node is called as PARENT NODE.

- The node which has a branch from it to any other node is called a parent node.
- Parent node can also be defined as "*The node which has child / children*".



Tree-Basic Terminology



4) Child – the node which is descendant of any node is called as CHILD Node.

- 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.

Here B & C are Children of A

Here G & H are Children of C

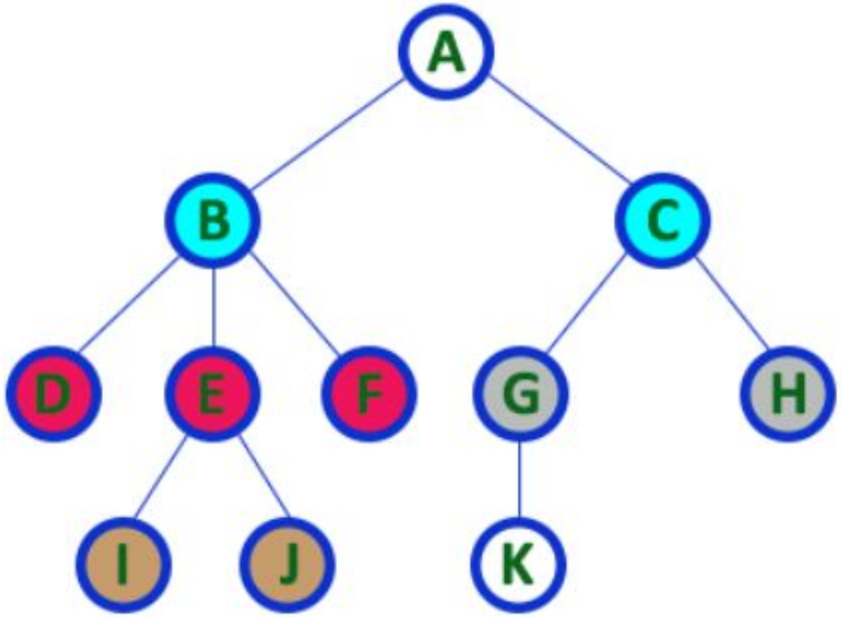
Here K is Child of G

descendant of any node is called as CHILD Node

Tree-Basic Terminology

5) Sibling – nodes which belong to same Parent are called as SIBLINGS.

- The nodes with the same parent are called Sibling nodes.



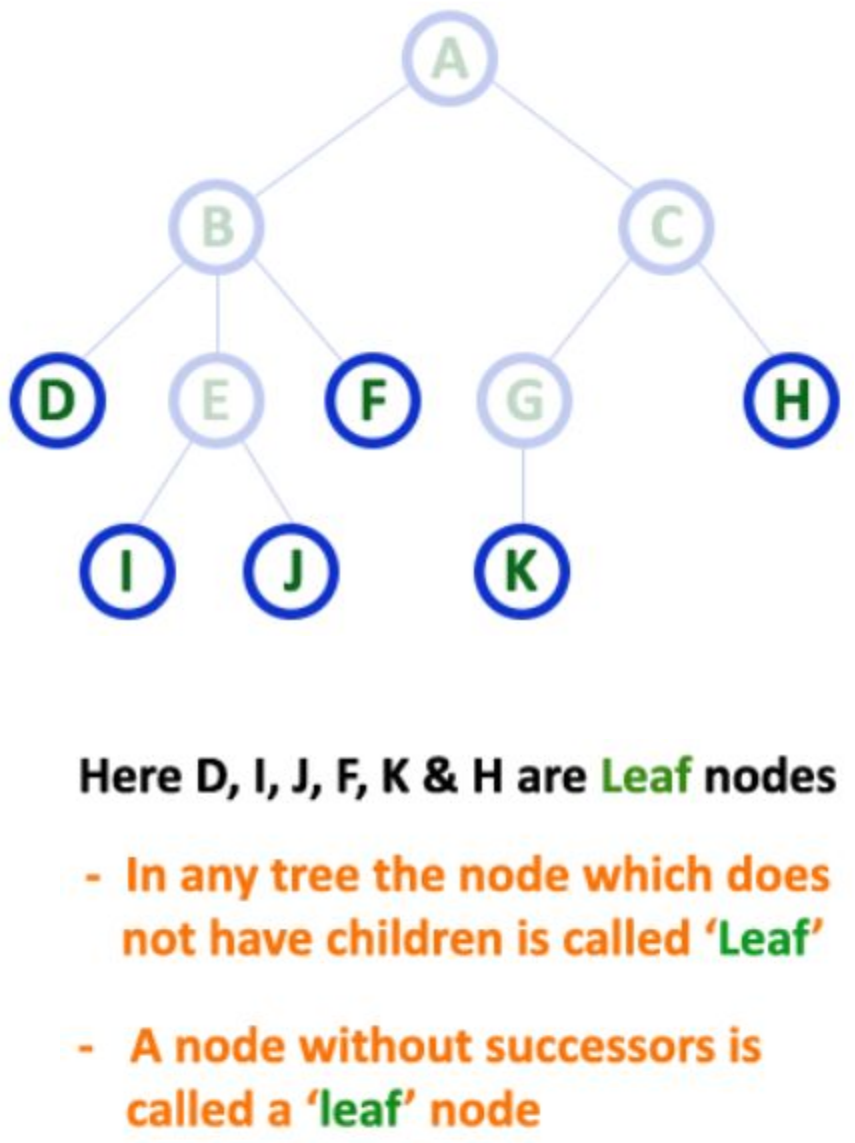
Here B & C are Siblings
Here D E & F are Siblings
Here G & H are Siblings
Here I & J are Siblings

- In any tree the nodes which has same Parent are called 'Siblings'
- The children of a Parent are called 'Siblings'

Tree-Basic Terminology

6) Leaf – the node which does not have a child is called as LEAF Node.

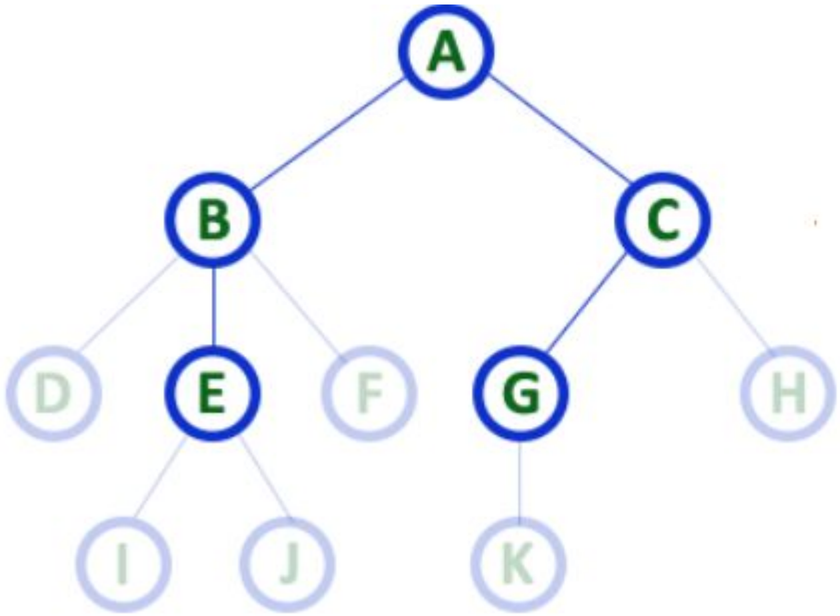
- A leaf is a node with no child.
- The leaf nodes are also called as *External Nodes*.
- In a tree, leaf node is also called as '*Terminal*' node.



