MODULE 4

TREES & GRAPHS

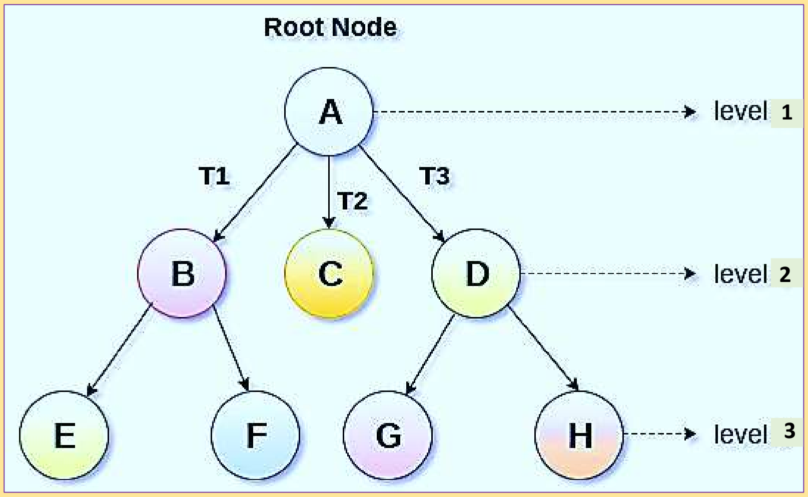
Prepared By Mr. EBIN PM, AP, IESCE 1

TREE

- A tree is a nonlinear data structure. Data elements are related in a hierarchal manner (parent-child relationship)
- The first node of the tree is called Root node.
- Final nodes are called Leaf nodes.
- Leaf nodes are also called terminal nodes.
- A child has a single parent but a parent has more than one children. So we can say that , here, a one to many relationship is exist.
- In a general tree, A node can have any number of children nodes but it can have only a single parent.

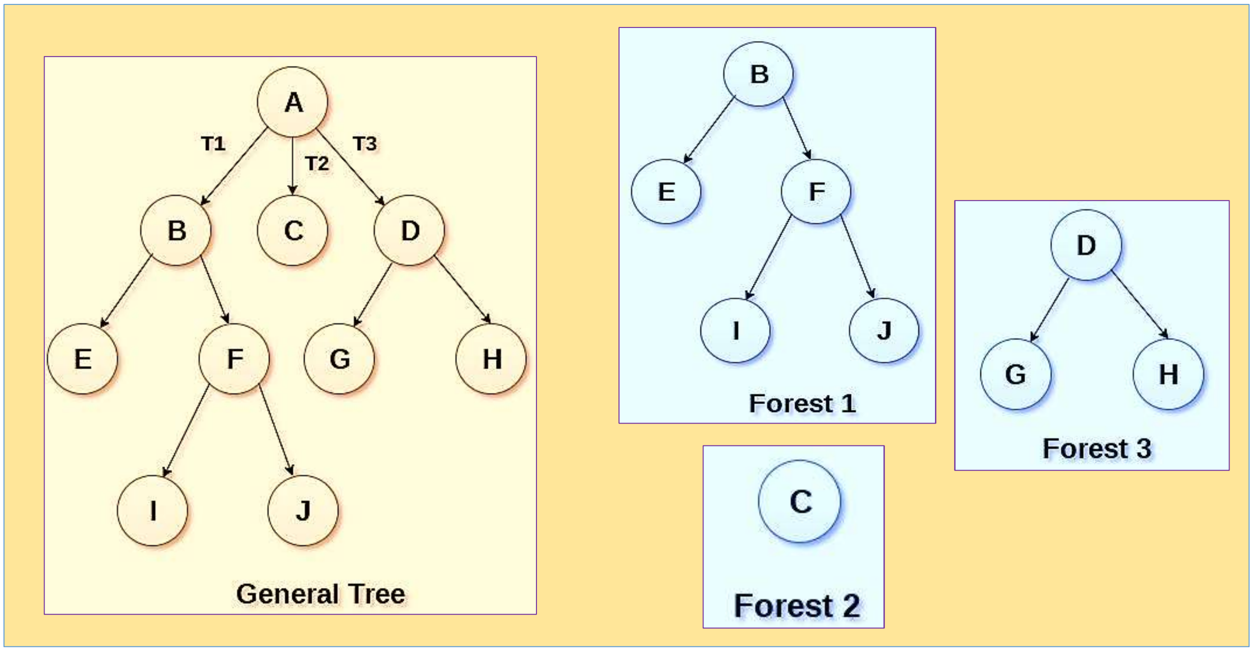
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 2

- The following image shows a tree, where the node A is the root node of the tree while the other nodes can be seen as the children of A.



❖BASIC TERMINOLOGY

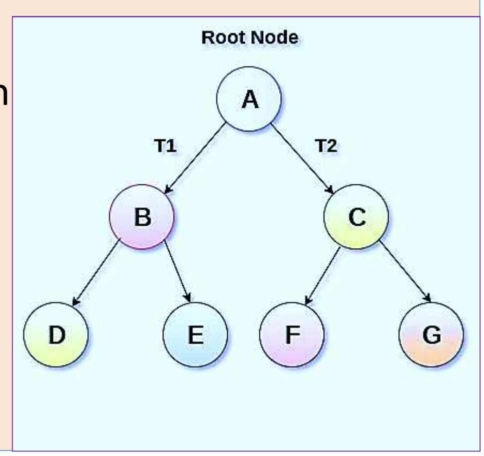
- **Root node** - topmost node in the tree hierarchy
- **Sub tree** - If the root node is not null, the tree T1, T2 and T3 is called sub-trees of the root node.
- **Degree** – The number of sub trees of a node. Eg: The degree of A is 3
- **Siblings** - Children of the same parent. Eg: G, H are the siblings of D
- **Level** – If a node is at level l, then its children are at level l+1
- **Depth** – The height or depth of a tree is defined to be the maximum level of any node in the tree
- **Forest** – If we remove the root of a tree , we get a forest. Eg: If we remove A, we get a forest with 3 trees.



BINARY TREE

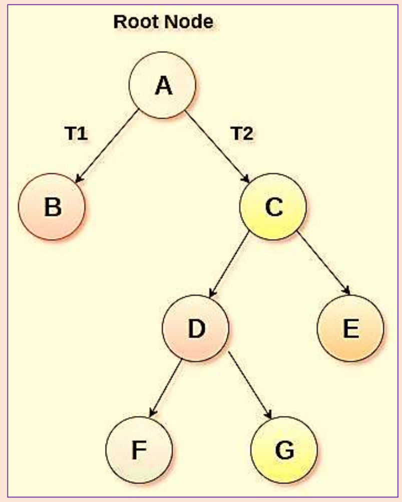
- A binary tree is either empty or consist of a root and 2 disjoint binary trees called the Left Sub tree and Right Sub tree .

- Each node can have at most two children
- B is the left child of A
- C is the right child of A



❖TYPES OF BINARY TREE

➢**Strictly Binary Tree –** All nodes must have a left child and a right child except leaf node



- Here B, F , G & E are leaf nodes

➢**Skewed Binary Tree -** It is a special kind of binary tree. Two type skewed binary trees are Left skewed and Right skewed

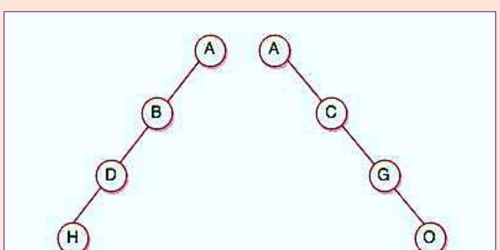


Fig. Left Skewed Binary Tree
Fig. Right Skewed Binary Tree

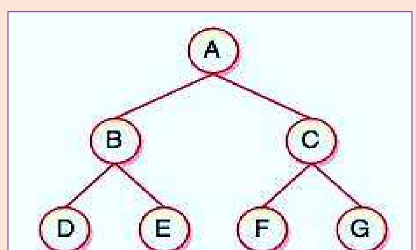


Fig. Full Binary Tree

- **Full Binary Tree -** If each node of binary tree has either two children or no child at all, is said to be a Full Binary Tree. Full binary tree is also called as Strictly Binary Tree.

➢Complete Binary Tree

- If all levels of tree are completely filled except the last level and the last level has all keys as left as possible, is said to be a Complete Binary Tree.
- Complete binary tree is also called as Perfect Binary Tree.
- In a complete binary tree, every internal node has exactly two children and all leaf nodes are at same level.
- For example, at Level 2, there must be 22 = 4 nodes and at Level 3 there must be 23 = 8 nodes.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 9

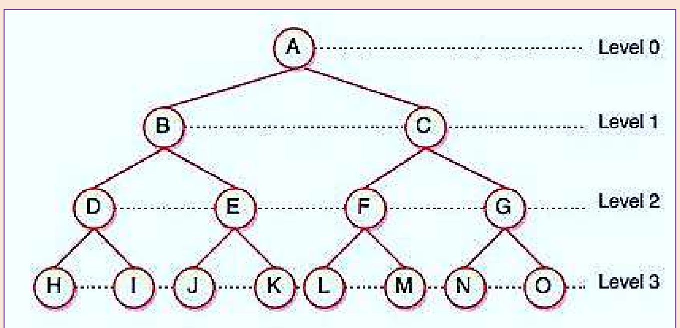


Fig. Complete Binary Tree

- **The maximum number of nodes on level "i" of a binary tree is 2^{i-1}**
- **The maximum number of nodes in a binary tree of depth k is $2^{k}-1$**

