DATASTRUCTURE USING C++ (MODULE I)

Prepared By

CHINJU KURIAKOSE

DEPT.OF COMPUTER SCIENCE

AL-AMEEN COLLEGE, EDATHALA

(SYLLABUS)

- Concept of Structured data - Data structure definition, Different types and classification of data structures,
- Arrays – Memory allocation and implementation of arrays in memory, array operations,
- Applications -sparse matrix representation and operations, polynomials representation and addition,
- Concept of search and sort – linear search, binary search, selection sort, insertion sort, quick sort.

DEFINITION

- Data structure is representation of the logical relationship existing between individual elements of data.
- In other words, a data structure is a way of organizing all data items that considers not only the elements stored but also their relationship to each other.

INTRODUCTION

- Data structure affects the design of both structural & functional aspects of a program.

  Program=algorithm + Data Structure

- You know that a algorithm is a step by step procedure to solve a particular function.

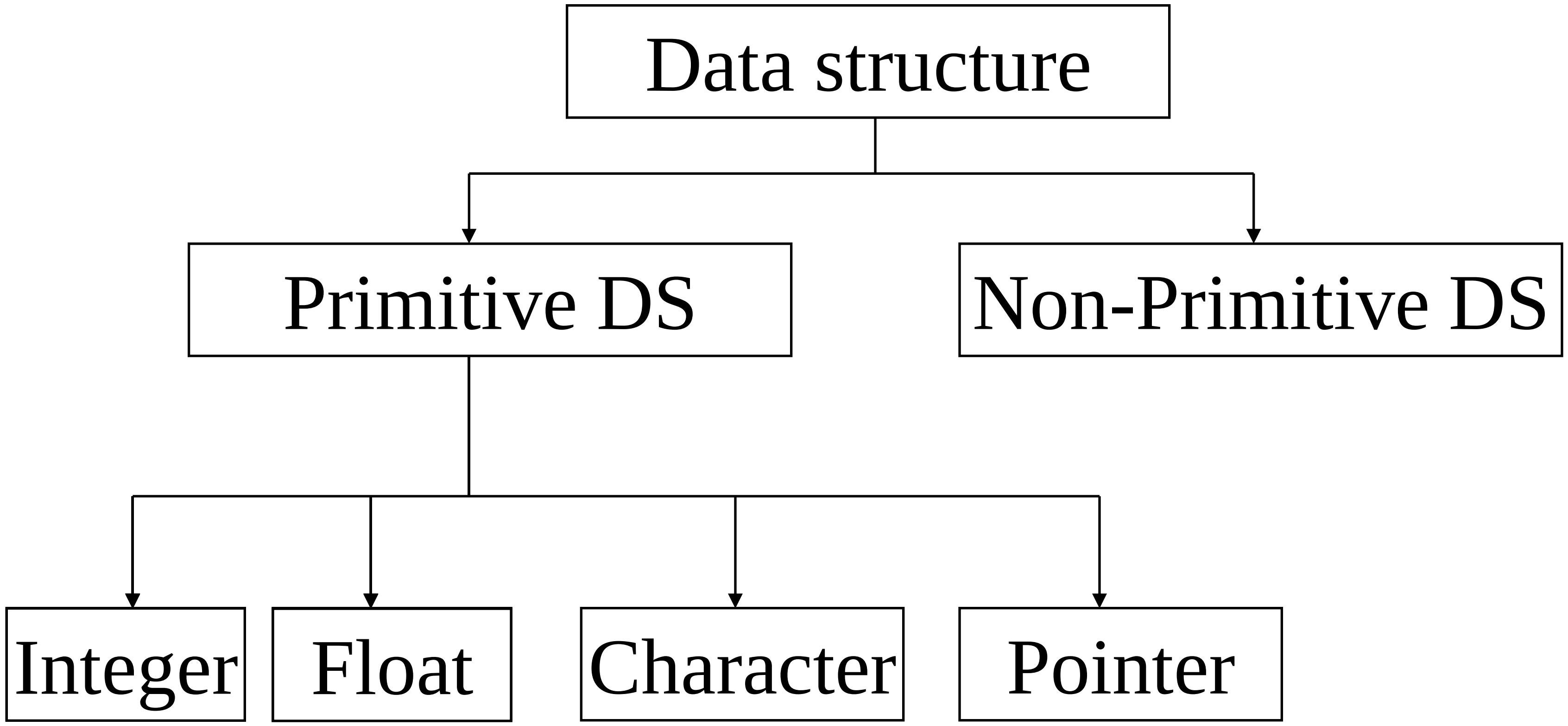
Introduction

- That means, algorithm is a set of instruction written to carry out certain tasks & the data structure is the way of organizing the data with their logical relationship retained.
- To develop a program of an algorithm, we should select an appropriate data structure for that algorithm.
- Therefore algorithm and its associated data structures from a program.

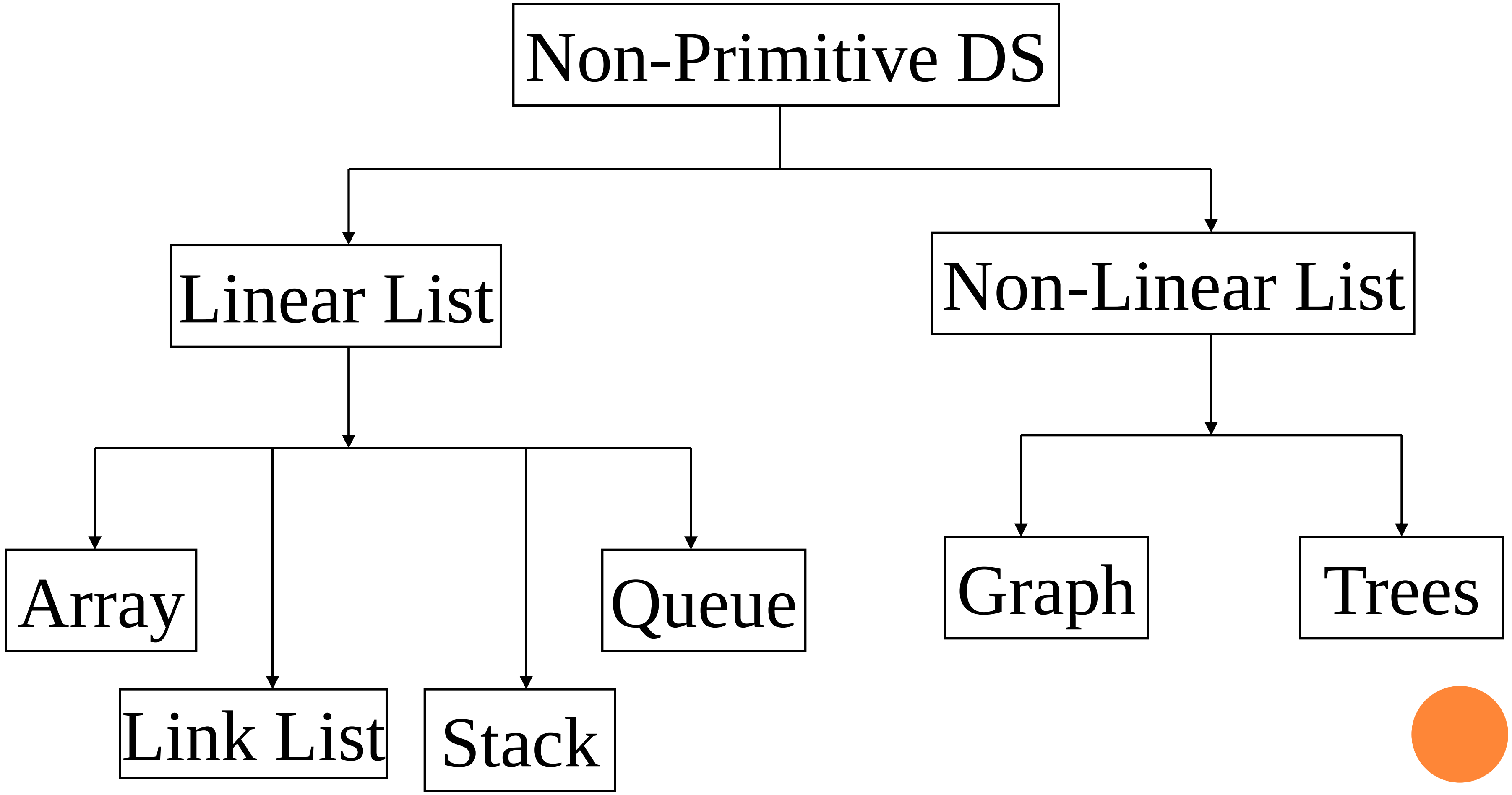
Classification of Data Structure

- Data structure are normally divided into two broad categories:
  - Primitive Data Structure
  - Non-Primitive Data Structure

Classification of Data Structure



Classification of Data Structure



Primitive Data Structure

- There are basic structures and directly operated upon by the machine instructions.
- In general, there are different representation on different computers.
- Integer, Floating-point number, Character constants, string constants, pointers etc, fall in this category.

Non-Primitive Data Structure

- There are more sophisticated data structures.
- These are derived from the primitive data structures.
- The non-primitive data structures emphasize on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items.

Non-Primitive Data Structure

- Lists, Stack, Queue, Tree, Graph are example of non-primitive data structures.
- The design of an efficient data structure must take operations to be performed on the data structure.

Non-Primitive Data Structure

- The most commonly used operation on data structure are broadly categorized into following types:
  - Create
  - Selection
  - Updating
  - Searching
  - Sorting
  - Merging
  - Destroy or Delete

DIFFERENT BETWEEN THEM

- A primitive data structure is generally a basic structure that is usually built into the language, such as an integer, a float.
- A non-primitive data structure is built out of primitive data structures linked together in meaningful ways, such as a or a linked-list, binary search tree, AVL Tree, graph etc.