Tree-Basic Terminology

7) Internal nodes – the node which has atleast one child is called as INTERNAL Node.

- Internal nodes are also called as 'Non-Terminal' nodes.



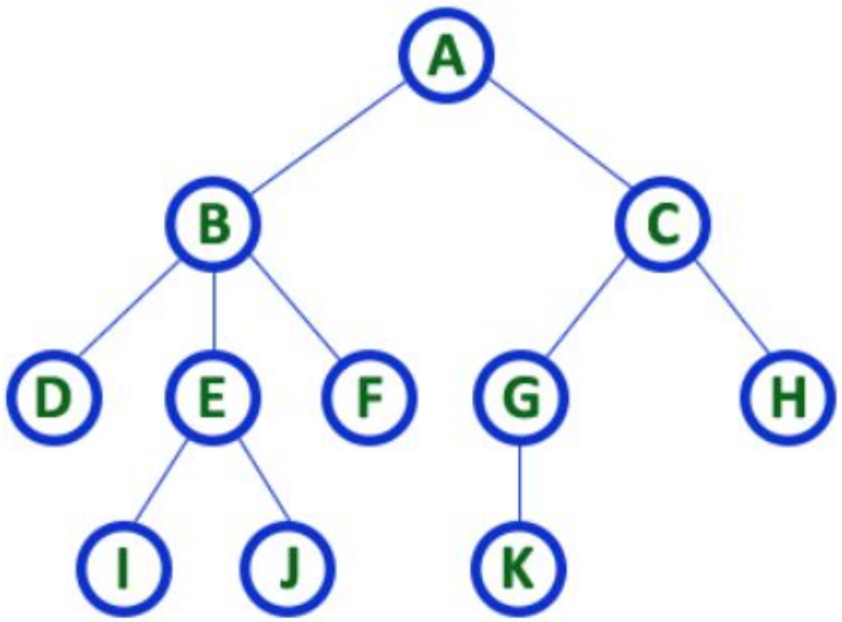
Here A, B, C, E & G are Internal nodes

- In any tree the node which has atleast one child is called 'Internal' node
- Every non-leaf node is called as 'Internal' node

Tree-Basic Terminology

8) Degree– the total number of children of a node is called as DEGREE of that Node.

- 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

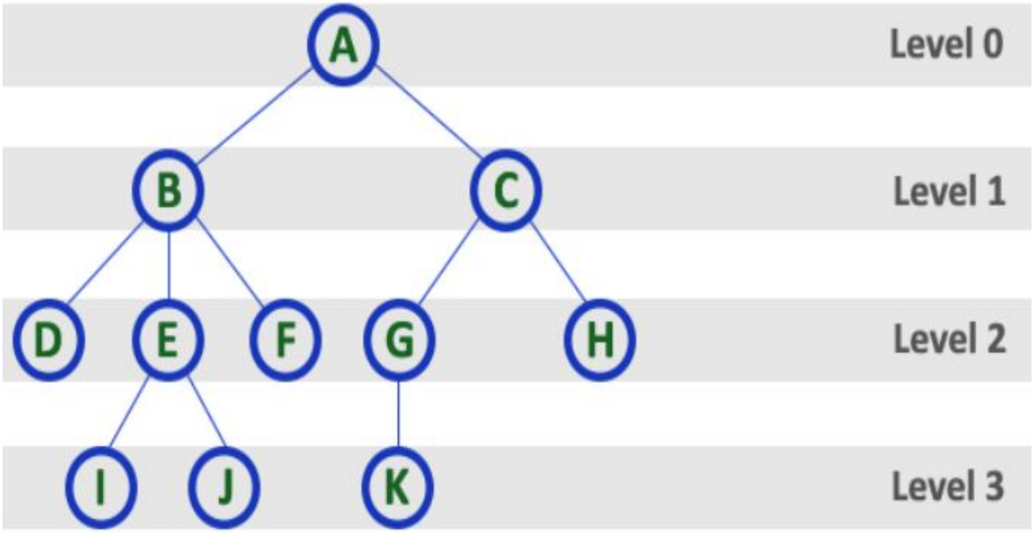
Here Degree of F is 0

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

Tree-Basic Terminology

9) Level– 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).

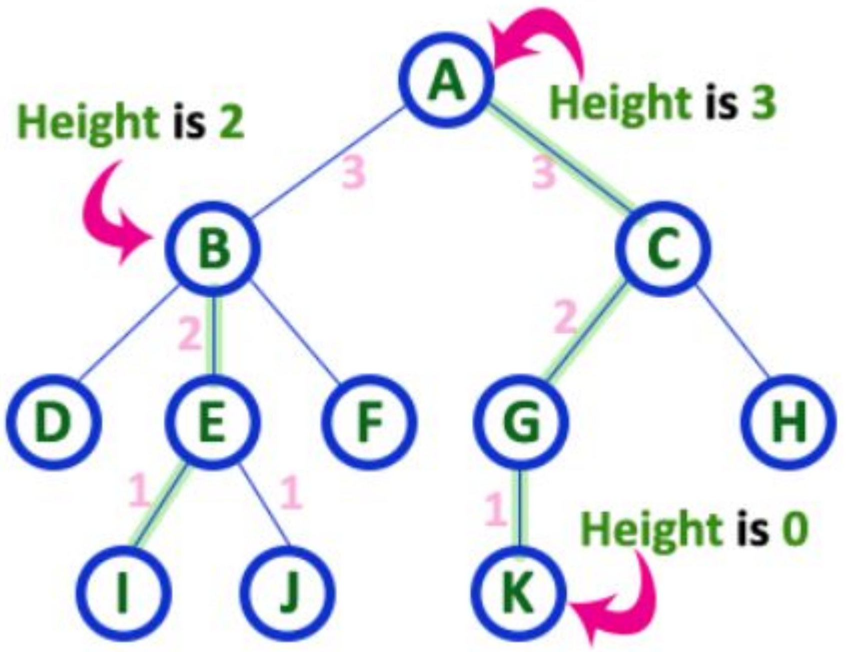
- The root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2 and so on...



Tree-Basic Terminology

10) Height – the total number of edges from leaf node to a particular node in the longest path is called as HEIGHT of that Node.

- Height of the root node is said to be height of the tree.
- Height of all leaf nodes is '0'.



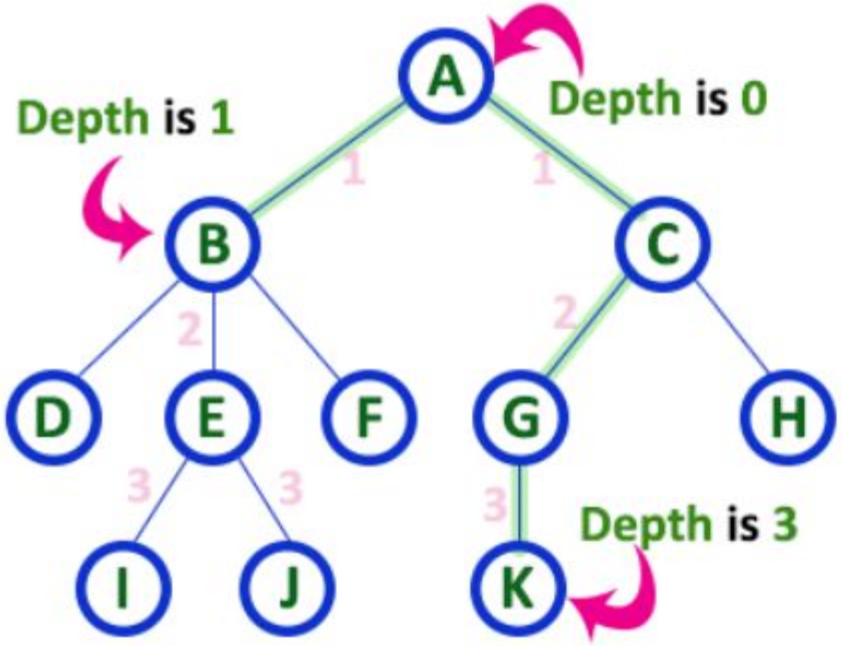
Here Height of tree is 3

- In any tree, 'Height of Node' is total number of Edges from leaf to that node in longest path.
- In any tree, 'Height of Tree' is the height of the root node.