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 10

TREE REPRESENTATION

Tree can be represented using two ways

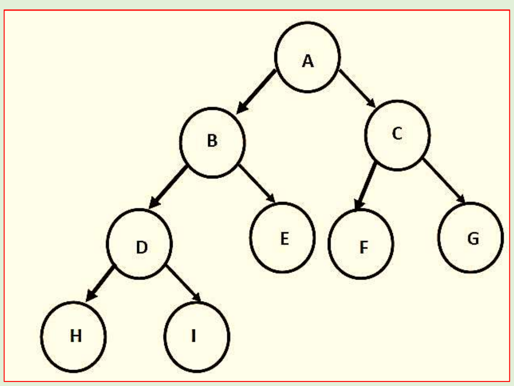
1. Using Array
2. Using Linked List

If a complete binary tree with n node, then depth = **$\lfloor \log_2 n \rfloor + 1$** , in an array representation,

- Parent (i) is at $\lfloor i/2 \rfloor$
- Lchild (i) is at 2i
- Rchild (i) is at 2i+1

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 11

Eg:1



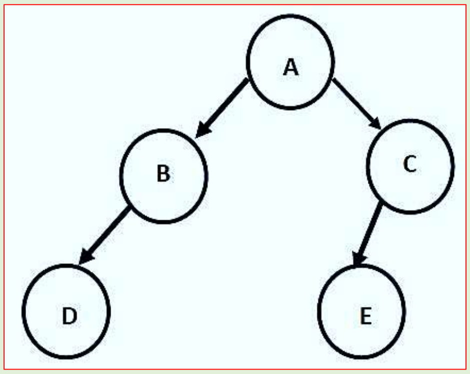
Size of array = 2^d = 2^4 = 16

In array representation, we do not use location “0”.

i →	0	1	2	3	4	5	6	7	8	9	10	
	-	A	B	C	D	E	F	G	H	I		......

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 12

Eg:2



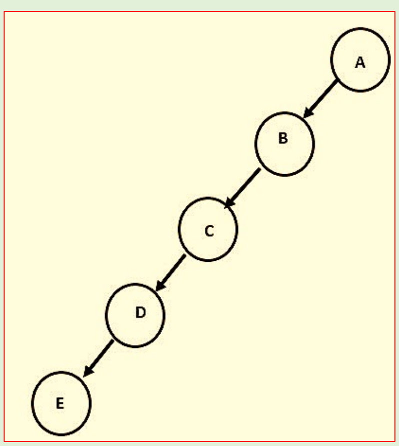
Size of array = 2^d = 2^3 = 8

In array representation, we do not use location “0”.

i →	0	1	2	3	4	5	6	7	8
	-	A	B	C	D		E		

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 13

Consider the following tree



i →	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
	-	A	B		C				D							E		

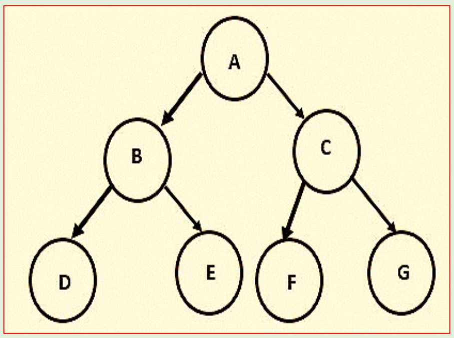
Skewed representation of a tree can be implemented in an array. But this representation is inefficient. So we use linked list representation of the tree.

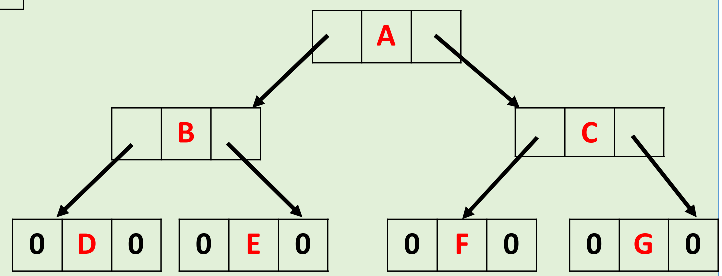
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 14

In linked list representation, a node is represented by

L child	Data	R child

Eg:





BINARY TREE TRAVERSALS

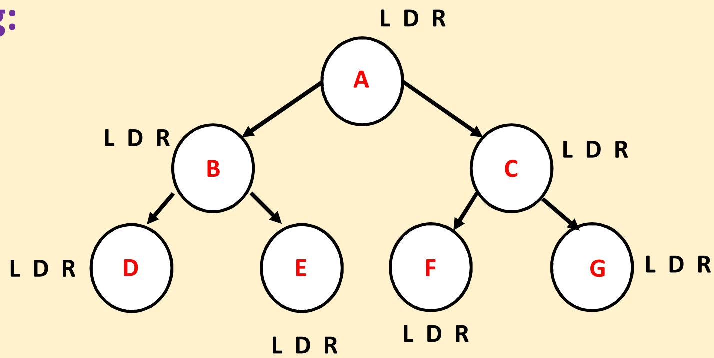
➢ Traversing means, visiting the node at least one time. Three type of traversals are:

- **In order (L D R)**
- **Pre order (D L R)**
- **Post order (L R D)**

❖In order Traversal

- It follows "**LDR**" form. (Left – Data – Right)

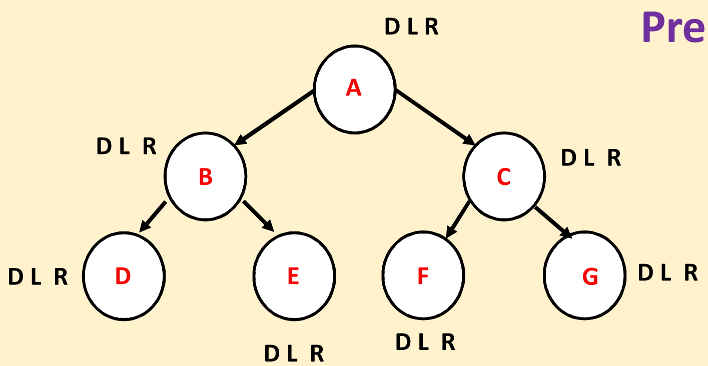
Eg:



D B E A F C G

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 17

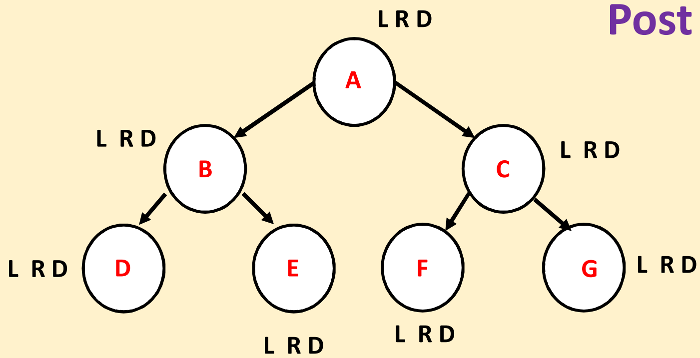
Preorder Traversal (DLR)



A B D E C F G

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 18

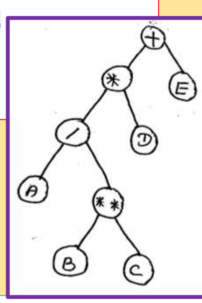
Post order Traversal (LRD)

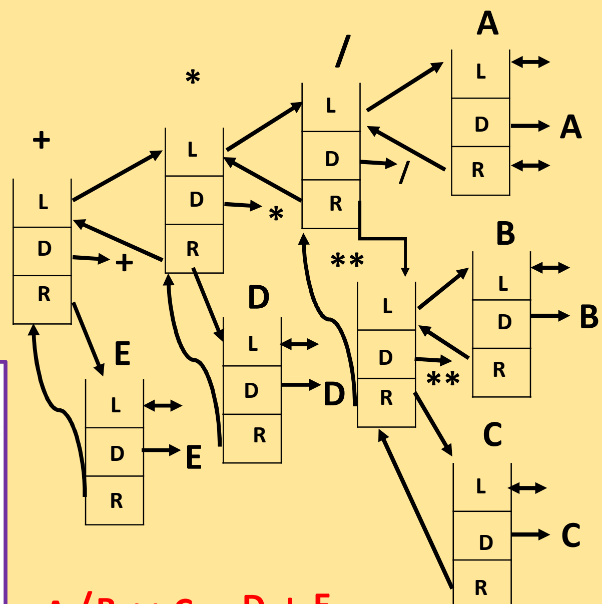


D E B F G C A

❖In order using Recursion

Algorithm inorder (current node)
{
// current node is a pointer to node in binary tree
if (current node ≠ nil)
{
inorder (current node . lchild);
write (current node . data);
inorder (current node . rchild);
}
}

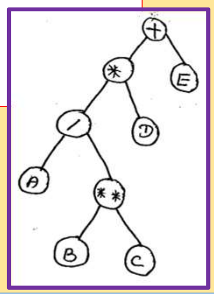


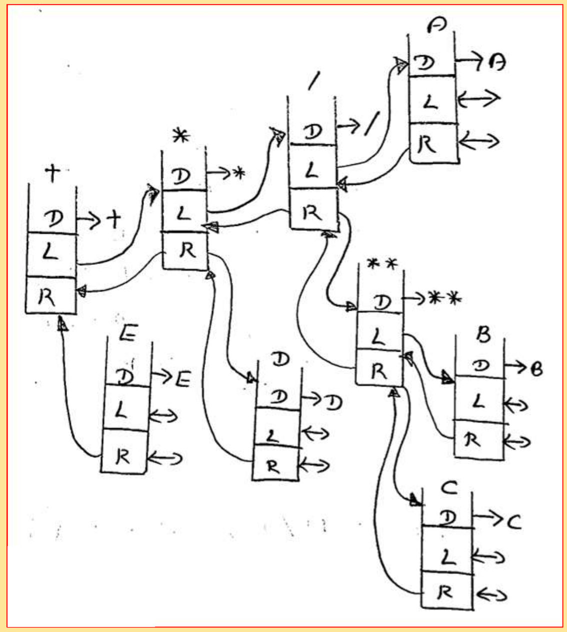


A / B ** C * D + E

❖Preorder using Recursion

```
Algorithm preorder (current node)
{
// current node is a pointer to node in binary tree
  if (current node ≠ nil)
   {
      write (current node . data);
      preorder (current node . lchild);
      preorder (current node . rchild);
   }
}
```