DESCRIPTION OF VARIOUS DATA STRUCTURES : ARRAYS

- An array is defined as a set of finite number of homogeneous elements or same data items.
- It means an array can contain one type of data only, either all integer, all float-point number or all character.

ARRAYS

- Simply, declaration of array is as follows:

  int arr[10]

- Where int specifies the data type or type of elements arrays stores.
- “arr” is the name of array & the number specified inside the square brackets is the number of elements an array can store, this is also called sized or length of array.

Arrays

- Following are some of the concepts to be remembered about arrays:
  - The individual element of an array can be accessed by specifying name of the array, following by index or subscript inside square brackets.
  - The first element of the array has index zero[0]. It means the first element and last element will be specified as:arr[0] & arr[9]

    Respectively.

ARRAYS

- The elements of array will always be stored in the consecutive (continues) memory location.
- The number of elements that can be stored in an array, that is the size of array or its length is given by the following equation:

  (Upperbound-lowerbound)+1

Arrays

- For the above array it would be

  (9-0)+1=10,where 0 is the lower bound of array and 9 is the upper bound of array.
- Array can always be read or written through loop. If we read a one-dimensional array it require one loop for reading and other for writing the array.

ARRAYS

- For example: Reading an array

```
For(i=0;i<=9;i++)
    scanf("%d",&arr[i]);
```

- For example: Writing an array

```
For(i=0;i<=9;i++)
    printf("%d",arr[i]);
```

ARRAYS

- If we are reading or writing two-dimensional array it would require two loops. And similarly the array of a N dimension would required N loops.
- Some common operation performed on array are:
  - Creation of an array
  - Traversing an array

ARRAYS

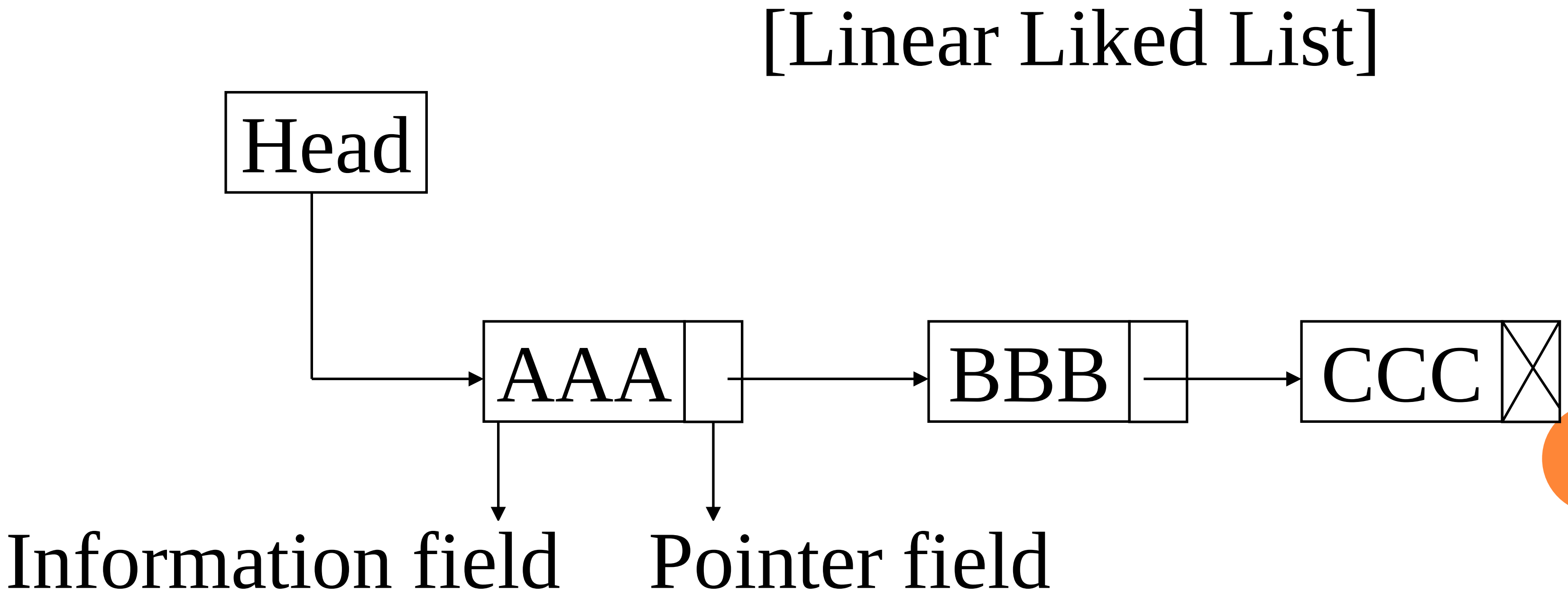
- Insertion of new element
- Deletion of required element
- Modification of an element
- Merging of arrays

LISTS

- A lists (Linear linked list) can be defined as a collection of variable number of data items.
- Lists are the most commonly used non-primitive data structures.
- An element of list must contain at least two fields, one for storing data or information and other for storing address of next element.
- As you know for storing address we have a special data structure of list the address must be pointer type.

LISTS

- Technically each such element is referred to as a node, therefore a list can be defined as a collection of nodes as show bellow:



LISTS

- Types of linked lists:
  - Single linked list
  - Doubly linked list
  - Single circular linked list
  - Doubly circular linked list

Stack

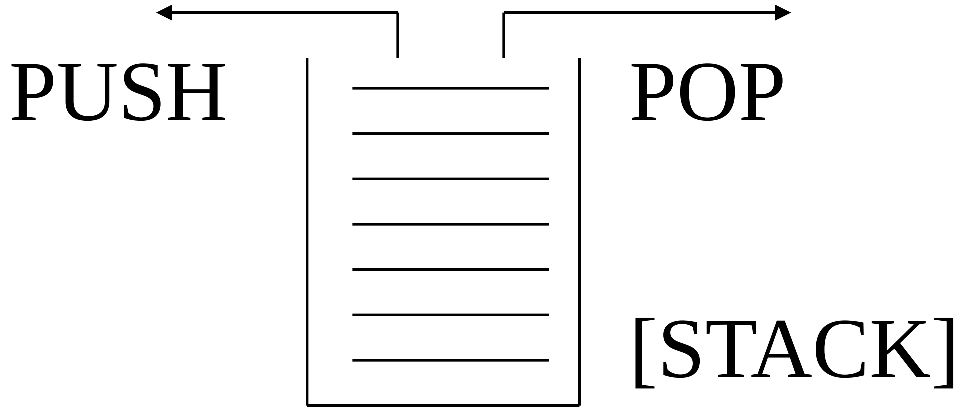
- A stack is also an ordered collection of elements like arrays, but it has a special feature that deletion and insertion of elements can be done only from one end called the top of the stack (TOP)
- Due to this property it is also called as last in first out type of data structure (LIFO).

STACK

- It could be through of just like a stack of plates placed on table in a party, a guest always takes off a fresh plate from the top and the new plates are placed on to the stack at the top.
- It is a non-primitive data structure.
- When an element is inserted into a stack or removed from the stack, its base remains fixed where the top of stack changes.

STACK

- Insertion of element into stack is called PUSH and deletion of element from stack is called POP.
- The bellow show figure how the operations take place on a stack:



Stack

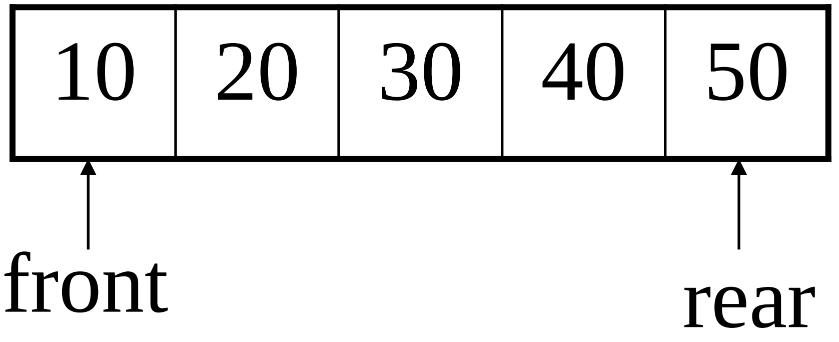
- The stack can be implemented into two ways:
  - Using arrays (Static implementation)
  - Using pointer (Dynamic implementation)

QUEUE

- Queue are first in first out type of data structure (i.e. FIFO)
- In a queue new elements are added to the queue from one end called REAR end and the element are always removed from other end called the FRONT end.
- The people standing in a railway reservation row are an example of queue.