Tree-Basic Terminology

11) Depth – the total number of egdes from root node to a particular node is called as DEPTH of that Node.

- The total number of edges from root node to a leaf node in the longest path is said to be Depth of the tree.
- The highest depth of any leaf node in a tree is said to be depth of that tree. In a tree, depth of the root node is '0'.



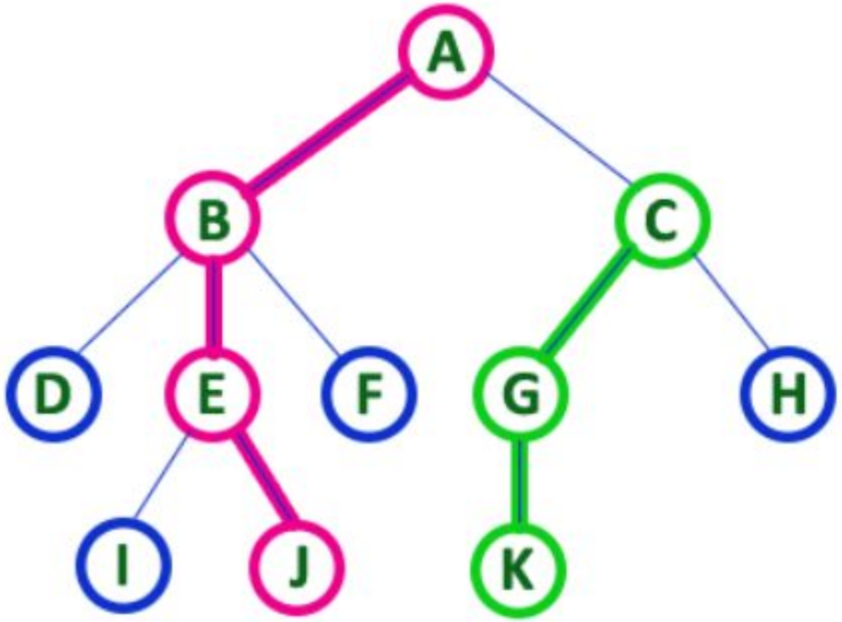
Here Depth of tree is 3

- In any tree, 'Depth of Node' is total number of Edges from root to that node.
- In any tree, 'Depth of Tree' is total number of edges from root to leaf in the longest path.

Tree-Basic Terminology

12) Path – 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 nodes in that path.
- 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

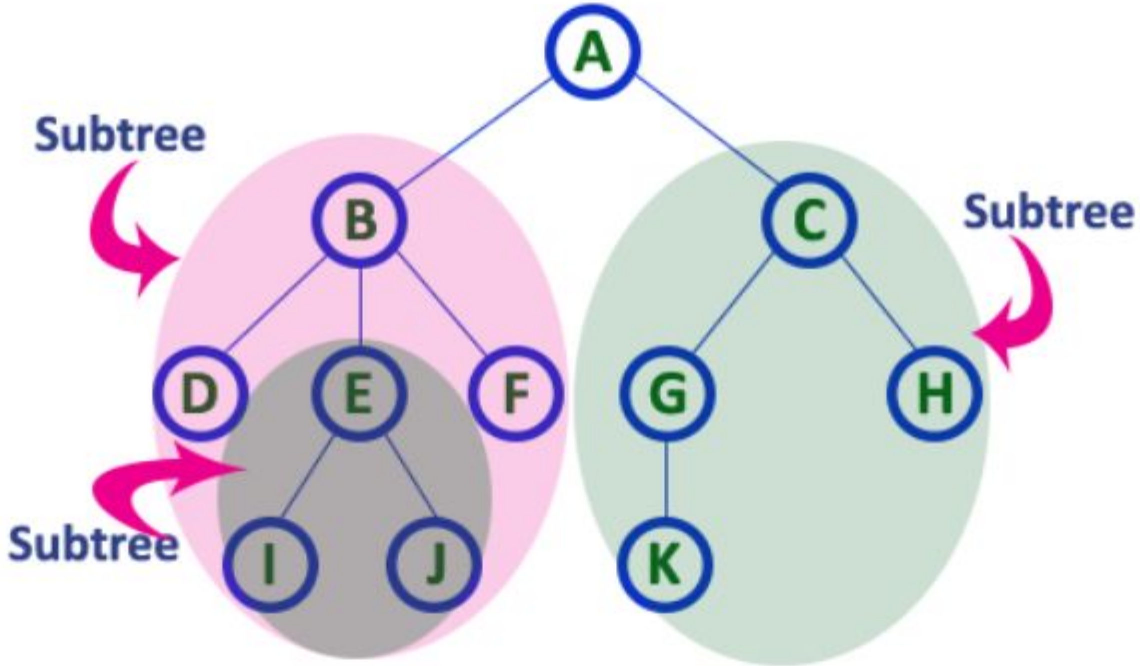
Here, 'Path' between C & K is
C - G - K

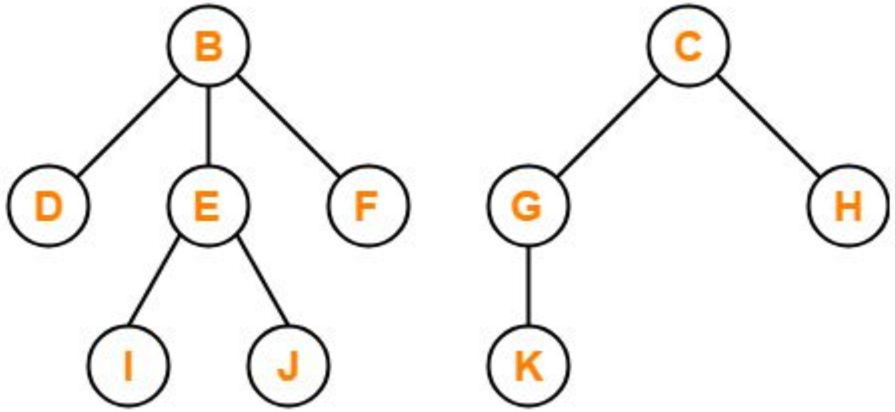
Tree-Basic Terminology

13) Subtree – each child from a node forms a subtree recursively.

- Every child node will form a subtree on its parent node.

14) **Forest:** It is a collection of disjoint trees. From a given tree if we remove its root then we get a forest.