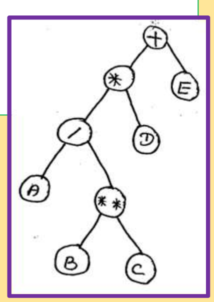
Ans:

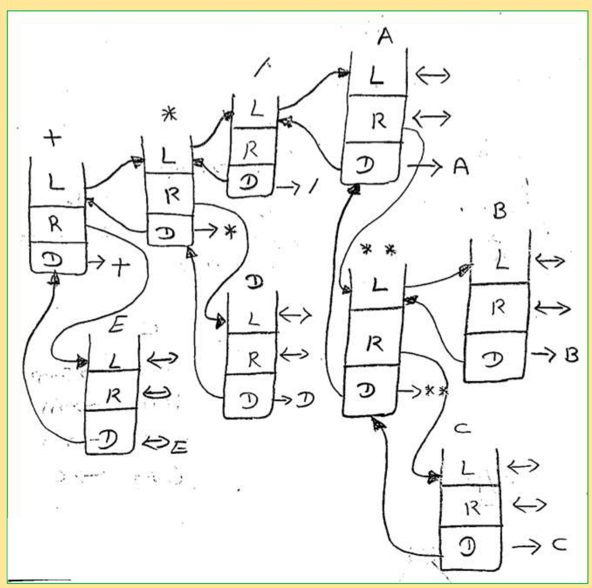
+ * / A ** B C D E

Prepared By Mr.EBIN PM, AP, IESCE   EDULINE   21

❖Postorder using Recursion

```
Algorithm postorder (current node)
{
// current node is a pointer to node in binary tree
  if (current node ≠ nil)
   {
      postorder (current node . lchild);
      postorder (current node . rchild);
      write (current node . data);
   }
}
```





Ans:

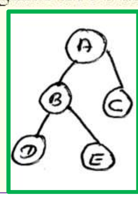
A B C ** / D * E +

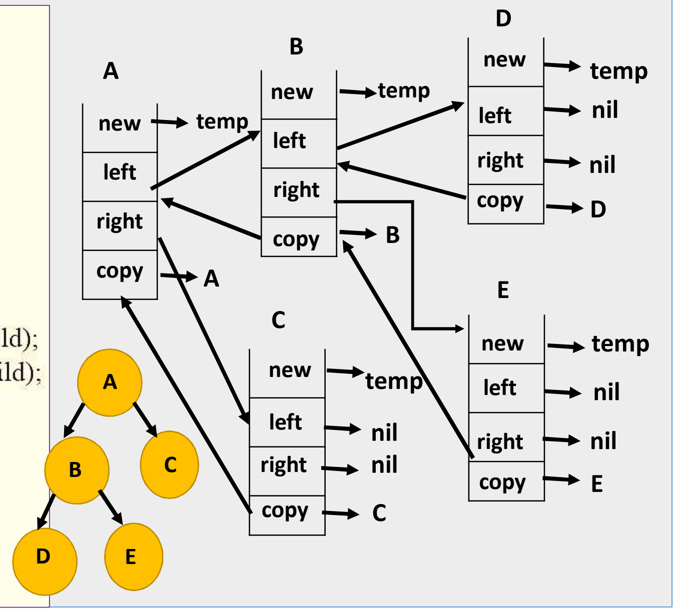
Prepared By Mr.EBIN PM, AP, IESCE   EDULINE   22

❖ALGORITHM TO CREATE A COPY OF A BINARY TREE

```
Algorithm copy (original tree)
{
// for a binary tree "original tree" , copy
Returns a pointer to an exact copy of the
Original tree

  if (original tree ≠ nil)
    {
      new (temp tree);
      temp tree .lchild = copy(original tree . lchild);
      temp tree .rchild = copy(original tree . rchild);
      temp tree . data = original tree . data;
      copy = temp tree;
    }
  else
     copy = nil;
}
```

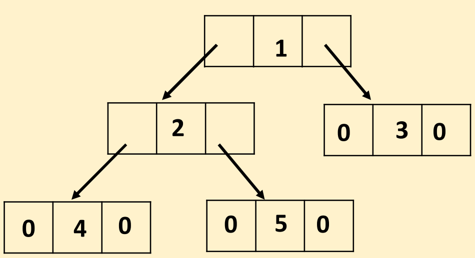




THREADED BINARY TREE

- In normal case, binary tree is represented using linked list. In that case, if 'n' nodes present, then total number of links is 2×n. Here n+1 link is a null link.

For example,



Number of node 'n' = 5

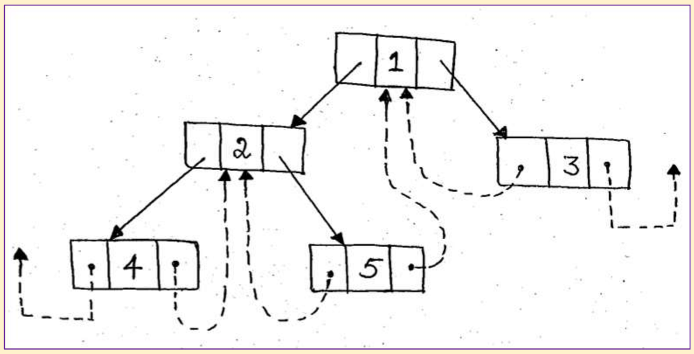
Total no.of links = 2×n = 2×5=10

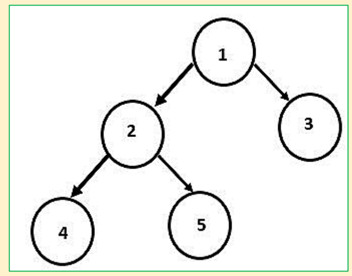
Null link = n+1 = 5+1 = 6

This is a disadvantage. So for avoiding this we use threaded binary trees

The idea is to replace the null link by pointer called thread, to other nodes in the tree. Here,

If the Lchild of a node is empty, then we pointed this on an inorder predecessor. If Rchild is empty, then we pointed this on an inorder successor.





Inorder = 4 2 5 1 3

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 25

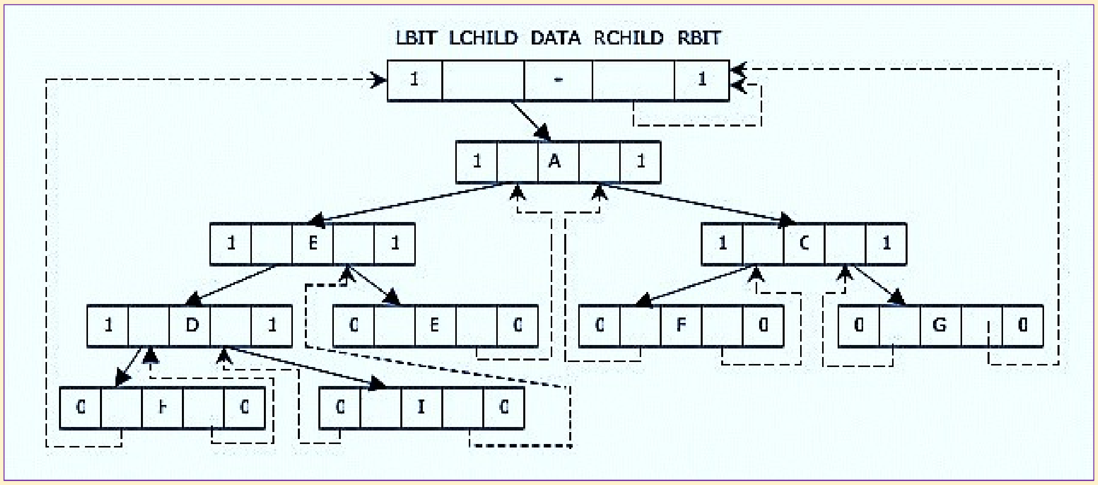
- The disadvantage of threaded binary tree is the dangling pointer.
- The left pointer of leftmost child is a dangling pointer and the right pointer of rightmost child is a dangling pointer.
- In memory representation we must distinguish the normal pointer and thread pointer. So we create a new node structure.

L thread	**L child**	**Data**	**R child**	**R thread**

- Lthread and Rthread are Boolean variables. It can receive either True or False value
- If Lthread = T, it indicate that the Lchild is a thread pointer
- If Lthread = F, it indicate that Lchild pointer is a normal pointer. That is it points to the left child.

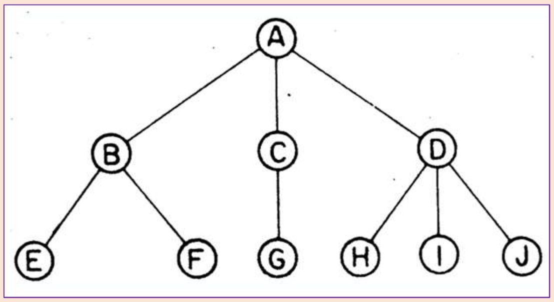
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 26

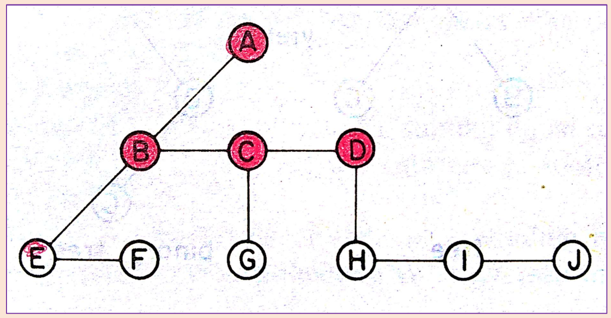
- For avoiding the dangling reference pointer, we include a head node to the tree.