QUEUE

- Each new person comes and stands at the end of the row and person getting their reservation confirmed get out of the row from the front end.
- The bellow show figure how the operations take place on a stack:



QUEUE

- The queue can be implemented into two ways:
  - Using arrays (Static implementation)
  - Using pointer (Dynamic implementation)

TREES

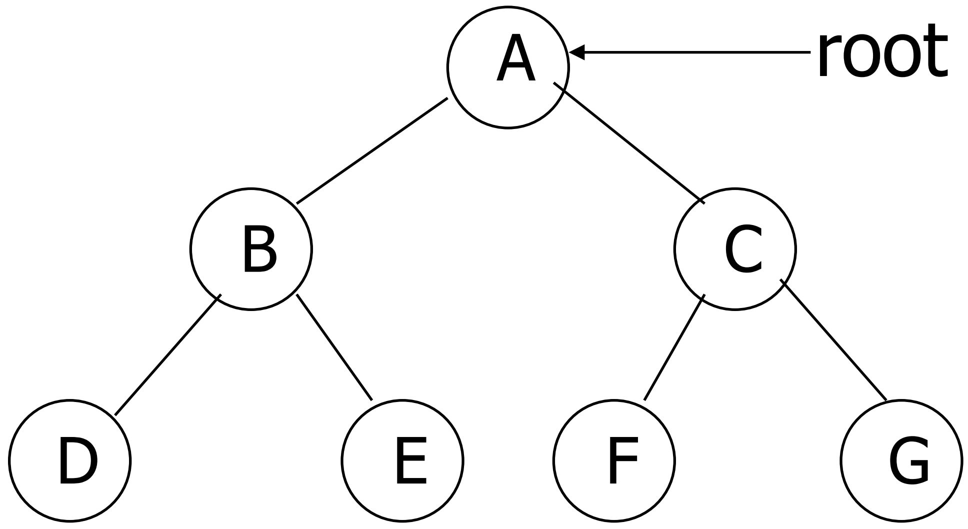
- A tree can be defined as finite set of data items (nodes).
- Tree is non-linear type of data structure in which data items are arranged or stored in a sorted sequence.
- Tree represent the hierarchical relationship between various elements.

Trees

- In trees:
- There is a special data item at the top of hierarchy called the Root of the tree.
- The remaining data items are partitioned into number of mutually exclusive subset, each of which is itself, a tree which is called the sub tree.
- The tree always grows in length towards bottom in data structures, unlike natural trees which grows upwards.

Trees

- The tree structure organizes the data into branches, which related the information.



Searching & Sorting

Searching

- The process of locating target data is known as searching
- Searching is the process of finding the location of the target among a list of object
- The two basic search techniques are the following:

1. Linear search
2. Binary search

Linear / Sequential Search

- A sequential search begins with the first available record and proceeds to the next available record repeatedly until we find the target key or conclude that it is not found
- Sequential search is also called as linear search
- Sequential search is used when the list is not sorted

Pros and Cons of Sequential Search

- Pros:

1. A simple and easy method
2. Efficient for small lists
3. Suitable for unsorted data
4. Suitable for storage structure, which does not support direct access to data, for example, magnetic tape
5. Best case is one comparison, worst case is n comparisons, and average case is (n+ 1)/2 comparisons
6. Complexity is in the order of n denoted as O(n)

- Cons:

1. Highly efficient for large data
2. Other search techniques such as binary search are found more suitable than sequential search for ordered data

Binary Search

- In binary search, the target is first searched at the mid of the list
- As the list is sorted (ascending or descending), if the target is not found at the mid, then it is searched either in upper half or in lower half
- If the list is in ascending order and if the target is smaller than the element at mid, then it is searched in upper half, else the target is searched in the lower half using binary search

Pros and Cons

- Pros:

1. Suitable for sorted data
2. Efficient for large lists
3. Suitable for storage structure that supports direct access to data
4. Time complexity is O(log2(n))

- Cons

1. Not usable for unsorted data
2. Not usable for storage structure that do not support direct access to data, for example, magnetic tape and linked list
3. Inefficient for small lists

Sorting

- Sorting is the operation of arranging the records of a table according to the key value of each record, or it can be defined as the process of converting an unordered set of elements to an ordered set of elements
- Sorting is a process of organizing data in a certain order to help retrieve it more efficiently

- The various internal sorting techniques are the following:

1. Selection sort
2. Insertion sort
3. Quick sort

Insertion Sort

- The insertion sort works just like its name suggests—it inserts each item into its proper place in the final list
- The simplest implementation of this requires two list structures: the source list and the list into which the sorted items are inserted

Selection Sort

- The selection sort algorithms construct the sorted sequence, one element at a time, by adding elements to the sorted sequence in order
- At each step, the next element to be added to the sorted sequence is selected from the remaining elements
- In this method, we sort a set of unsorted elements in two steps
- In the first step, find the smallest element in the structure
- In the second step, swap the smallest element with the element at the first position
- Then, find the next smallest element and swap with the element at the second position
- Repeat these steps until all elements get arranged at proper positions

Quick Sort

- Quick sort is based on divide-and-conquer strategy
- Quick sort is thus in-place, divide-and-conquer based massively recursive sort technique
- This technique reduces unnecessary swaps and moves the element at great distance in one move
- The recursive algorithm consists of four steps:
- If there is one or less element in the array to be sorted, return immediately
- Pick an element in the array to serve as a 'pivot' usually the left-most element in the list)
- Partition the array into two parts—one with elements smaller than the pivot and the other with elements larger than the pivot by traversing from both the ends and performing swaps if needed
- Recursively repeat the algorithm for both partitions

Module II (Syllabus)

- Stacks – Concepts, organization and operations on stacks using arrays (static), examples, Applications -Conversion of infix to postfix and infix to prefix, postfix evaluation, subprogram calls and execution,Multiple stacks representation.
- Queues - Concepts, organization and operations on queues, examples.Circular queue – limitations of linear queue, organization and operations on circular queue. Double ended queue, Priority queue.

Stack

- Stacks in real life: stack of books, stack of plates
- Add new items at the top
- Remove an item at the top
- Push(X) – insert X as the top element of the stack
- Pop() – remove the top element of the stack and return it.

Stack Push and Pop Operation Using Arrays

- Check that stack is full or not if full print “stack full stack”
- If not full then increase top count and add data in array at top ‘s index

- A queue is a linear structure for which items can be only inserted at one end and removed at another end.
- A queue is a FIFO (First-In First-Out) structure. There are two basic types of queues:
- Simple queue
- Circular queue

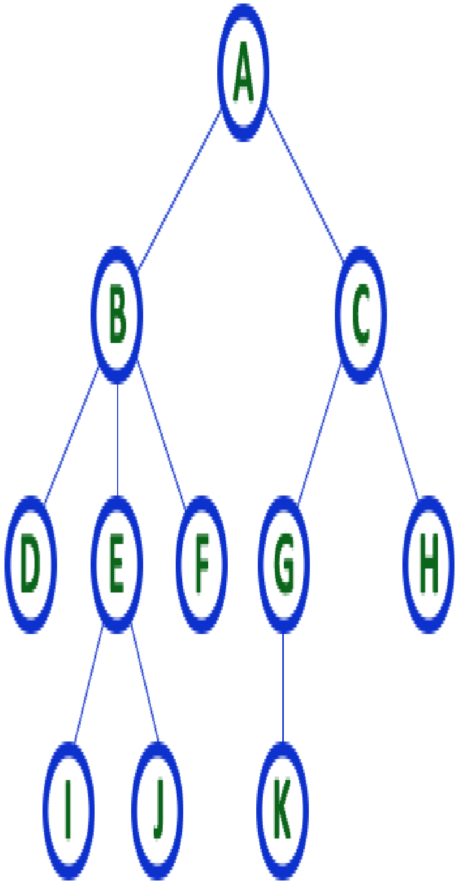
Tree (syllabus)

- Trees - Concept of recursion, trees, tree terminology, binary trees, representation of binary trees, strictly binary trees, complete binary tree, extended binary trees, creation and operations on binary tree, binary search trees, Creation of binary search tree, tree traversing methods – examples, binary tree representation of expressions.

TREE

- Tree is a non-linear data structure in which items are arranged in a sorted sequence.
- It is used to represent hierarchical relationship existing among several data items.
- The graph theoretic definition of tree is : it is a finite set of one or more data items (node) such that

1. There is a special data item called the **root** of the tree
2. And its remaining data items are partitioned into number of mutually exclusive subsets, each of which is itself a tree. And they are called **subtree**.



TREE with 11 nodes and 10 edges

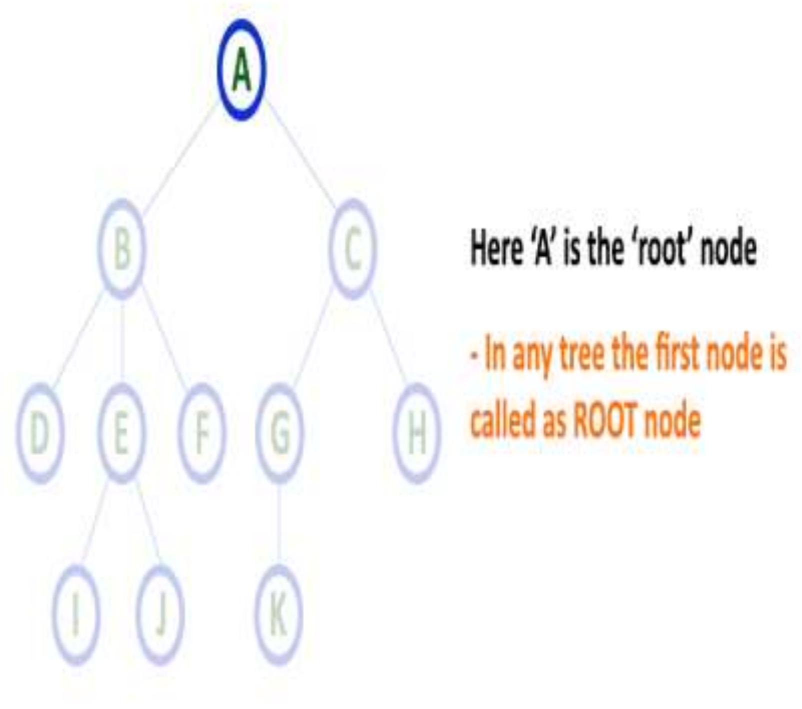
- In any tree with 'N' nodes there will be maximum of 'N-1' edges
- In a tree every individual element is called as 'NODE'

- Natural trees grows upward from the ground into the air.
- Tree in data structure grows downwards from top to bottom

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.