Forest

Characteristics of Trees

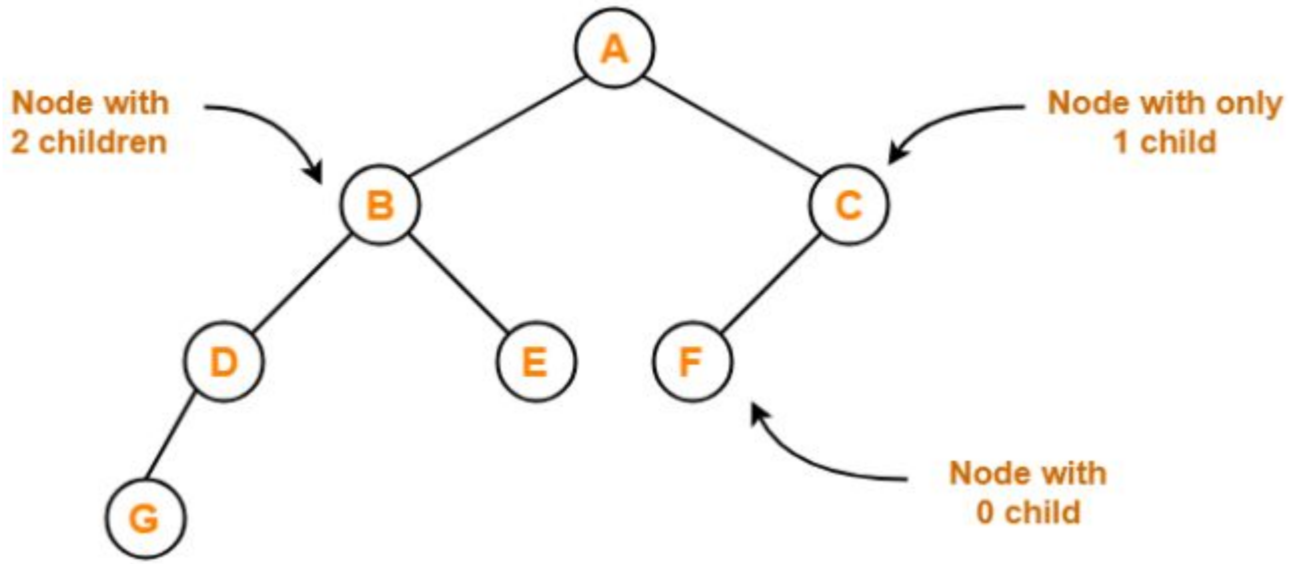
- Non-linear data structure
- Combines advantages of an ordered array
- Searching as fast as in ordered array
- Insertion and deletion as fast as in linked list

Application of Trees

- Directory structure of a file store
- Structure of an arithmetic expressions
- Used in almost every 3D video game to determine what objects need to be rendered.
- Used in almost every high-bandwidth router for storing router-tables.
- used in compression algorithms, such as those used by the .jpeg and .mp3 file- formats.

Introduction to Binary Trees

- Binary tree is a special tree data structure in which each node can have at most 2 children.
- Each node has either 0 child or 1 child or 2 children.



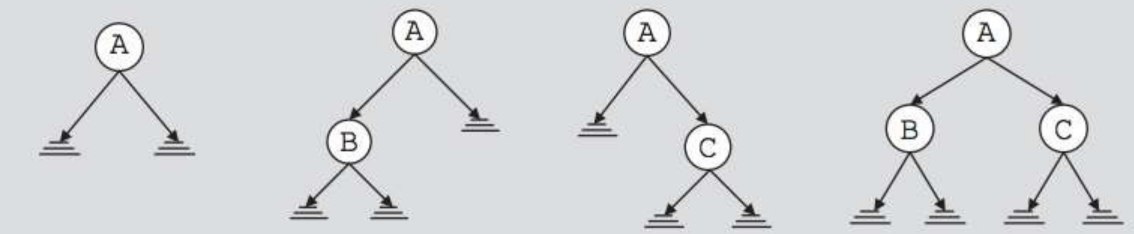
Binary Trees

- A binary tree, T, is either empty or such that

I. T has a special node called the root node

II. T has two sets of nodes LT and RT, called the left subtree and right subtree of T, respectively.

III. LT and RT are binary trees.



(a) Binary tree with one node

(b) Binary tree with two nodes

(c) Binary tree with two nodes

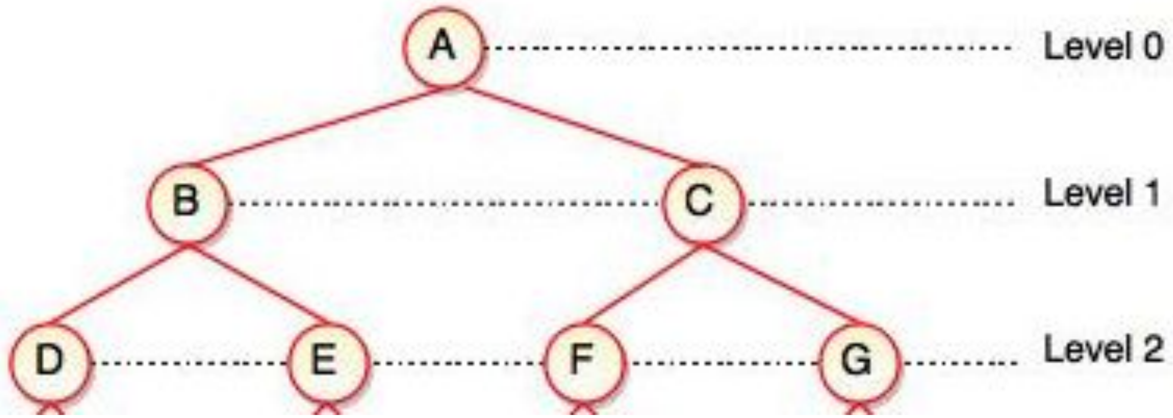
(d) Binary tree with three nodes

Properties of Binary Trees

1) **The maximum number of nodes at level 'l' of a binary tree is 2^l**

For e.g.: Maximum number of nodes at level-2 in a binary tree= 2^2= 4

Thus, in a binary tree, maximum number of nodes that can be present at level-2 = 4.

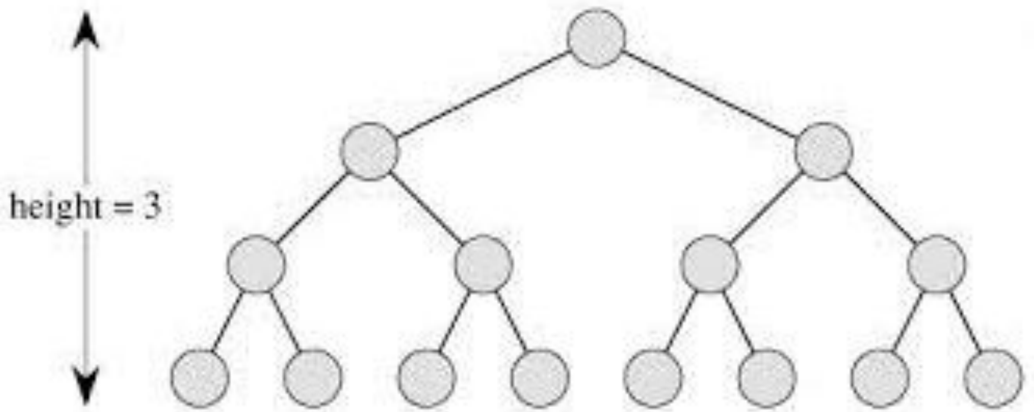


Properties of Binary Trees

2) Maximum number of nodes in a binary tree of height H =$2^{H+1} - 1$

For e.g.: Maximum number of nodes in a binary tree of height 3

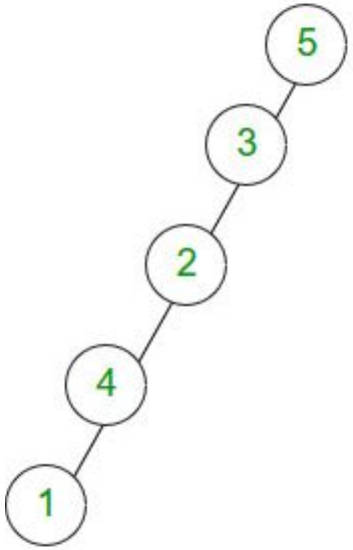
= $2^{3+1} - 1$= 16-1=15 nodes



Properties of Binary Trees

3) Minimum number of nodes in a binary tree of height H = H + 1

For e.g.: To construct a binary tree of height = 4, we need at least 4 + 1 = 5 nodes.