BINARY TREE REPRESENTATION OF TREES

- Every tree can be represented as a binary tree.
- To obtain a binary tree representation, we need a relationship between the nodes that can be characterized by at most two quantities.
- One such relationship is **Left most child – Next – Right sibling**
- Every node has at most one left most child and at most one next right sibling.
- In the following figure, the left most child of B is E and the next right sibling of B is C. The node structure is given below

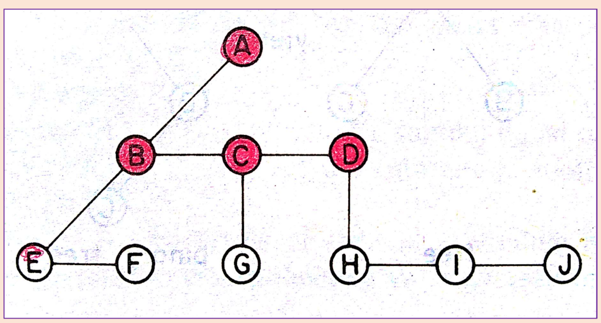




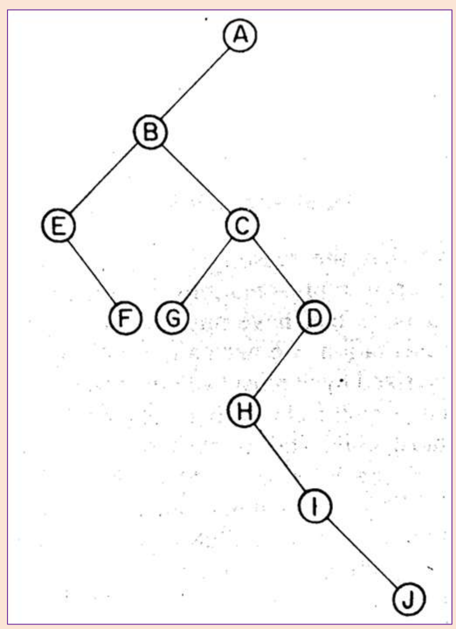
Node Structure

DATA	
CHILD	SIBLING

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 29



We tilt the figure roughly 45 degree clockwise and this is the binary tree representation.



Prepared By Mr.EBIN PM, AP, IESCE EDULINE 30

CREATION OF A BINARY TREE

Preorder = ABCDEFGHI

Inorder = BCAEDGHFI

In preorder , the first one is the root node.ie, A.

Scanning is done from left to right, so we complete the left side first.

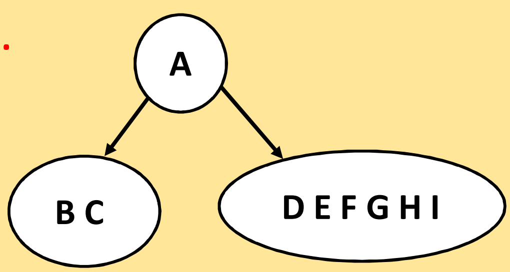
Scan →

Preorder : A B C D E F G H I

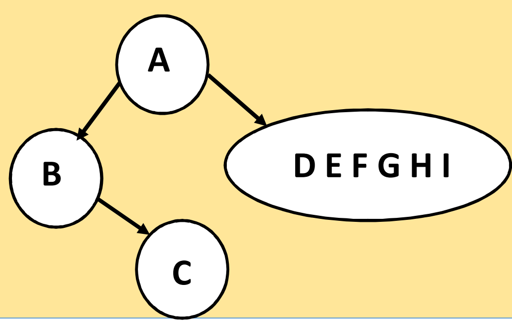
Inorder : B C A E D G H F I

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 31

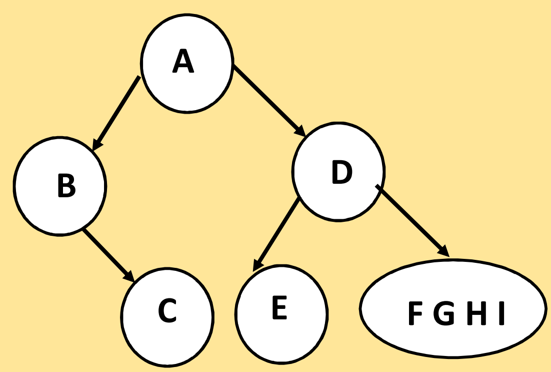
1.



2.



3.

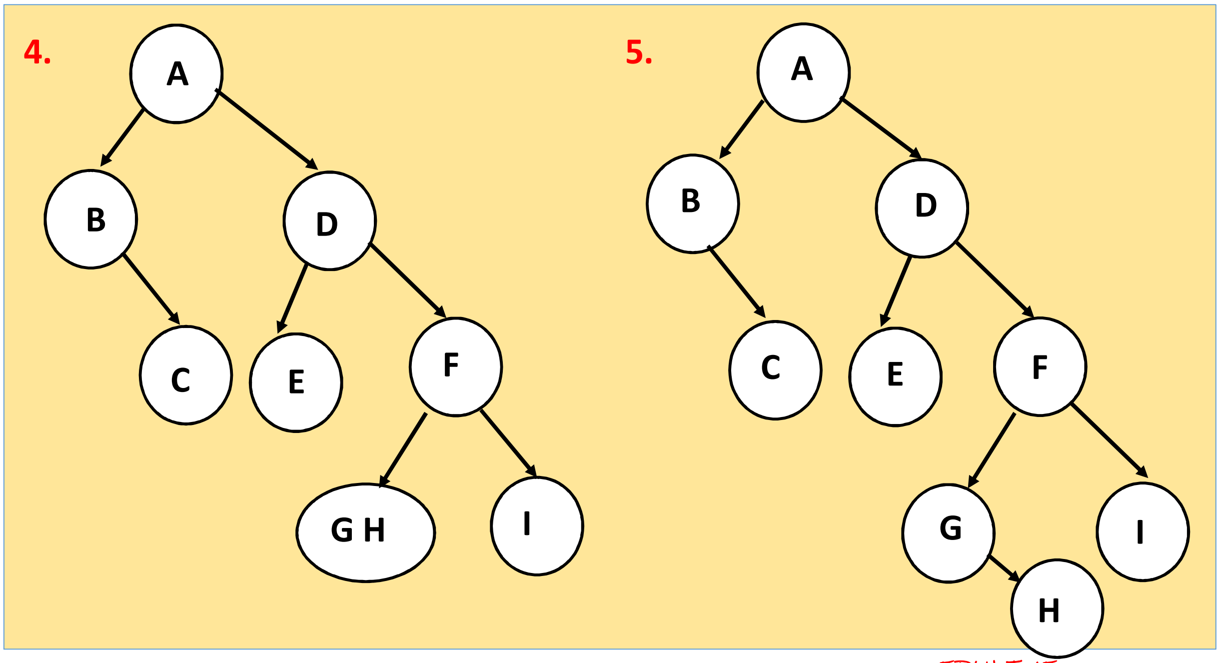


Scan →

Preorder : A B C D E F G H I

Inorder : B C A E D G H F I

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 32



Prepared By Mr.EBIN PM, AP, IESCE EDULINE 33

CREATION OF A BINARY TREE

Post order = A B C */ D – E F G + * +

Inorder = A / B * C – D + E * F + G

In post order , the last one is the root node.ie, +

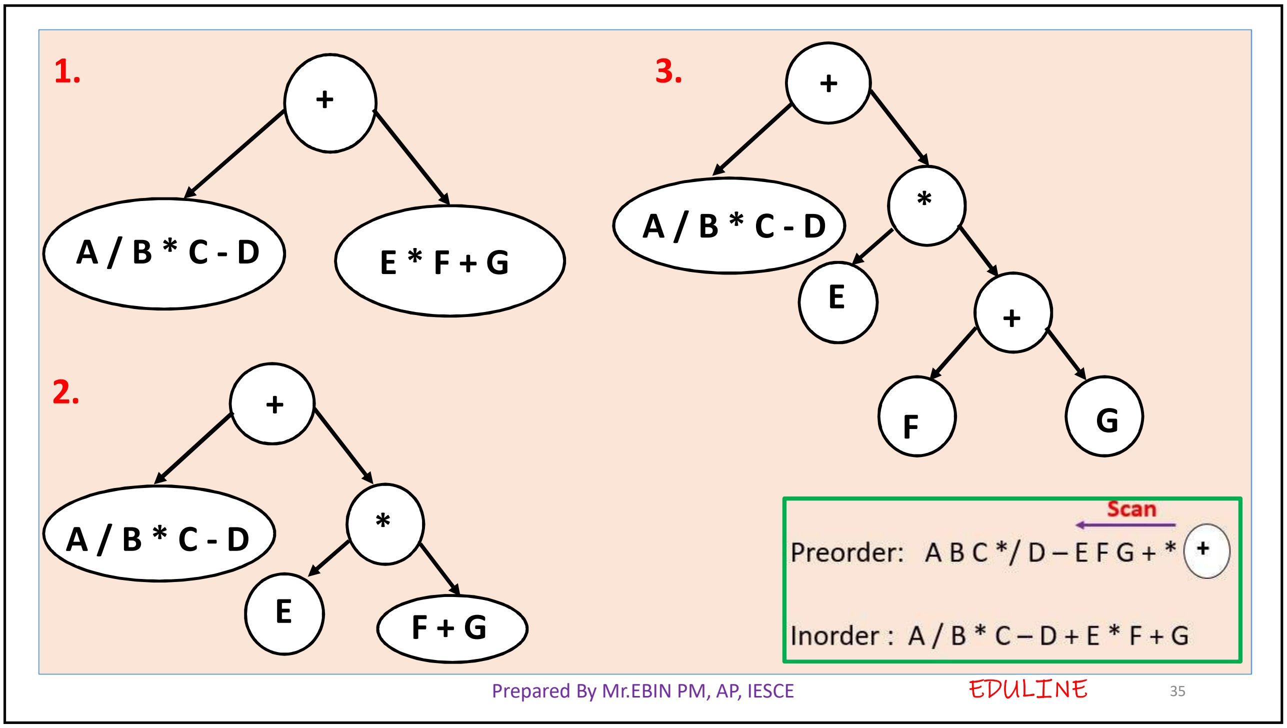
Scanning is done from right to left, so we complete the right side first.