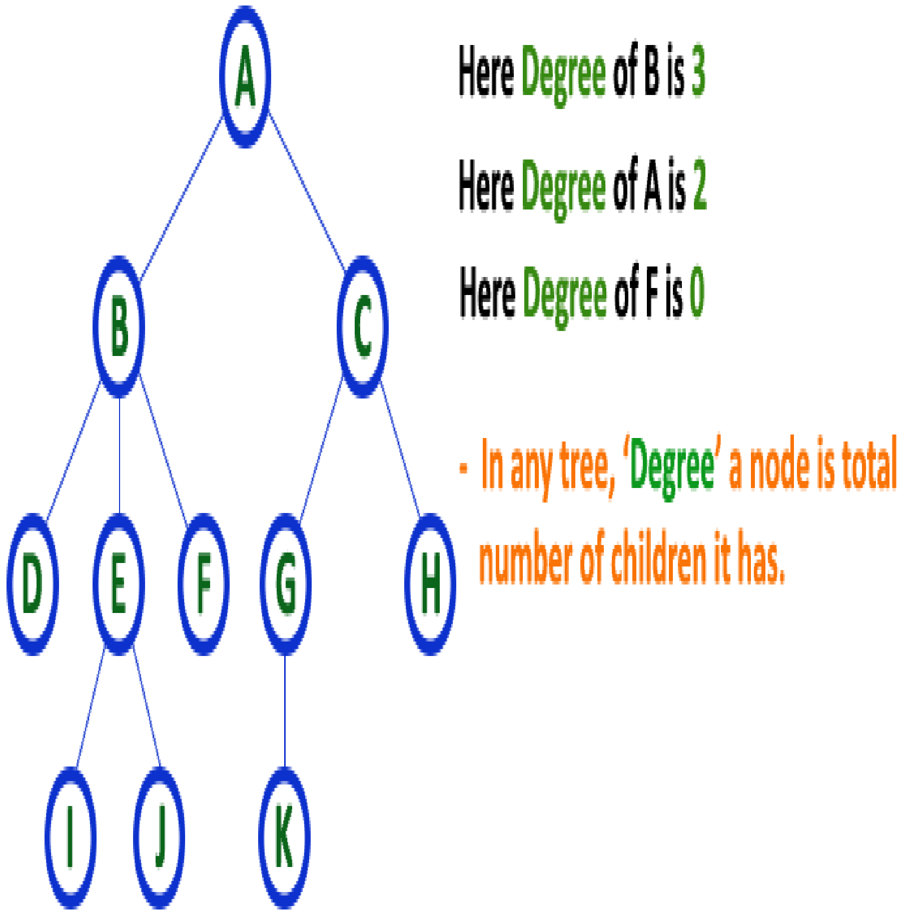

A
B
C
D
E
F
G
H
I
J
K
Here 'A' is the 'root' node
- In any tree the first node is called as ROOT node

NODE

- Each data item in a tree is called a node.
- It is a basic structure in a tree
- It specifies the data information and links to other data items.
- There are 13 nodes in the above

DEGREE OF A NODE

- 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.

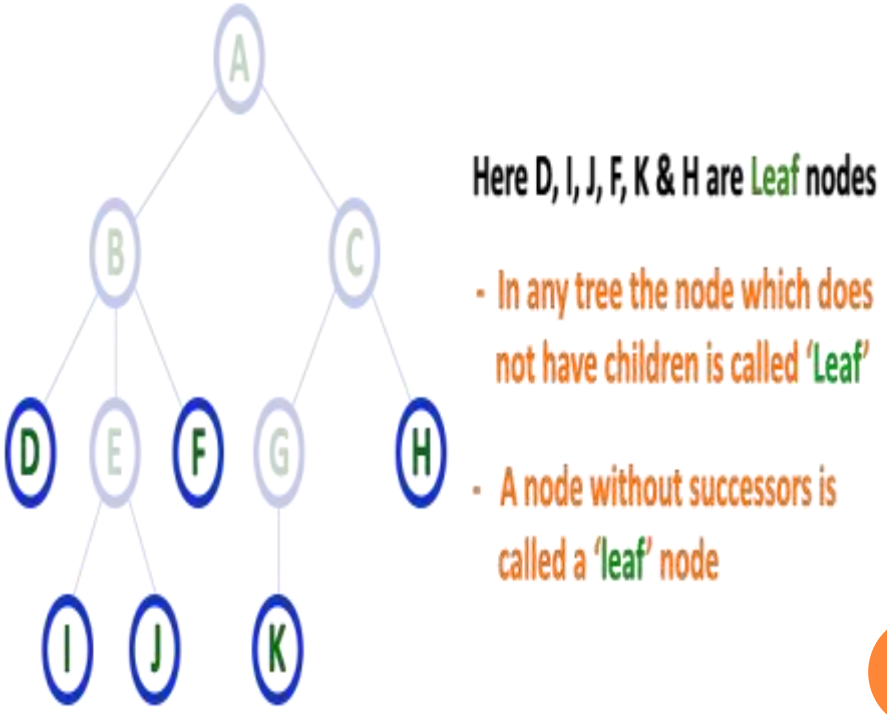


DEGREE OF A TREE

- The highest degree of a node among all the nodes in a tree is called as '**Degree of Tree**‘
- In the above tree the node B has dgree 3, this value is the maximum. So , the degree of the tree of the above is 3

TERMINAL NODE

- A node with degree zero is called a **terminal node** or a **leaf** or **external node.**

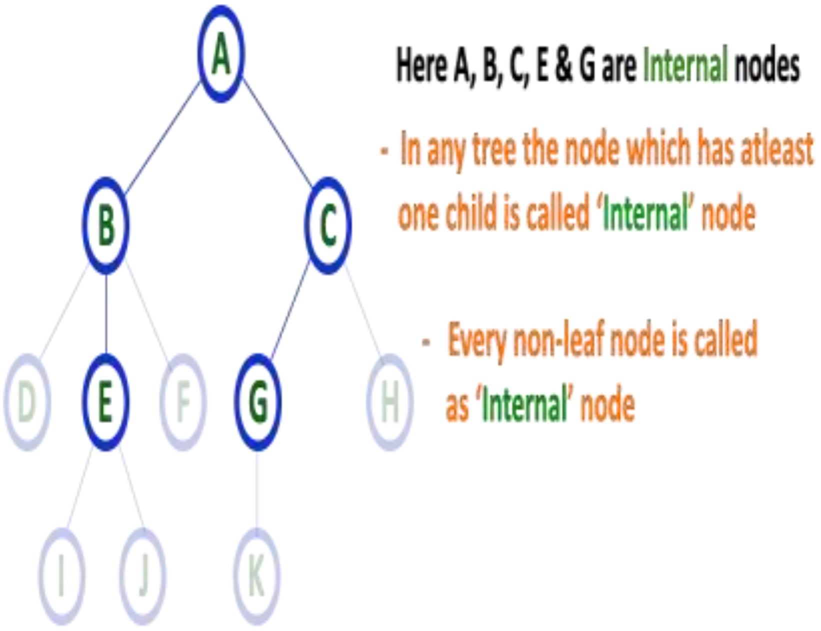


Here D, I, J, F, K & H are Leaf nodes

- In any tree the node which does not have children is called 'Leaf'
- A node without successors is called a 'leaf' node

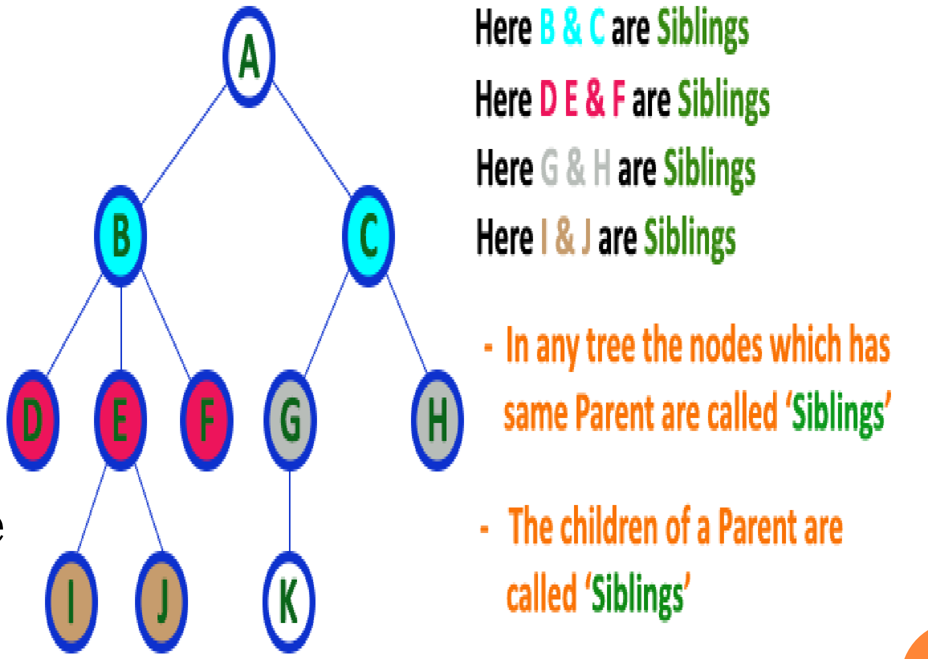
NON – TERMINAL NODE

- In a tree data structure, the node which has atleast one child is called as **NON – TERMINAL NODE** Or
- **INTERNAL Node**. In simple words, any node whose degree is not zero.