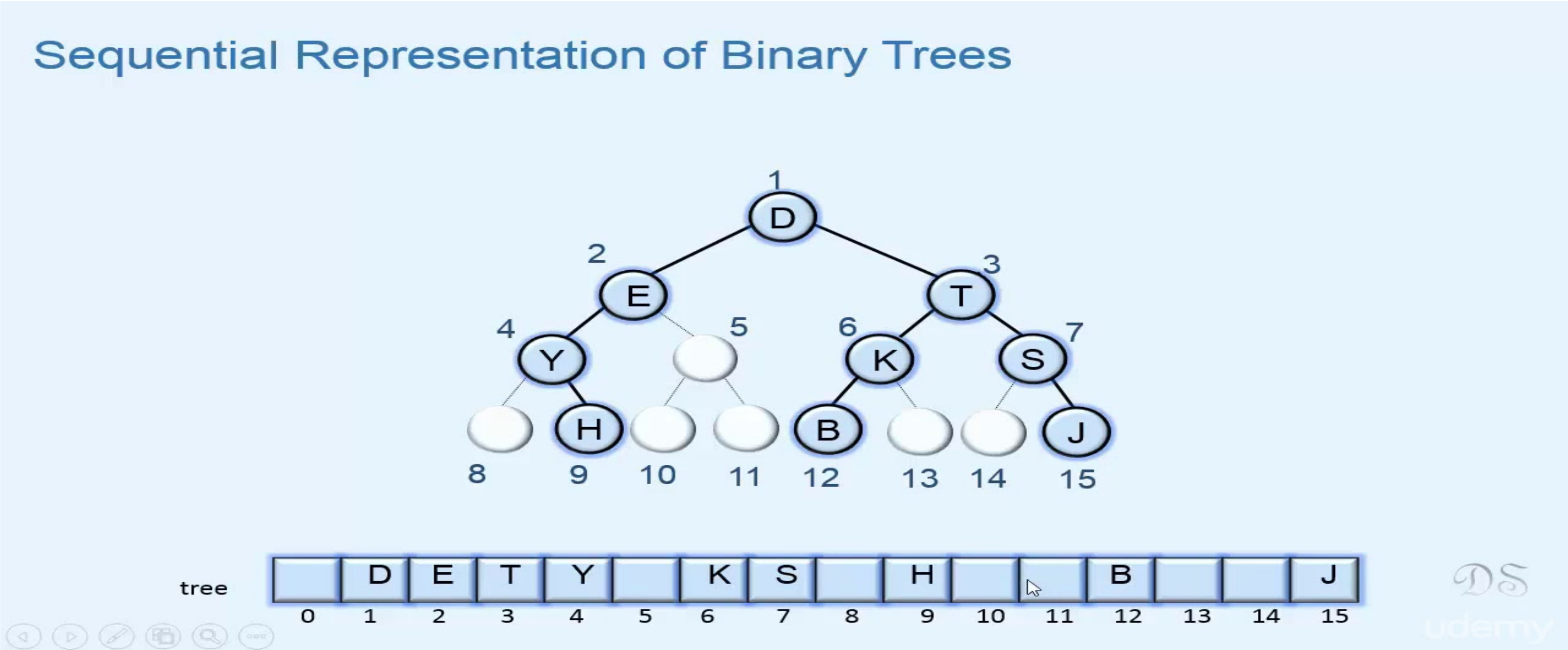
Representation of Binary Trees

1) Array Representation

2) Linked Representation

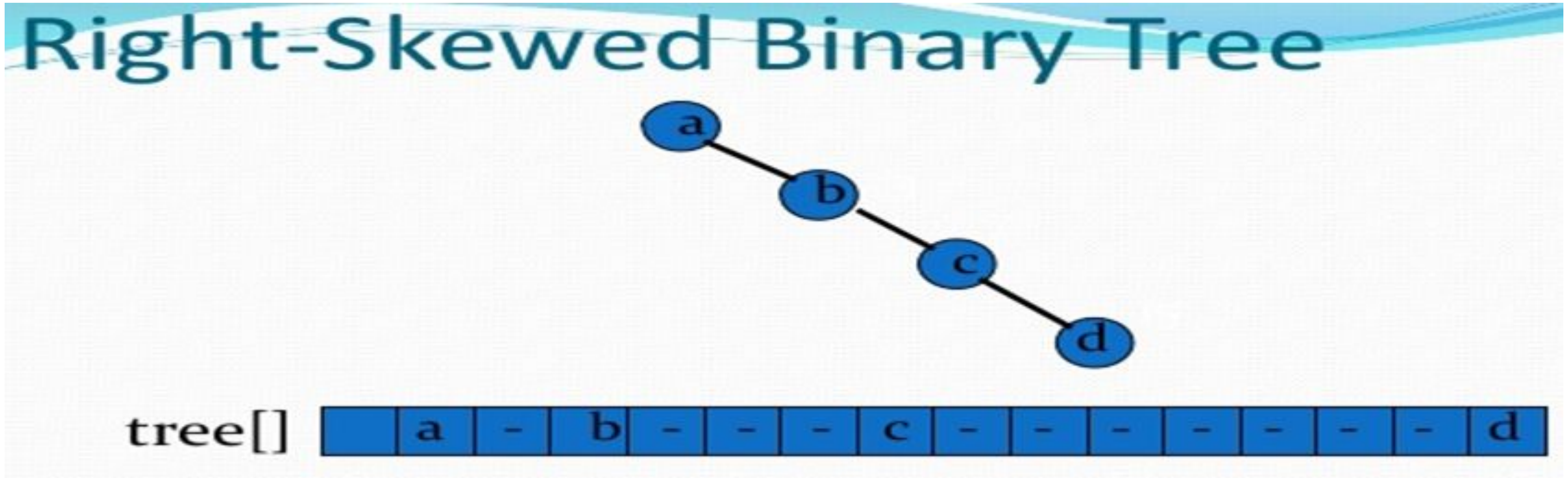
Binary Tree – Array Representation

- Number the nodes using the numbering scheme for a full binary tree. The node that is numbered i is stored in tree[i].
- if any node is stored at K position then the left child of a node is stored at index 2k and the right child is stored at index 2K + 1 and the parent of a node is stored at floor(K/2) index.



Binary Tree – Array Representation

- Advantage: Direct access to any node is possible
- Disadvantage: In case of skewed trees, array is not fully utilized leading to wastage of memory space



Binary Tree – Linked List Representation

- Use a double linked list to represent a binary tree.
- Every node consists of three fields.
  - left child address,
  - actual data
  - right child address.