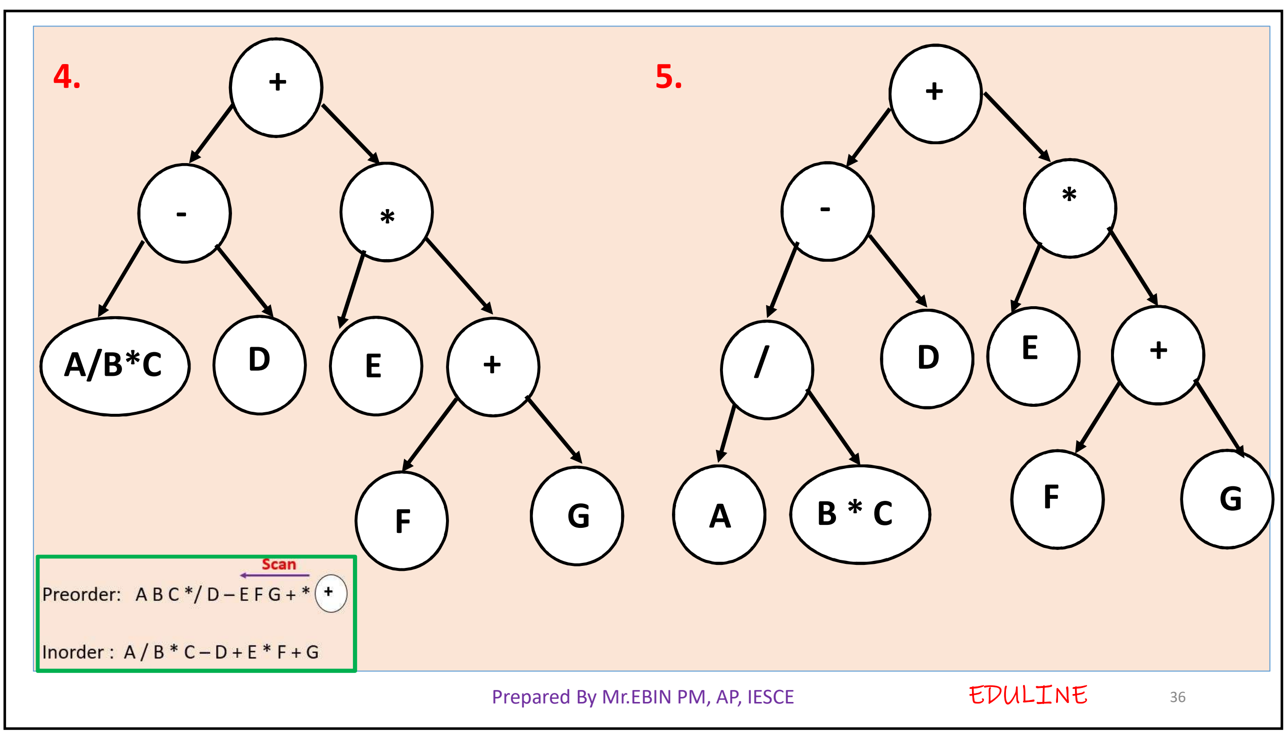
Scan

Preorder: A B C */ D – E F G + * +

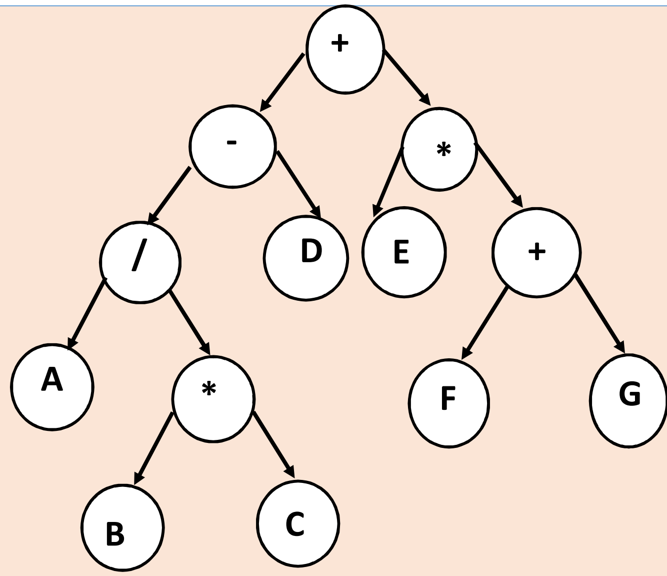
Inorder : A / B * C – D + E * F + G

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 34


1.
+
A / B * C - D
E * F + G
2.
+
A / B * C - D
*
E
F + G
3.
+
A / B * C - D
*
E
+
F
G
Scan
Preorder: A B C */ D – E F G + * +
Inorder : A / B * C – D + E * F + G
Prepared By Mr.EBIN PM, AP, IESCE
EDULINE
35


4.
+
-
*
A/B*C
D
E
+
F
G
Scan
Preorder: A B C */ D – E F G + * +
Inorder : A / B * C – D + E * F + G
5.
+
-
*
/
D
E
+
A
B * C
F
G
Prepared By Mr.EBIN PM, AP, IESCE
EDULINE
36

6.



BINARY SEARCH TREE (BST)

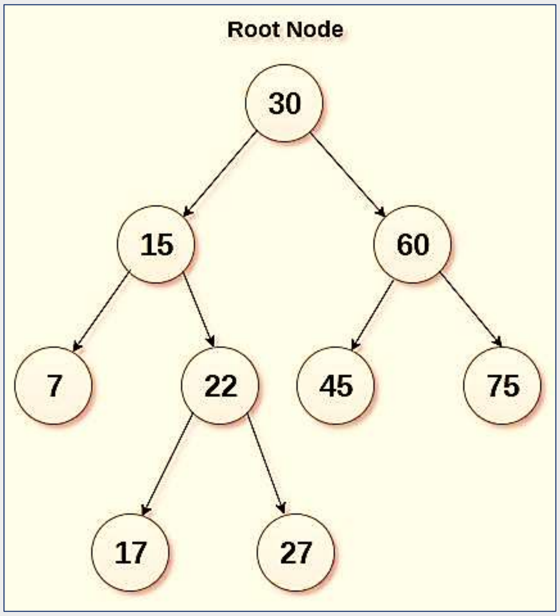
➢A Binary Search Tree is a binary tree ; either it is empty or each node in the tree contains an identifier and

a) All identifiers in the left sub-tree of T are less than the identifier in the root node T.

b) All identifiers in the right sub-tree of T are greater then the identifier in the root node T

c) The left and right sub-tree of T are also binary search trees.

Eg:

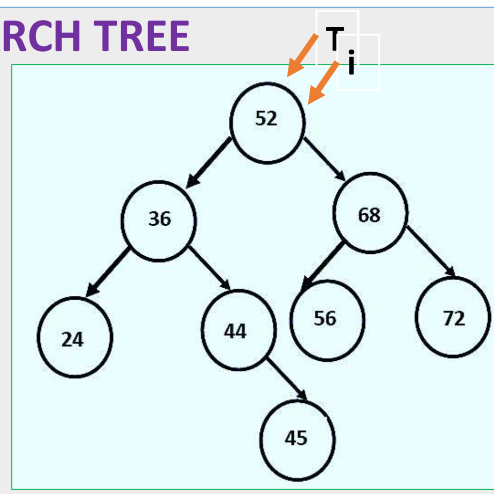


(Fig: Binary Search Tree)

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 39

❖SEARCHING IN BINARY SEARCH TREE

```
Algorithm search (T, x, i)
{
   i = T;
   done = false;
  while (i ≠ 0 and done = false) do
   {
     if (x < IDENT(i)) then
        i=Lchild(i);
     elseif (x > IDENT(i)) then
        i=Rchild(i);
     else
        done=true;
   }
 }
```

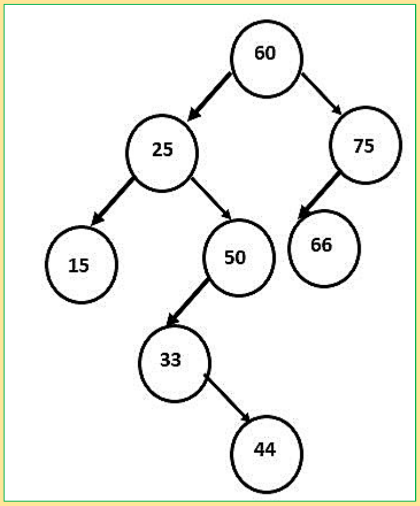


Searching element x = 44

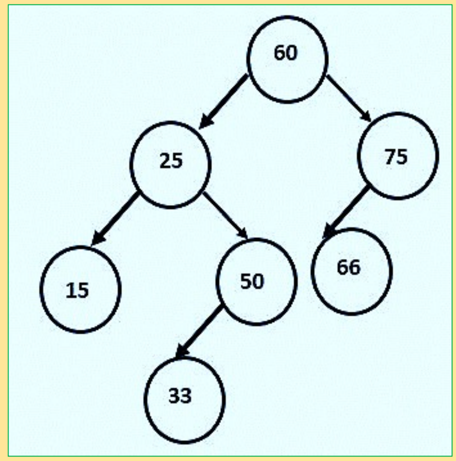
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 40