A
B
C
D
E
F
G
H
I
J
K
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

SIBLINGS

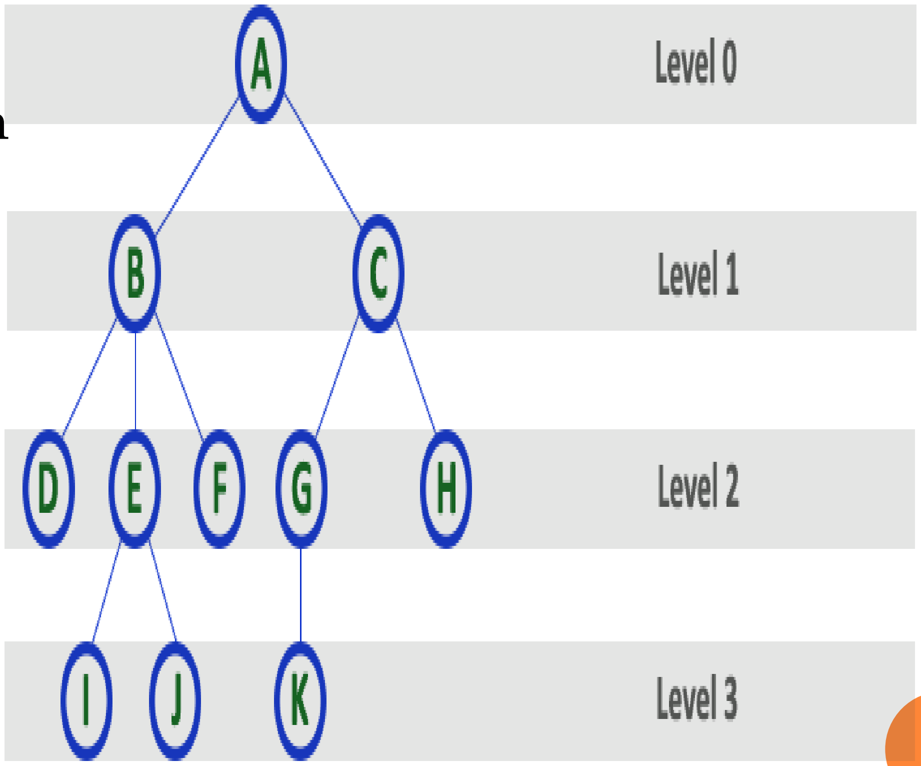
- In a tree data structure, nodes which belong to same Parent are called as **SIBLINGS** .
- In simple words, the children node of a given parent node are called as Sibling nodes.



A
B
C
D
E
F
G
H
I
J
K
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'

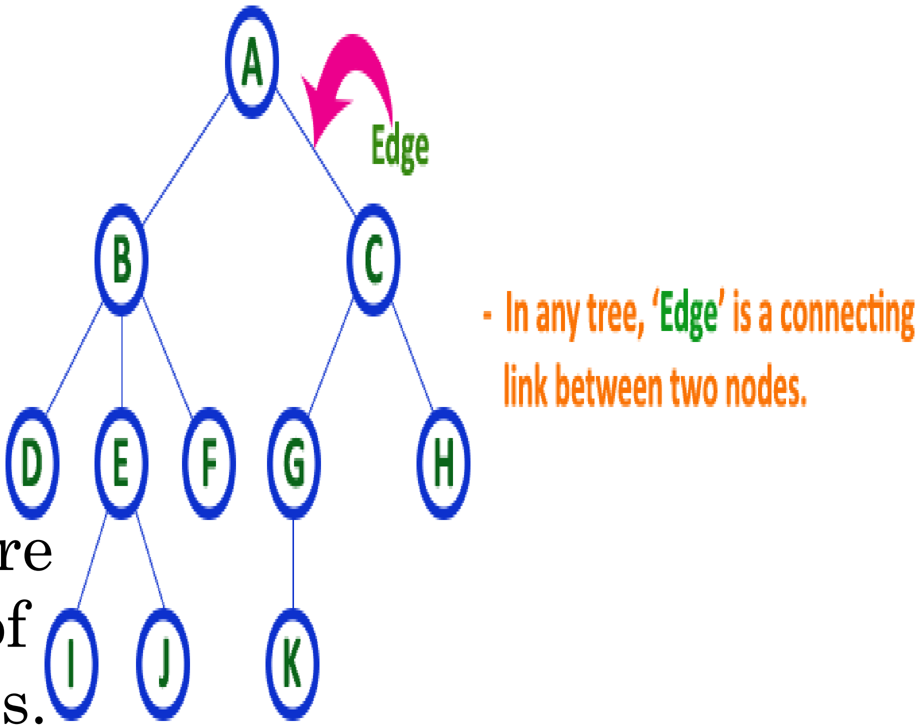
LEVEL

- 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).



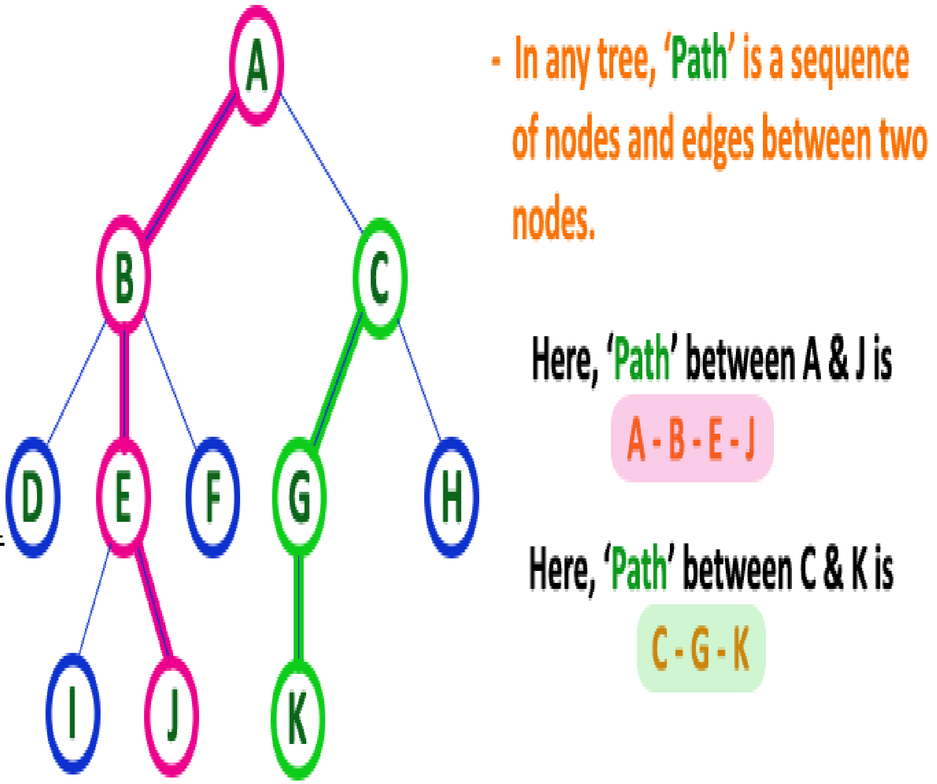
EDGE

- 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.


A
B
C
D
E
F
G
H
I
J
K
Edge
- In any tree, 'Edge' is a connecting link between two nodes.

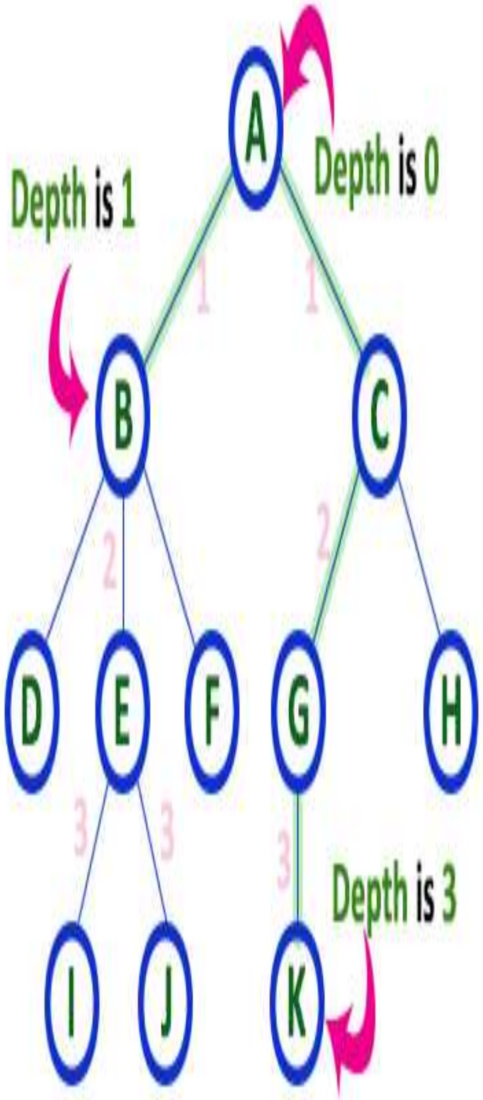
PATH

- 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** 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
A
B
C
D
E
F
G
H
I
J
K

DEPTH

- It is the maximum level of any node in a given tree
- In a tree data structure, the total number of egdes from root node to a particular node is called as **DEPTH** of that Node.
- 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**.
- In simple words, 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

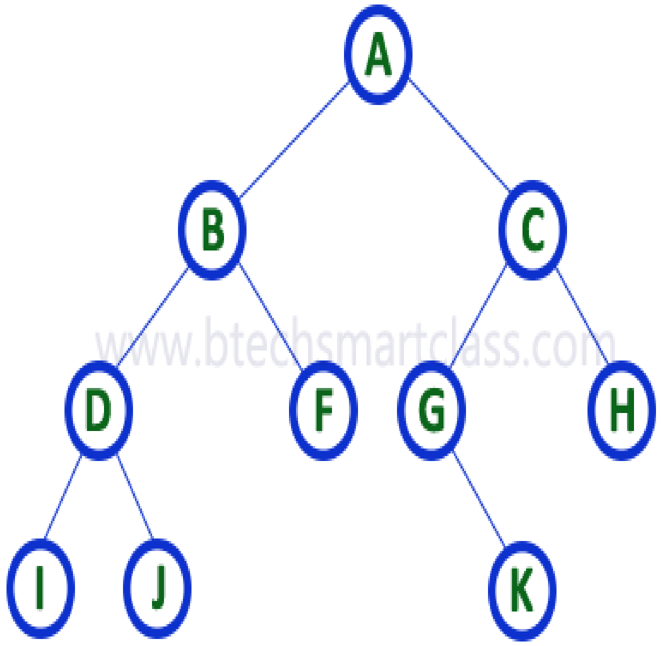
- In any tree, 'Depth of Node' is total number of Edges from root to that node.
- In any tree, 'Depth of Tree' is total number of edges from root to leaf in the longest path.