- Advantages:
  - No wastage of space
  - Insertions and deletions are easier
- Disadvantages:
  - Does not provide direct access
  - Needs additional space for storing left and right subtrees

```
typedef struct node
{
        int data;
struct node *lc,*rc;
};
```

Binary Tree – Linked List Representation

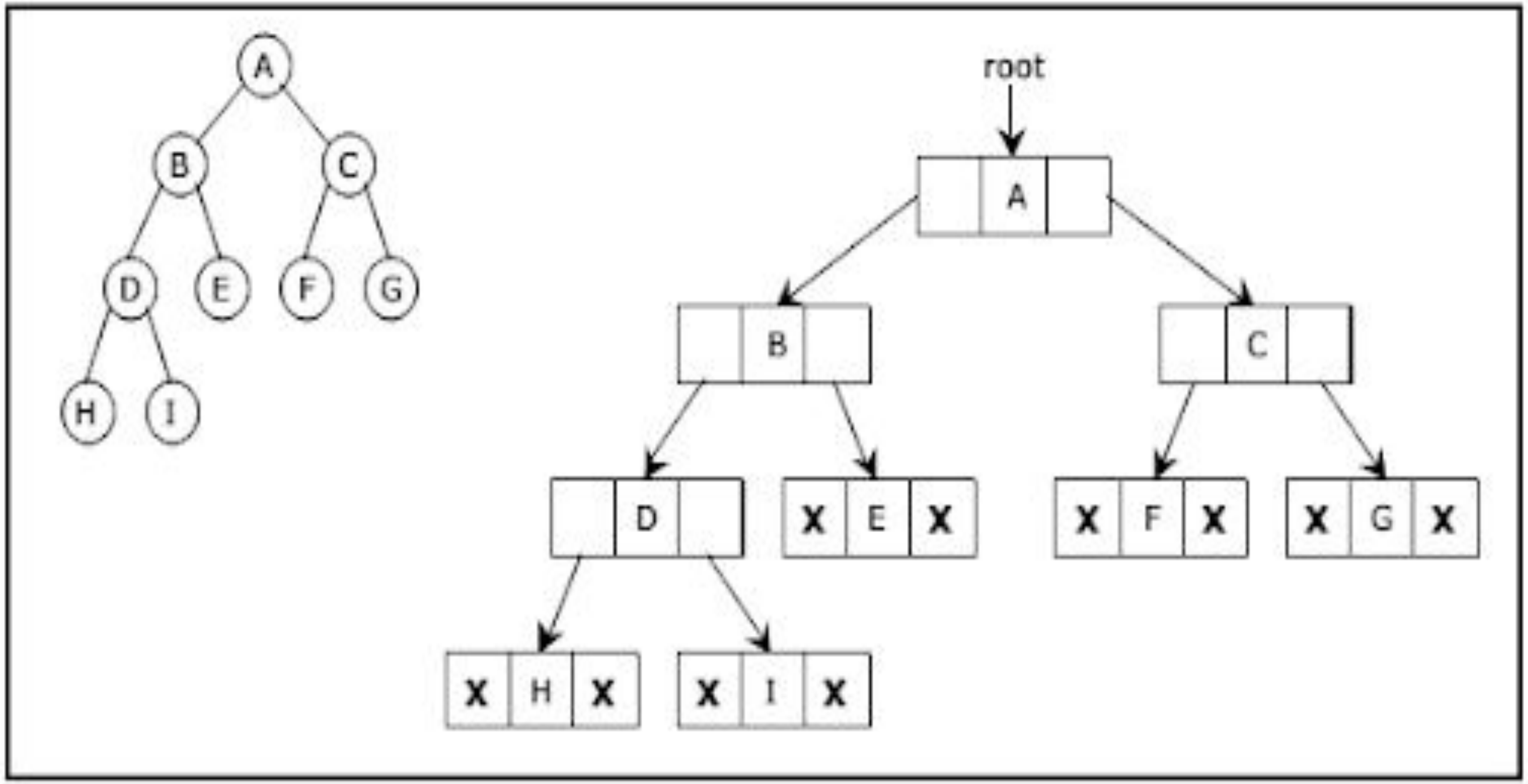
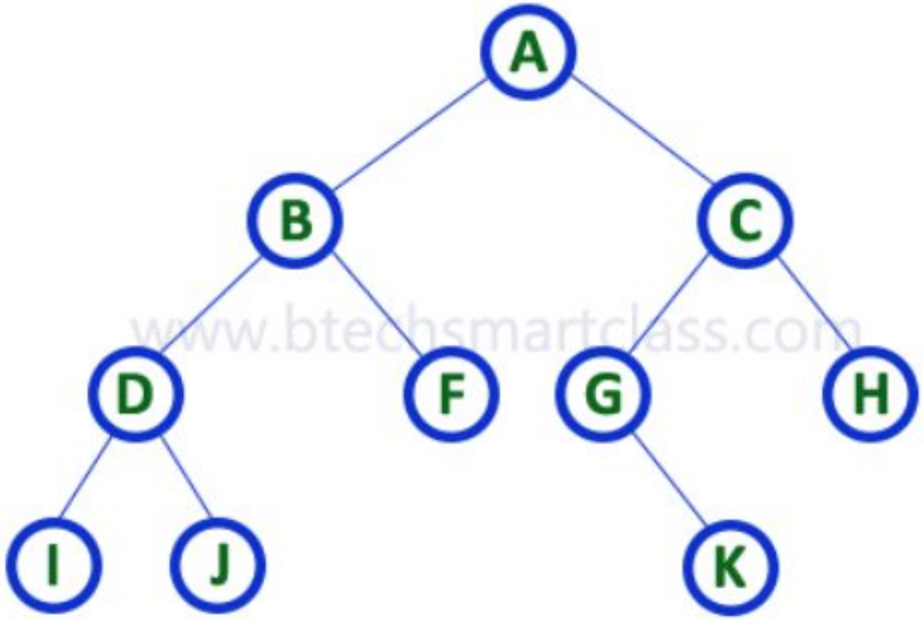
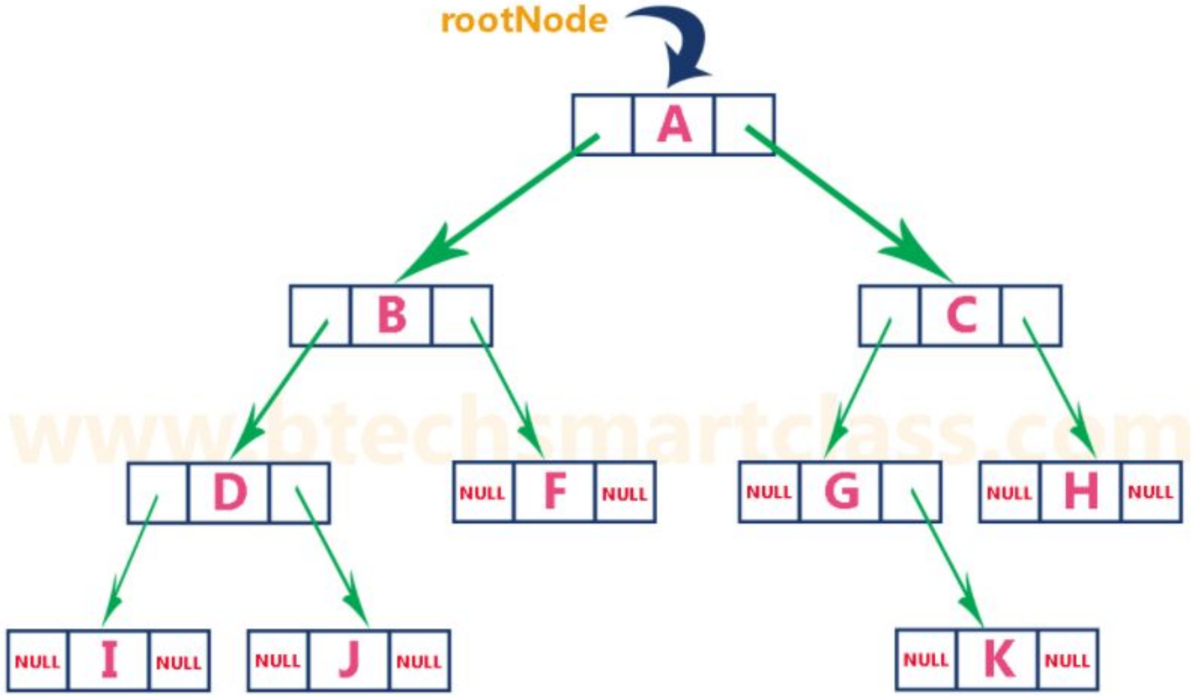