DELETION FROM A BINARY SEARCH TREE

Original Tree

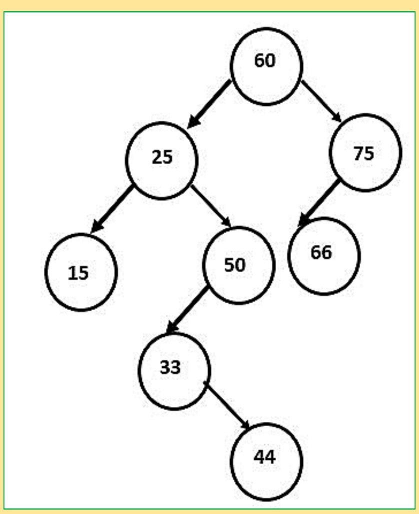


When we delete 44 (No children)

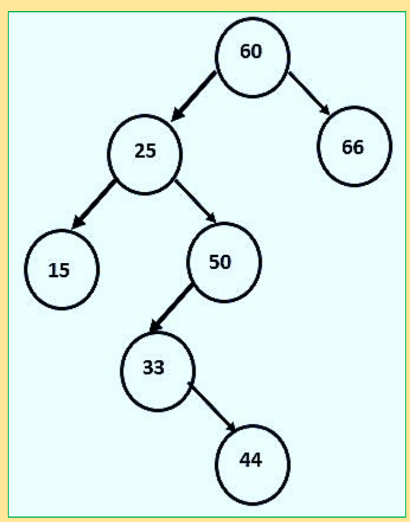


Prepared By Mr.EBIN PM, AP, IESCE EDULINE 41

Original Tree

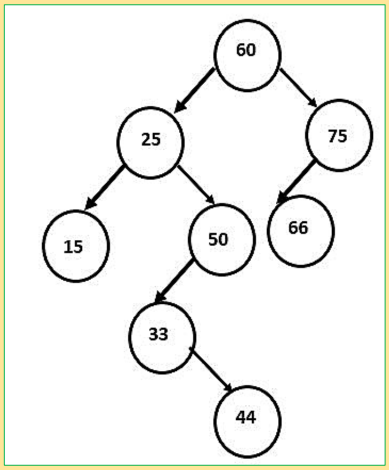


When we delete 75 (one child)

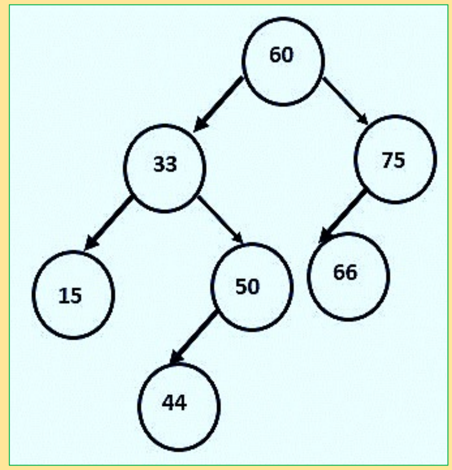


Prepared By Mr.EBIN PM, AP, IESCE EDULINE 42

Original Tree



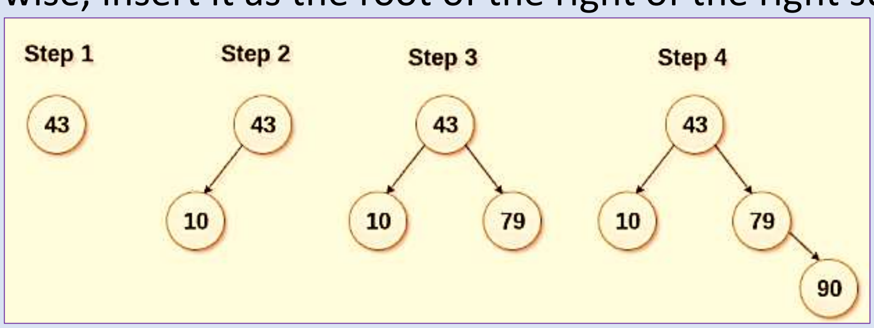
When we delete 25 (one children)

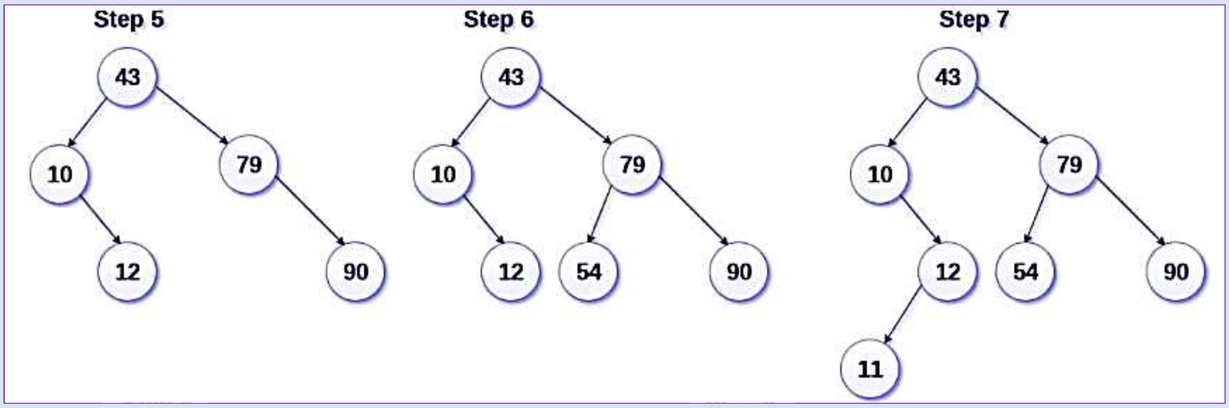


- When we delete a node which has two children, we replace it with its own inorder successor. 33 is the inorder successor of node 25

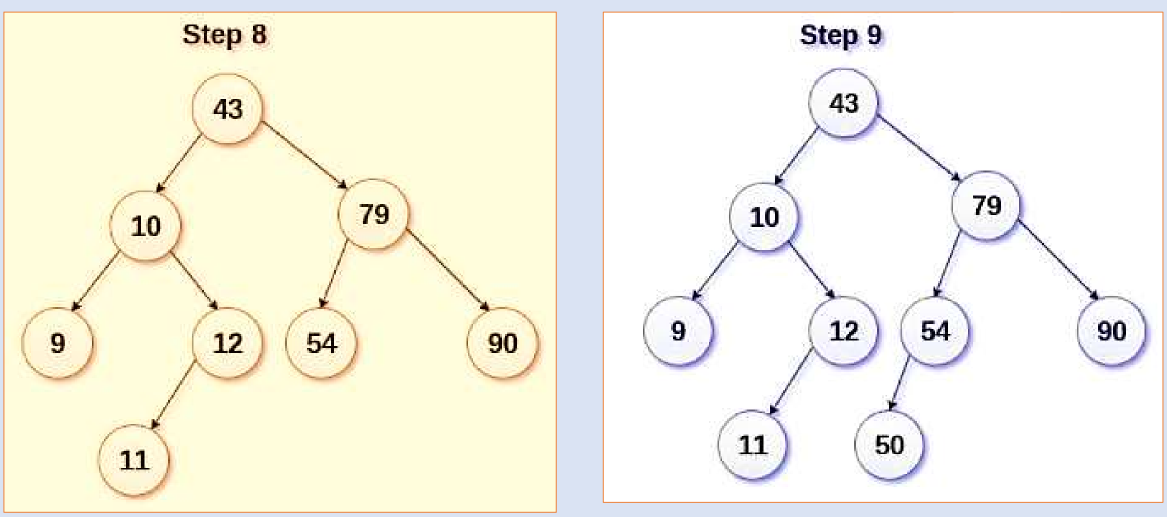
❖**Create the binary search tree using the following data elements** **43, 10, 79, 90, 12, 54, 11, 9, 50**

1. Insert 43 into the tree as the root of the tree.
2. Read the next element, if it is lesser than the root node element, insert it as the root of the left sub-tree.
3. Otherwise, insert it as the root of the right of the right sub-tree.





Prepared By Mr.EBIN PM, AP, IESCE EDULINE 45

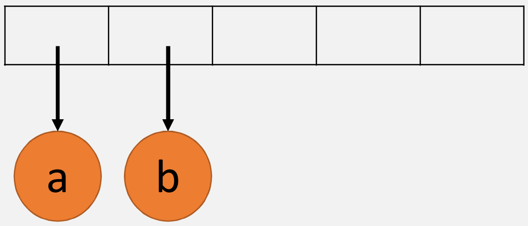


Prepared By Mr.EBIN PM, AP, IESCE EDULINE 46

CONSTRUCTING AN EXPRESSION TREE

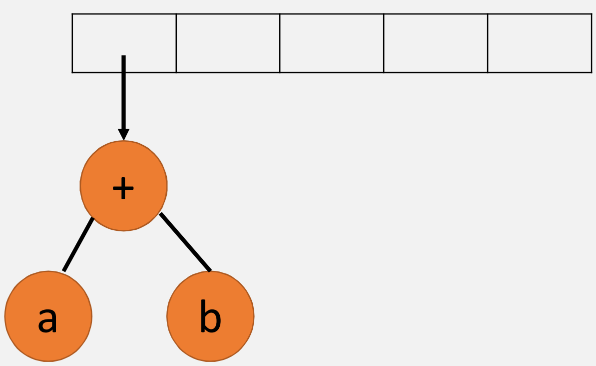
Eg: **a b + c d e + * *** (postfix expression)

- The first two symbols are operands and we create nodes and push pointers to the stack.