- FOREST

- It is a disjoint trees.
- In a given tree, if you remove its root node then it becomes a forest

Binary Tree

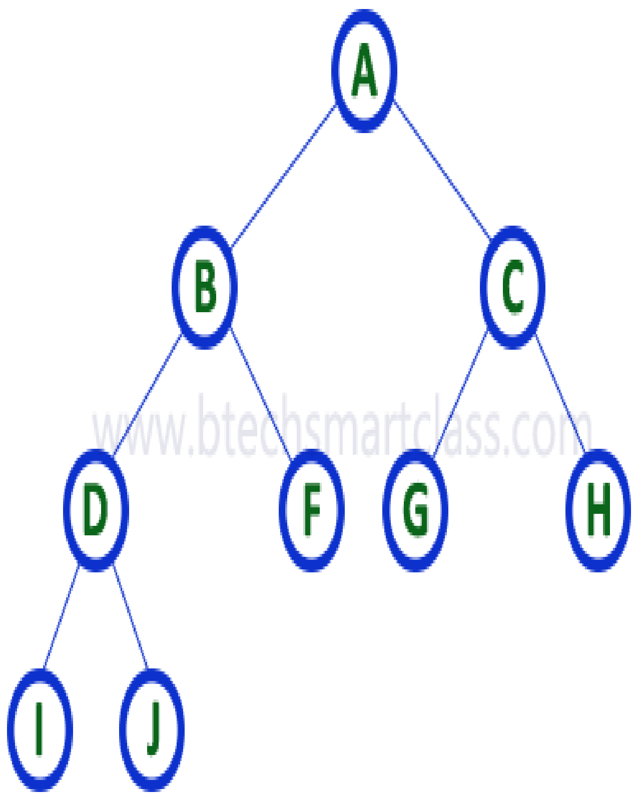
- A binary tree is a very important and most commonly used non-linear data structure.
- In a normal tree, every node can have any number of children.
- Binary tree is a special type of tree data structure in which every node can have a **maximum of 2 children**.
- That means, there may be a zero degree node or a one degree node and two degree node.
- One is known as **left child** and the other is known as **right child**.


A
B
C
D
F
G
H
I
J
K

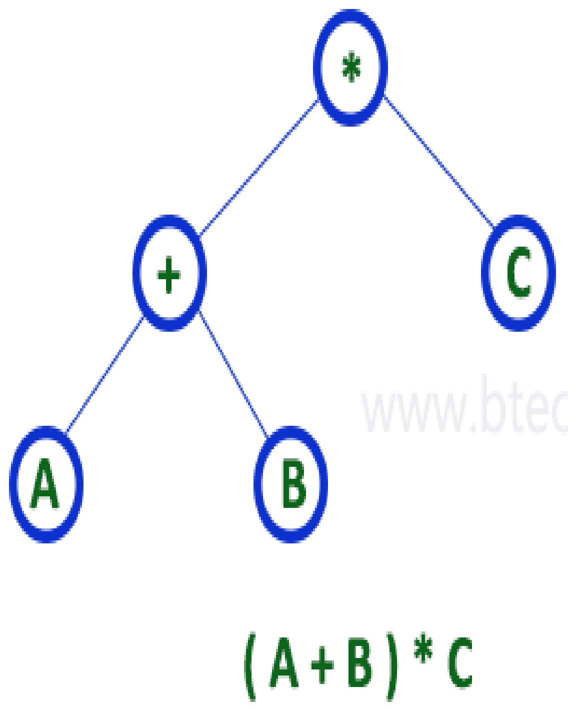
THERE ARE DIFFERENT TYPES OF BINARY TREES AND THEY ARE…

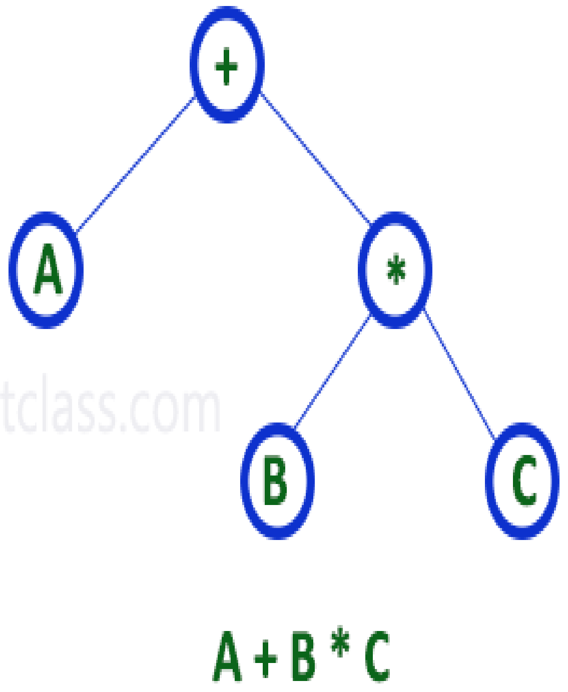
- **Strictly Binary Tree**

- In a binary tree, every node can have a maximum of two children.
- But in strictly binary tree, every node should have exactly two children or none.
- That means every internal node must have exactly two children.
- If every non-terminal node in a binary tree consist of non empty left subtree and right subtree, then such a tree is called strictly binary tree