Figure 5.2.7. Linked representation for the binary tree

Binary Tree – Linked List Representation





Binary Tree – Linked List Representation

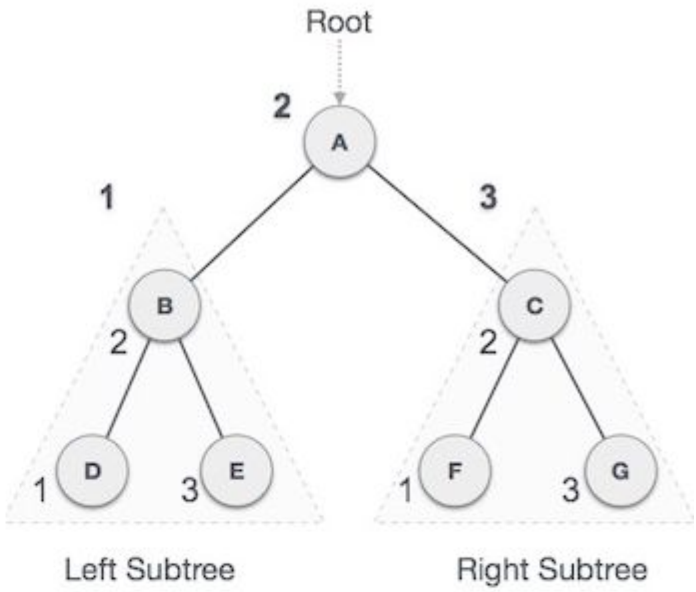
```
#include<stdio.h>

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

node *create()
{
	node *p;
	int x;
	printf("Enter data(-1 for no data):");
	scanf("%d",&x);
	if(x==-1)
		return NULL;
	p=(node*)malloc(sizeof(node));
	p->data=x;
	printf("Enter left child of %d:\n",x);
	p->left=create();

	printf("Enter right child of %d:\n",x);
	p->right=create();
return p;
}

void preorder(node *t) //address of root node is passed in t
{
	if(t!=NULL)
	{
		printf("\n%d",t->data); //visit the root
		preorder(t->left); //preorder traversal on left subtree
		preorder(t->right); //preorder traversal om right subtree
	}
}

int main()
{
	node *root;
	root=create();
	printf("\nThe preorder traversal of tree is:\n");
	preorder(root);
return 0;
}
```

Binary Tree Traversals

- Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal.
- There are three types of binary tree traversals.
- **In - Order Traversal**
- **Pre - Order Traversal**
- **Post - Order Traversal**

1. In-order Traversal (follows LDR)

- The left subtree is visited first, then the root and later the right sub-tree.



$D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$

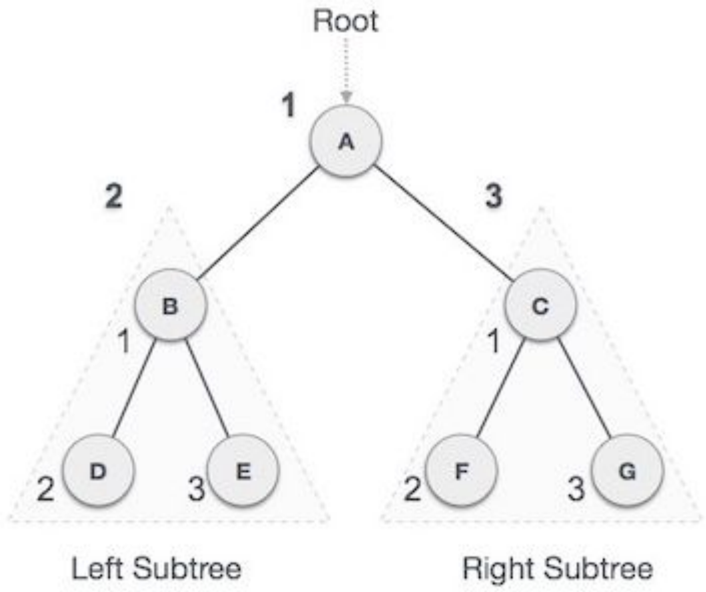
Algorithm

```
Until all nodes are traversed -
Step 1 - Recursively traverse left subtree.
Step 2 - Visit root node.
Step 3 - Recursively traverse right subtree.
```

```
void inorder(struct Node *root)
{
  if(root != NULL){
    inorder(root->left);
    printf("%d\t",root->data);
    inorder(root->right);
  }
}
```

2. Pre-order Traversal (follows DLR)

- The root node is visited first, then the left subtree and finally the right subtree.
-



A → B → D → E → C → F → G

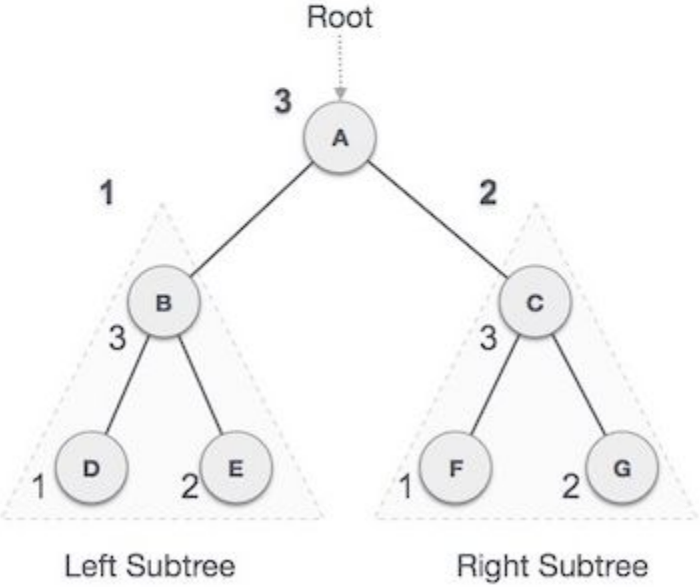
Algorithm

```
Until all nodes are traversed −
Step 1 − Visit root node.
Step 2 − Recursively traverse left subtree.
Step 3 − Recursively traverse right subtree.
```

```
void preorder(struct Node *root)
{
  if(root != NULL){
    printf("%d\t",root->data);
    preorder(root->left);
    preorder(root->right);
  }
}
```

3. Post-order Traversal (follows LRD)

- The left subtree is visited first, then the right subtree and finally the root node.
-