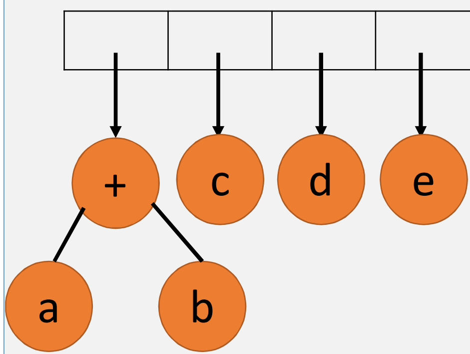
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 47

- Next "+" is read. So 2 pointers from stack are popped and new tree is formed and pointer to it is pushed on to the stack

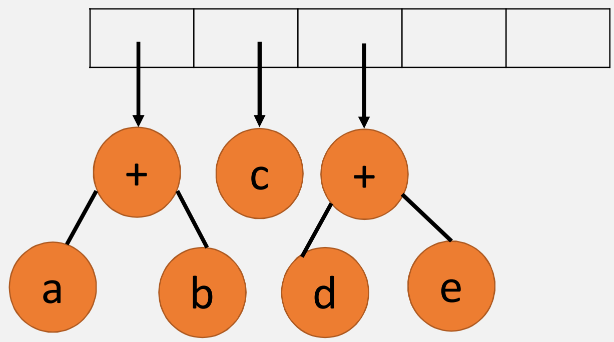


- Next "c" , "d" and "e" are read . Node is created and its pointers push into the stack.

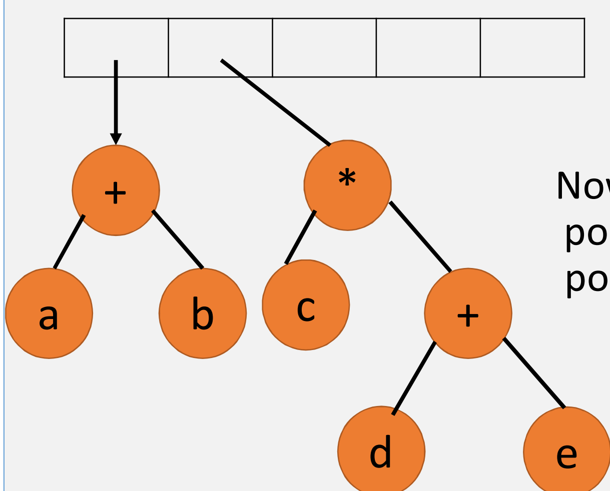
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 48



Now “+” is read and 2 operands are popped from stack

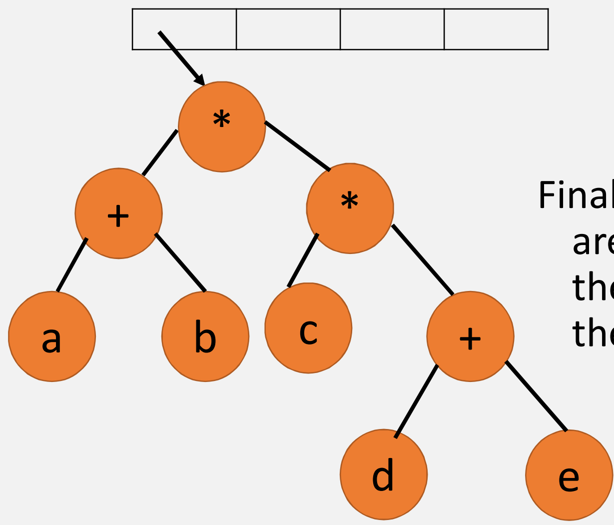


Prepared By Mr.EBIN PM, AP, IESCE EDULINE 49



Now “*” is read and 2 operands are popped from stack and another pointer is pushed on to the stack

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 50



Finally, * is read , so the two pointers are popped and perform operation, then the new pointer is added to the stack

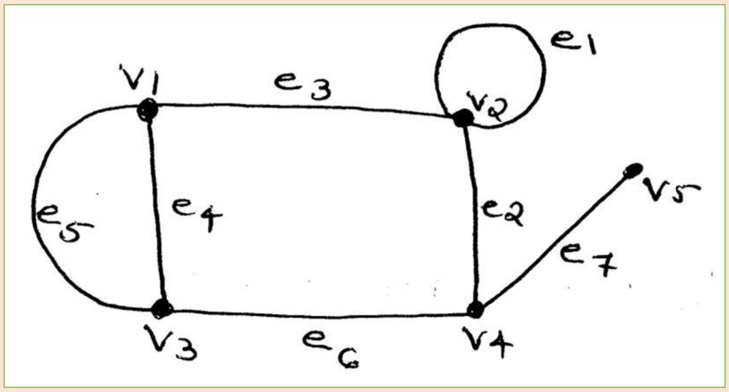
GRAPH

- Graph is a nonlinear data structure.
- Graph is shown as a tuple **G=(V , E)**

  V = set of vertex = $\{v_1, v_2, v_3, \ldots\ldots. v_n\}$

  E = set of edges = $\{e_1, e_2, e_3, \ldots\ldots. e_n\}$
- Graph has no hierarchal relationship.
- If a tree contains a closed loop or circuit , it is a graph.
- Tree is an acyclic graph

Eg:

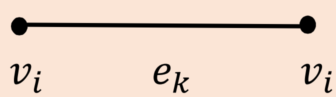


- Edges E = {e_1, e_2, e_3,……. e_7}
- Vertices V = {v_1, v_2, v_3, v_4 ,v_5}

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 53

➢End Vertices

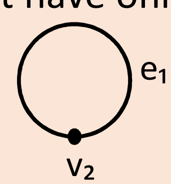
- The two vertices associated with any edge are called end vertices



v_i and v_j are the end vertex of e_k

➢Self Loop

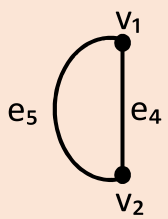
- Any edge that have only one vertex are called self loop



Prepared By Mr.EBIN PM, AP, IESCE EDULINE 54

➢Parallel Edges

- In some cases, two edges are present associated with 2 vertex. This edges are called parallel edges.



v_1, v_2 – vertices

e_4, e_5 – parallel edges

➢Simple Graph

- A graph with neither self loop nor parallel edges are called simple graph.

➢Isomorphic Graph

- One graph is able to represent several styles is called isomorphic graph.

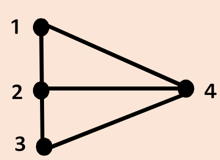


Fig (a)

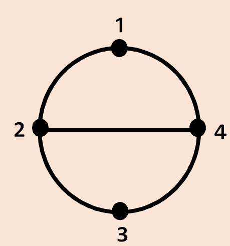
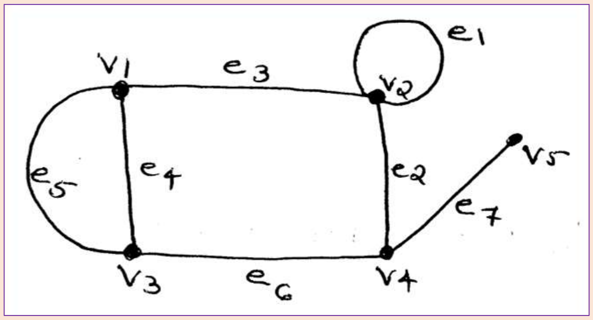


Fig (b)

- Graph (a) and (b) are same , because edges and vertices are same in both graphs.

➢Degree (valency) of graph

- The number of edges incident on a vertex v_i , with self loops counted twice is called the degree **d (v_i)** of vertex v_i



$d(v_1) = d(v_3) = d(v_4) = 3$

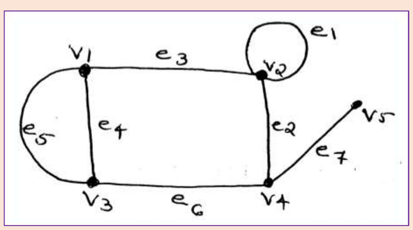
$d(v_2)=4$ (self loop counted twice)

$d(v_5) = 1$

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 57

❖**Theorem -1** **:** The sum of degree of all vertex in a graph G is twice the number of edges in the graph G

That is $\sum_{i=1}^{n} d(v_i) = 2e$



The number of edges = 7

Sum of degree of all vertices =

$d(v_1) = 3$

$d(v_2) = 4$ = (3+4+3+3+1) = **14**

$d(v_3) = 3$ $\sum_{i=1}^{n} d(v_i)$ = 14 = 2e

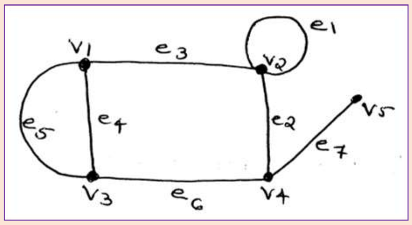
$d(v_4) = 3$ (One edge will contribute

$d(v_5) = 1$ for two degree counts)

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 58

❖**Theorem -2 :** The number of vertices of odd degree in a graph is always even.

Consider the following graph



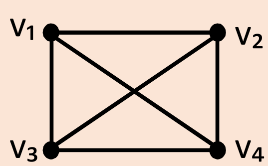
The total number of vertex which have odd degree is 4.

That is, v_1 , v_3, v_4, v_5

4 is an even number

➢**Regular Graph**

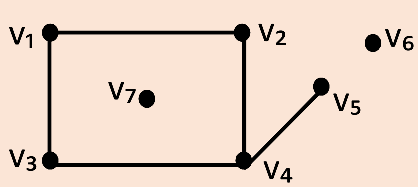
- If the degree of all vertex is same, it is called regular graph.