A
B
C
D
F
G
H
I
J

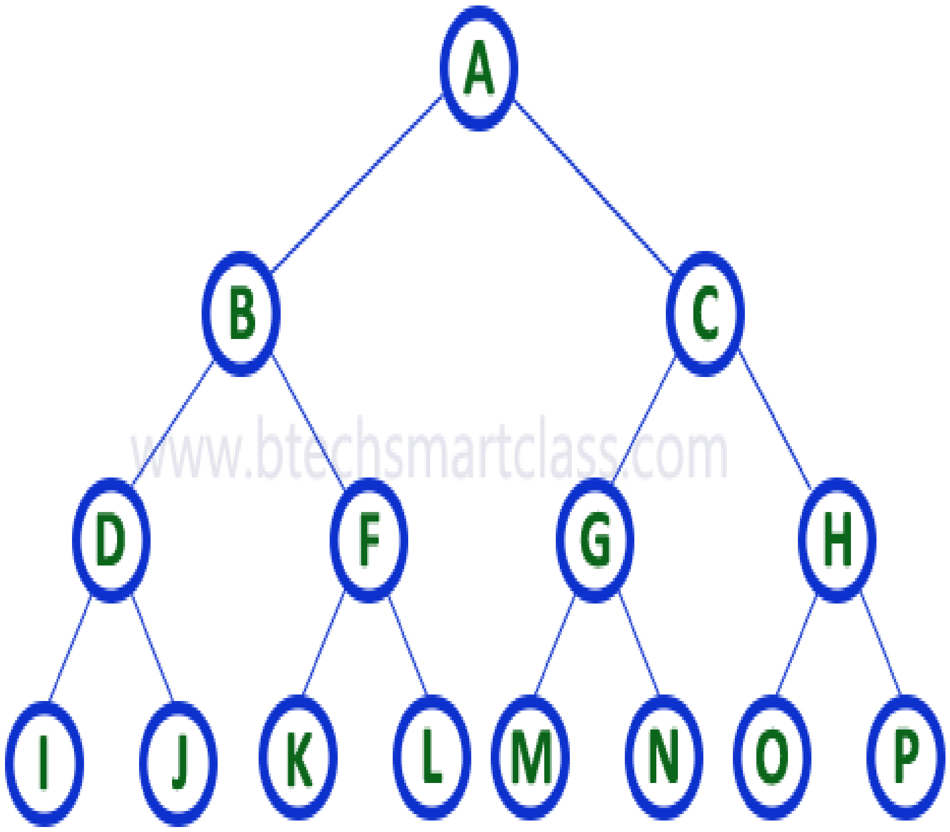
Example





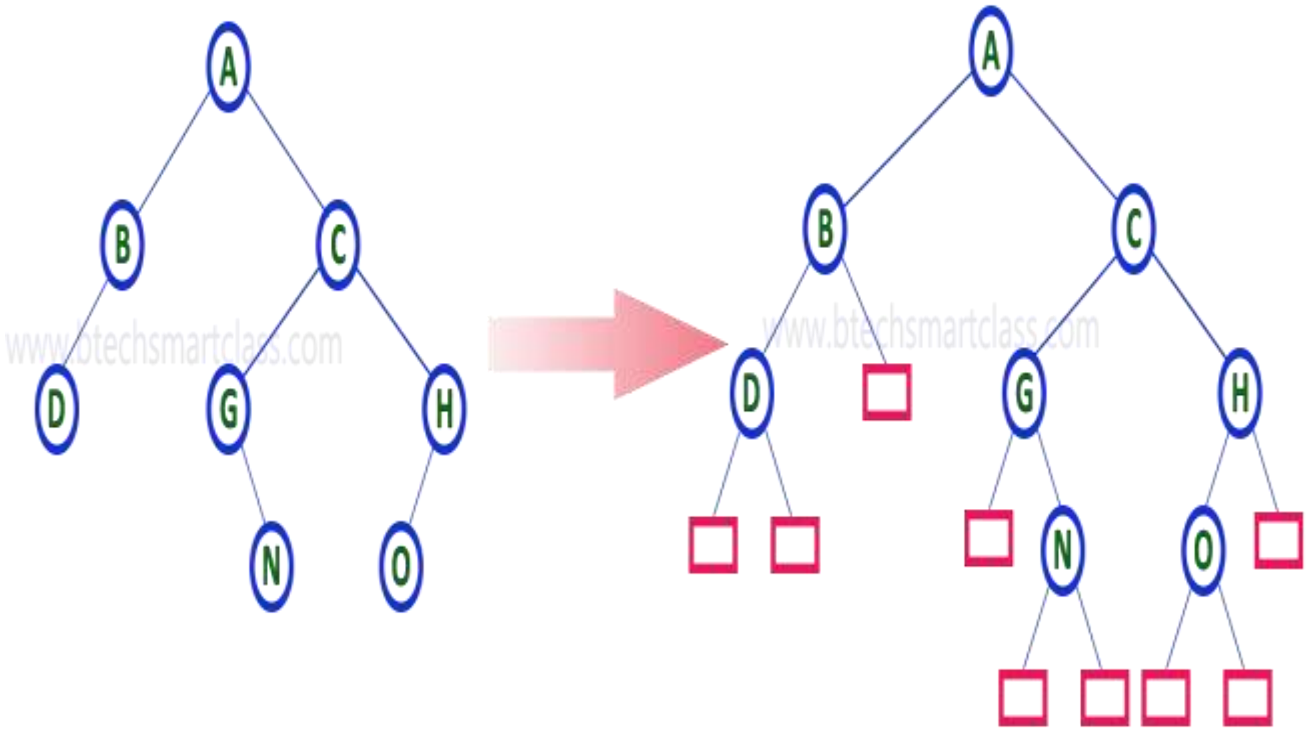
Complete Binary Tree

- In a binary tree, every node can have a maximum of two children.
- But in strictly binary tree, every node should have exactly two children or none and in complete binary tree all the nodes must have exactly two children and at every level of complete binary tree there must be 2^{level} number of nodes.
- In a complete binary tree, there is exactly one node at level 1, two node at level 1, and four nodes at level 2 and so on.
- For example at level 2 there must be $2^2 = 4$ nodes and at level 3 there must be $2^3 = 8$ nodes.


A
B
C
D
F
G
H
I
J
K
L
M
N
O
P

Extended Binary Tree

- A binary tree can be converted into Full Binary tree by adding dummy nodes to existing nodes wherever required.
- The full binary tree obtained by adding dummy nodes to a binary tree is called as Extended Binary Tree.

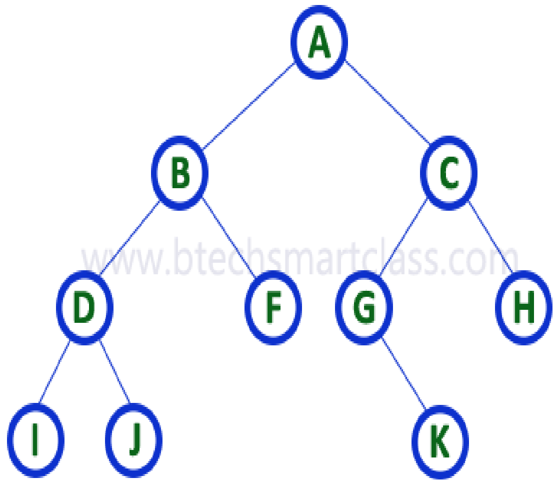

A
B
C
D
G
H
N
O
www.btechsmartclass.com
A
B
C
D
G
H
N
O
www.btechsmartclass.com

Binary Tree Representations

- A binary tree data structure is represented using two methods. Those methods are as follows...

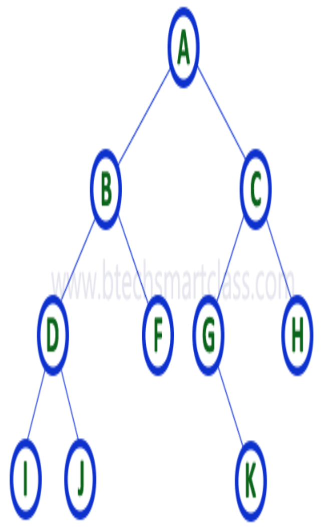
1. **Array Representation**
2. **Linked List Representation**

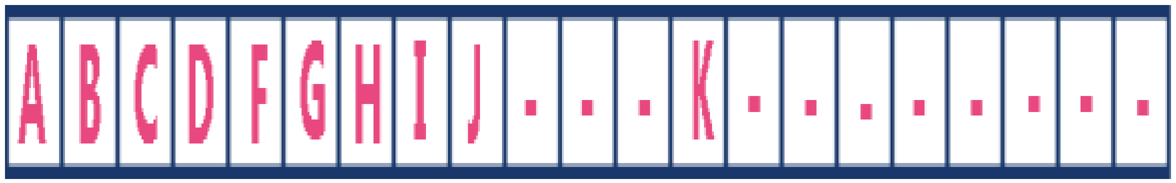
- Consider the following binary tree...



ARRAY REPRESENTATION

- In array representation of binary tree, we use a one dimensional array (1-D Array) to represent a binary tree.
- Consider the above example of binary tree and it is represented as follows…





A	B	C	D	F	G	H	I	J	.	.	.	K	.	.	.	.	.	.	.	.

- An array can be used to store the nodes of a binary tree.
- In c++, arrays start with index 0 to (MAXSIZE - 1)

How to identify the father, the left child and the right child

- It is very simple to identity the father and children of a node.
- For any node n, 0<=n<=(MAXSIZE - 1), then we have

1. Father (n) : The father of node having index n is at floor ((n-1)/2) if n is not equal to 0, Then it is the root node and has no father.
   Example: Consider a node numbered 3 (ie, D).
   The father of D is B whose index is 1.
   This is obtained from
   floor ((3-1)/2)
   =(2/2)
   =1

2 . lchild(n) : The left child of a node numbered n is at (2n+1).

For example, in the above binary tree

a) lchild(A) = lchild(0)
   = 2*0+1
   = 1
   ie, the node with index 1 is B
b) Lchild(C) = lchild(2)
   = 2*2+1
   = 5
   ie, the node with index 5 is G

3 . rchild(n) : The right child of a node numbered n is index (2n+2).

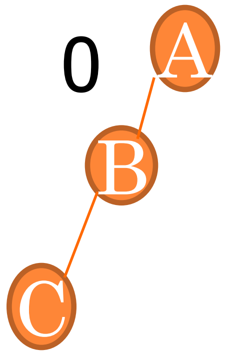
For example, in the above binary tree

a) rchild(A) = rchild(0)
   = 2*0+2
   = 2
   ie, the node with index 2 is C
b) rchild(B) = rchild(2)
   = 2*2+2
   = 6
   ie, the node with index 6 is H

3 . Siblings : If the left child at index n is given then its right sibling is at (n+1). And similarly, if the right child at index n is given then its left sibling is at (n-1).

For example : the right sibling is at (n-1) of node indexed is 4 is at index 5 in an array representation

- An array representation is more ideal for the complete binary trees.
- But , this is not suitable for other than complete binary tree as it results in unnecessary wastage of memory space.
- Consider the following In an a binary tree



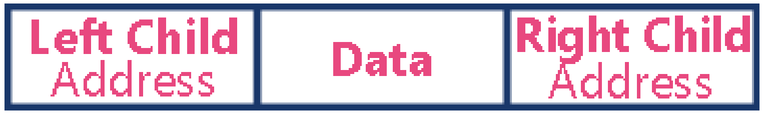
- It is a skewed binary tree
- Since only the left subtree present, this type of binary tree is called left skewed binary tree
- The array representation is

A
B
-
C
-
-

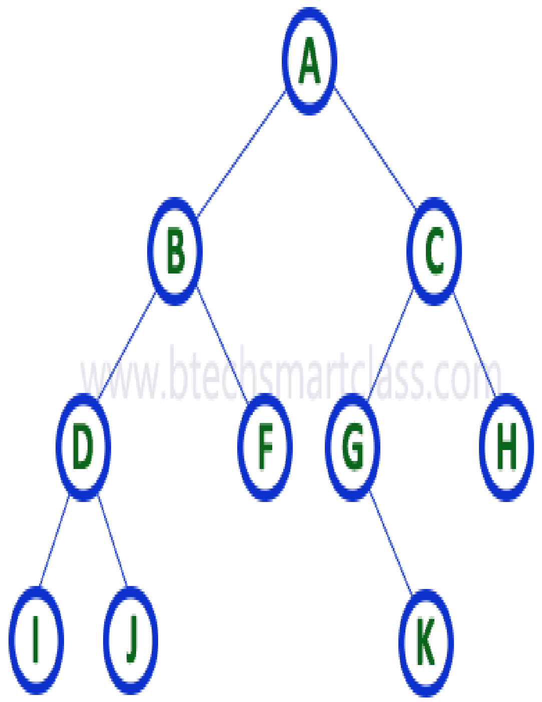
- The right child of A, is empty, and its both left child and right child are also empty whose index is 4.
- Therefore , these indexes in array binary tree are left unused. This result in wastage of memory.