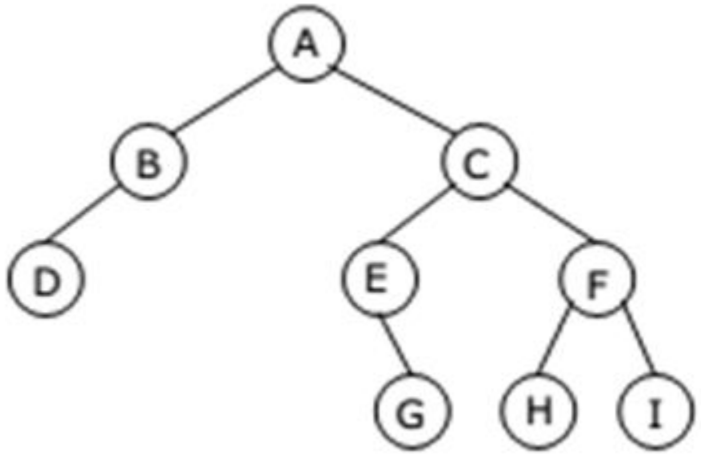
$D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$

Algorithm

```
Until all nodes are traversed –
Step 1 − Recursively traverse left subtree.
Step 2 − Recursively traverse right subtree.
Step 3 − Visit root node.
```

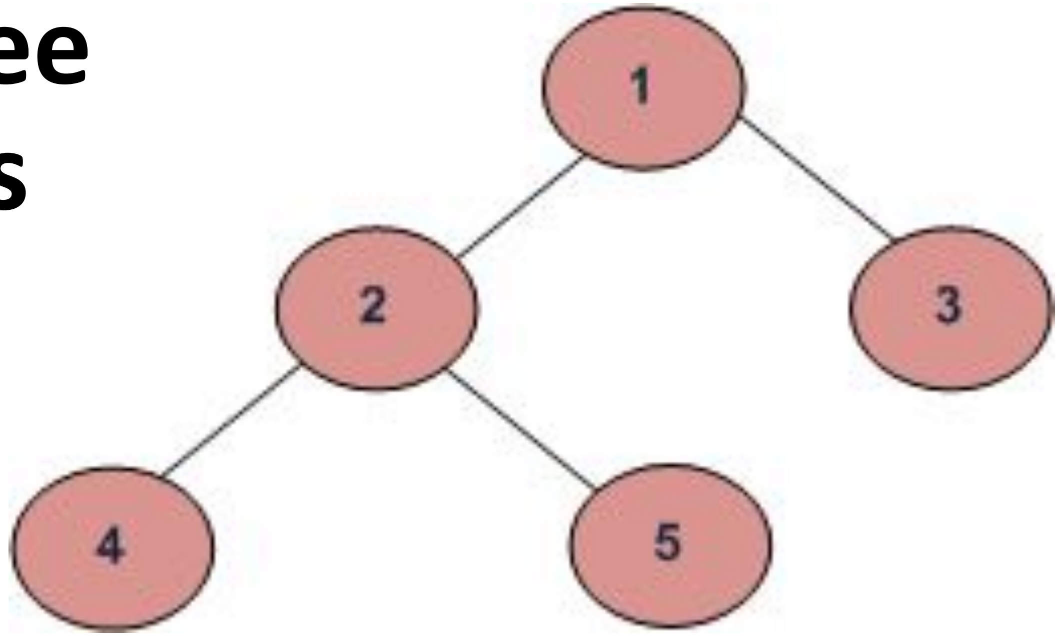
```
void postorder(struct Node *root)
{
  if(root != NULL){
    postorder(root->left);
    postorder(root->right);
    printf("%d\t",root->data);
  }
}
```

Binary Tree Traversals



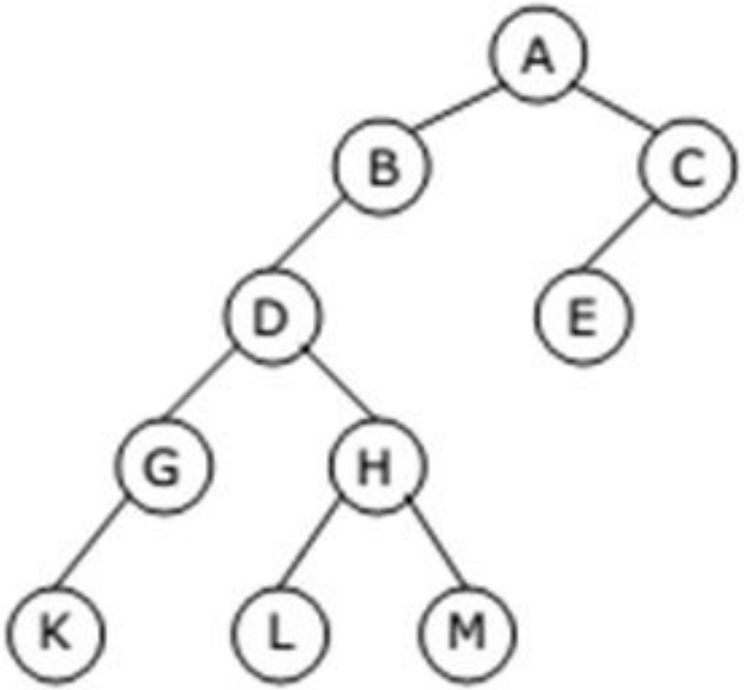
- Inorder traversal yields:
  D, B, A, E, G, C, H, F, I
- Preorder traversal yields:
  A, B, D, C, E, G, F, H, I
- Postorder traversal yields:
  D, B, G, E, H, I, F, C, A

Binary Tree Traversals



Depth First Traversals:
(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1
Breadth-First or Level Order Traversal: 1 2 3 4 5

Binary Tree Traversals



- Inorder travarsal yields:
  K, G, D, L, H, M, B, A, E, C
- Preorder traversal yields:
  A, B, D, G, K, H, L, M, C, E
- Postorder travarsal yields:
  K, G, L, M, H, D, B, E, C, A

```c
// C program for different tree traversals
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
and a pointer to right child */
struct node {
	int data;
	struct node* left;
	struct node* right;
};

/* Helper function that allocates a new node with the
given data and NULL left and right pointers. */
struct node* newNode(int data)
{
	struct node* node
		= (struct node*)malloc(sizeof(struct node));
	node->data = data;
	node->left = NULL;
	node->right = NULL;

	return (node);
}
```

```c
/* Given a binary tree, print its nodes according to the
"bottom-up" postorder traversal. */
void printPostorder(struct node* node)
{
	if (node == NULL)
		return;

	// first recur on left subtree
	printPostorder(node->left);

	// then recur on right subtree
	printPostorder(node->right);

	// now deal with the node
	printf("%d ", node->data);
}
```

```c
/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct node* node)
{
	if (node == NULL)
		return;

	/* first recur on left child */
	printInorder(node->left);

	/* then print the data of node */
	printf("%d ", node->data);

	/* now recur on right child */
	printInorder(node->right);
}
```

```c
/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct node* node)
{
	if (node == NULL)
		return;

	/* first print data of node */
	printf("%d ", node->data);

	/* then recur on left subtree */
	printPreorder(node->left);

	/* now recur on right subtree */
	printPreorder(node->right);
}
```

```
/* Driver program to test above functions*/
int main()
{
	struct node* root = newNode(1);
	root->left = newNode(2);
	root->right = newNode(3);
	root->left->left = newNode(4);
	root->left->right = newNode(5);

	printf("\nPreorder traversal of binary tree is \n");
	printPreorder(root);

	printf("\nInorder traversal of binary tree is \n");
	printInorder(root);

	printf("\nPostorder traversal of binary tree is \n");
	printPostorder(root);

	getchar();
	return 0;
}
```