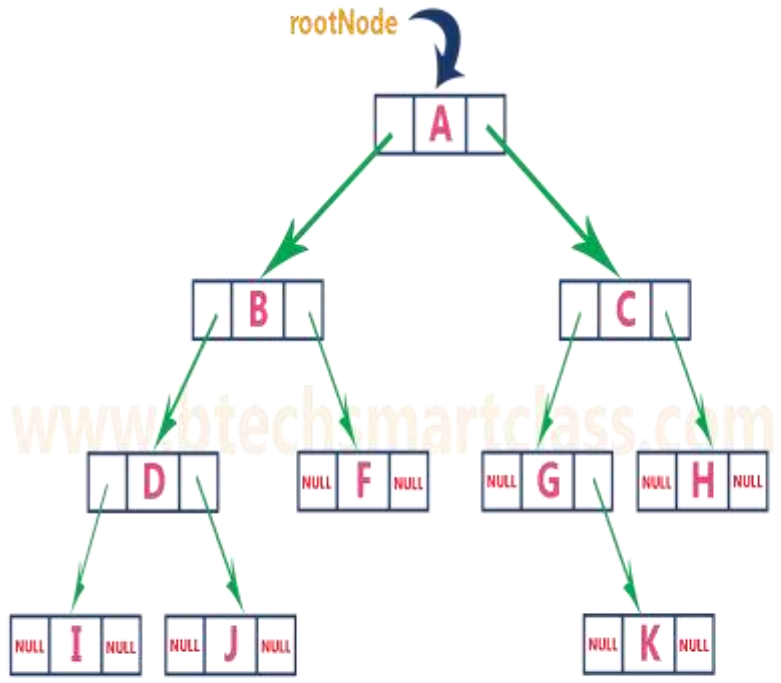
Linked List Representation

- We use double linked list to represent a binary tree.
- In a double linked list, every node consists of three fields.
- First field for storing left child address, second for storing actual data and third for storing right child address.
- In this linked list representation, a node has the following structure...



The above example of binary tree represented using Linked list representation is shown as follows...





Creation of binary tree

Create_tree(Info,Node)

where

- ✓ Info -> information for which we have to create node.
- ✓ Node -> structure type variable to point both left and right child.

Step 1. [check whether the tree is empty]

```
        if Node = NULL
        Node = create a node
        Left_Child [ Node ] = NULL
        Right_Child [ Node ] = NULL
```

```
Step 2 . [Test for the left child]
            if Info [ Node ] > = Info
            Left_Child [ Node ] = call Create_Tree
                                 ( Info,Left_Child [ Node ])
            else
             Right_Child [ Node ] = call Create_Tree
                                 ( Info,Right_Child [ Node ])
Step 3 . Return ( Node)
```

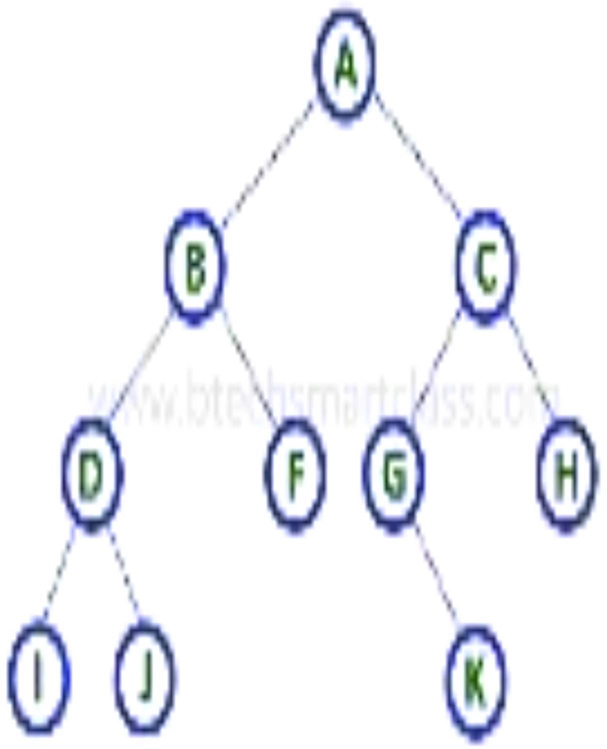
Binary Tree Traversals

- When we wanted to display a binary tree, we need to follow some order in which all the nodes of that binary tree must be displayed. In any binary tree displaying order of nodes depends on the traversal method.
- **Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal.**
- There are three types of binary tree traversals.

1. **In - Order Traversal**
2. **Pre - Order Traversal**
3. **Post - Order Traversal**

1. In - Order Traversal (leftChild - root - rightChild)

- In In-Order traversal, the root node is visited between left child and right child.
- In this traversal, the left child node is visited first, then the root node is visited and later we go for visiting right child node.
- This in-order traversal is applicable for every root node of all subtrees in the tree.
- This is performed recursively for all nodes in the tree.


A
B
C
D
F
G
H
I
J
K

- In the above example of binary tree, first we try to visit left child of root node 'A', but A's left child is a root node for left subtree.
- so we try to visit its (B's) left child 'D' and again D is a root for subtree with nodes D, I and J.
- So we try to visit its left child 'I' and it is the left most child. So first we visit **'I'** then go for its root node **'D'** and later we visit D's right child **'J'**. With this we have completed the left part of node B.
- Then visit **'B'** and next B's right child **'F'** is visited.
- With this we have completed left part of node A.

- Then visit root node **'A'**. With this we have completed left and root parts of node A.
- Then we go for right part of the node A. In right of A again there is a subtree with root C.
- So go for left child of C and again it is a subtree with root G. But G does not have left part so we visit **'G'** and then visit G's right child K. With this we have completed the left part of node C.
- Then visit root node **'C'** and next visit C's right child **'H'** which is the right most child in the tree so we stop the process.

  That means here we have visited in the order of **I - D - J - B - F - A - G - K - C - H** using In-Order Traversal.

- The inorder traversal of a non-empty binary tree

1. Traverse the left subtree in inorder(L)
2. Visit the root node (N).
3. Traverse the right subtree inorder(R)

- The algorithm in inorder traversal

```
Void inorder(struct rect *tree)
{
 if(tree!=NULL)
 {
   inorder(tree -> left);
   cout<<tree -> num);
   inorder(tree -> right);
}
}
```

2. Pre - Order Traversal (root - leftChild - rightChild)

- In Pre-Order traversal, the root node is visited before left child and right child nodes.
- In this traversal, the root node is visited first, then its left child and later its right child.
- This pre-order traversal is applicable for every root node of all subtrees in the tree.

- In the above example of binary tree, first we visit root node **'A'** then visit its left child **'B'** which is a root for D and F. So we visit B's left child **'D'** and again D is a root for I and J. So we visit D's left child **'I'** which is the left most child.
- So next we go for visiting D's right child **'J'**. With this we have completed root, left and right parts of node D and root, left parts of node B.
- Next visit B's right child **'F'**. With this we have completed root and left parts of node A.
- So we go for A's right child **'C'** which is a root node for G and H.

.

- After visiting C, we go for its left child **'G'** which is a root for node K.
- So next we visit left of G, but it does not have left child so we go for G's right child **'K'**.
- With this we have completed node C's root and left parts.
- Next visit C's right child **'H'** which is the right most child in the tree. So we stop the process.

That means here we have visited in the order of **A-B-D-I-J-F-C-G-K-H** using Pre-Order Traversal.

- The preorder traversal of a non-empty binary tree

1. Visit the root node (N).
2. Traverse the left subtree in preorder(L)
3. Traverse the right subtree preorder(R)

- The algorithm in preorder traversal

```
Void preorder(struct rect *tree)
{
 if(tree!=NULL)
 {
    cout<<tree -> num);
    preorder(tree -> left);
    preorder(tree -> right);
}
}
```

3. Post - Order Traversal (leftChild - rightChild - root)

- In Post-Order traversal, the root node is visited after left child and right child.
- In this traversal, left child node is visited first, then its right child and then its root node.
- This is recursively performed until the right most node is visited.

Here we have visited in the order of **I - J - D - F - B - K - G - H - C - A** using Post-Order Traversal.

3. Post - Order Traversal (leftChild - rightChild - root)

- In Post-Order traversal, the root node is visited after left child and right child.
- In this traversal, left child node is visited first, then its right child and then its root node.
- This is recursively performed until the right most node is visited.

Here we have visited in the order of **I - J - D - F - B - K - G - H - C - A** using Post-Order Traversal.

- The postorder traversal of a non-empty binary tree

1. Traverse the left subtree in postorder(L)
2. Traverse the right subtree postorder(R)
3. Visit the root node (N).

- The algorithm in postorder traversal

```
Void postorder(struct rect *tree)
{
 if(tree!=NULL)
 {
     postorder(tree -> left);
     postorder(tree -> right);
     cout<<tree -> num);

}
}
```

Technique for conversion of an expression into binary tree

- Divide and conquer technique is used to convert an expression into binary tree
- Example : A + (B + C * D + E) + F / G

1. Note the order of precedence. All expressions in parenthesis are to be evaluated first.
2. Exponention will come next.
3. Division and multiplication will be the next order of precedence.
4. Subtraction and addition will be the last to be processed.