Degree of all vertices are 3. So it is a regular graph

➢**Isolated Vertex**

- A vertex having no incident edge or no edge will be incident to a vertex is called isolated vertex

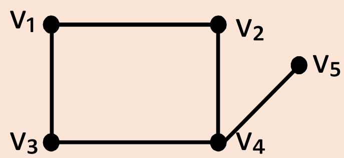


V_6 and V_7 are isolated vertices

Degree of isolated vertex is zero

➢Pendant Vertex

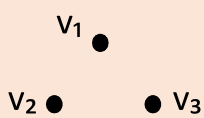
- A vertex of degree one is called pendant vertex or an end vertex



v_5 is pendant vertex; degree = 1

➢Null Graph

- A graph without any edge is called null graph. In other words every vertex in a null graph is an isolated vertex



is a null graph

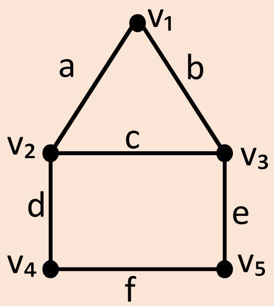
❖Applications of Graphs

1. Graph is used to represent communication network
2. It is also used to represent electrical networks
3. Seating problem solving
4. Graph coloring
5. Project planning
6. Solving puzzles

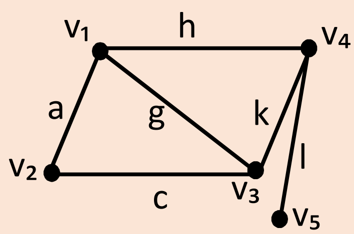
❖Operations on Graph

The main operations on graph are:

- **Union**
- **Intersection**
- **Ring sum**



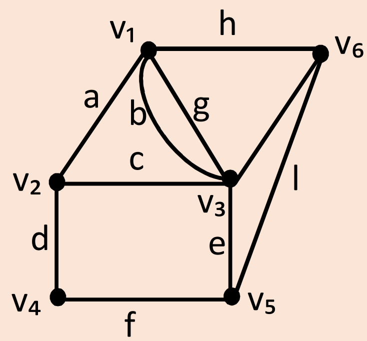
G1



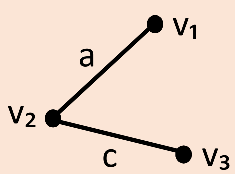
G2

Union

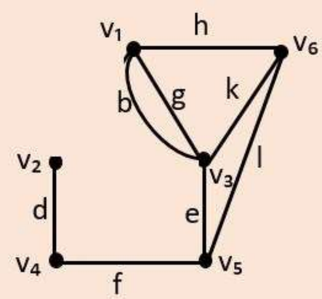
G1 U G2

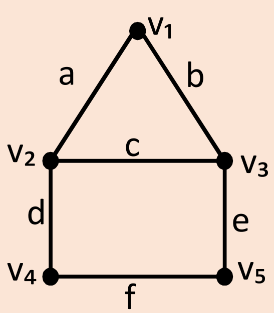


Intersection (G1 ∩ G2)

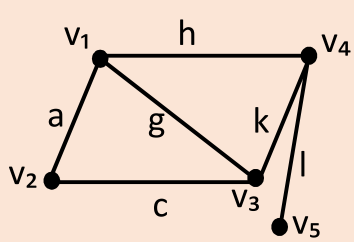


Ring Sum (G1 ⊕ G2)





G1



G2

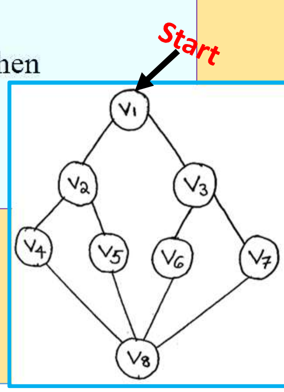
GRAPH TRAVERSALS

- 2 type traversals are performed on a graph. They are
- **Depth First Search (DFS)**
- **Breadth First Search (BFS)**
- For performing DFS, we use stack and queue for BFS

DFS

Once at a vertex V , we scan all edges incident on V and then move to an adjacent vertex W. At W, we scan all edges incident on W. This process is continued till all edges in the graph are scanned.


```
Algorithm DFS (V)
{
  // given an undirected graph G (V, E)
     with n vertex and array visited
     of n initially set to false.
  visited [V] = true;
  for each vertex W adjacent to V do
    {
      if not visited [W] then
          DFS [W];
    }
} // end of algorithm
```



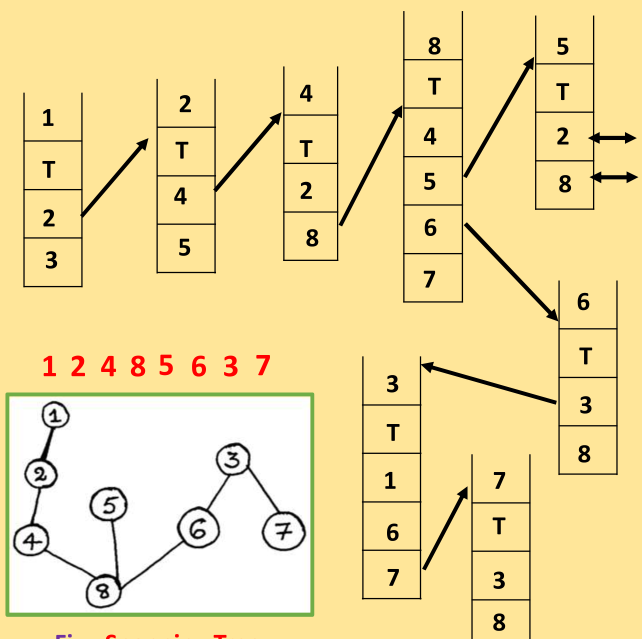
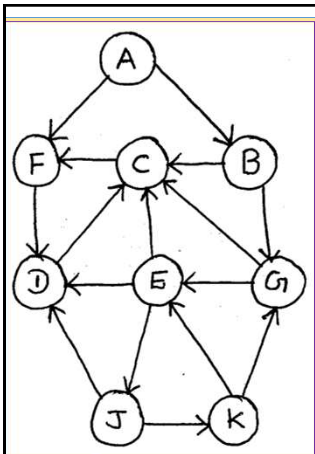


Fig : Spanning Tree

DFS



Start

J D C F K E G

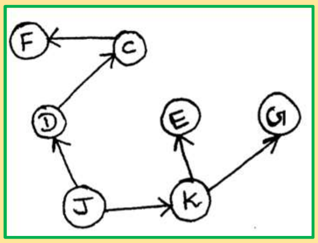
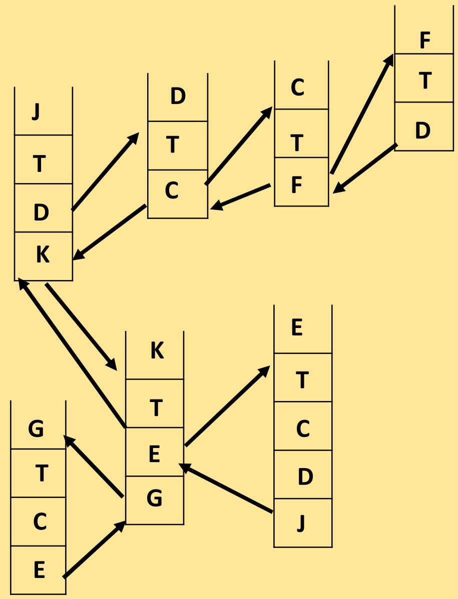


Fig : Spanning Tree

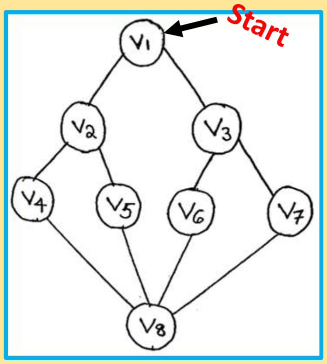


❖BFS

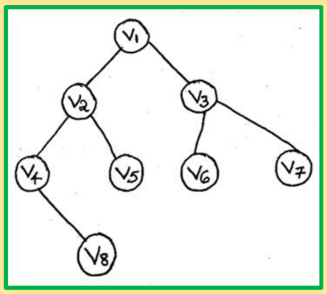
```
Algorithm BFS (V)
{
  // A BFS of G (V, E) is carried out beginning at vertex V and
    array visited of n initially set to false.

   visited [V] = true;
   initialize Queue [Q];
   add (Q, V);
   while not empty Queue[Q] do
      {
        delete (Q, V);
        for all vertex to adjacent do
          {
            if not visited [W] then
              {
                add [Q, W];
                visited [W]=true;
              }// end of if
          }// end of for
      }// end of while
 }// end of algorithm
```

Eg:

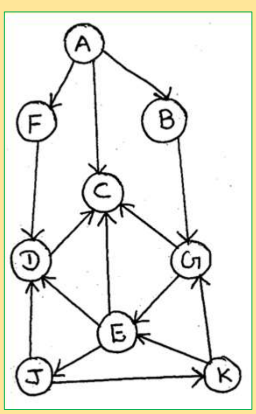


VISITED	VERTEX VISITED	ADJACENT	Q
T	v1	v3 v2	v1
T	v2	v5 v4 v3	V1 v2
T	v3	v7 v6 v5 v4	V1 V2 v3
T	v4	v8 v7 v6 v5	V1 V2 V3 v4
T	v5	v8 v7 v6	V1 V2 V3 V4 v5
T	v6	v8 v7	V1 V2 V3 V4 V5 v6
T	v7	v8	V1 V2 V3 V4 V5 V6 v7
T	v8	---	V1 V2 V3 V4 V5 V6 V7 v8



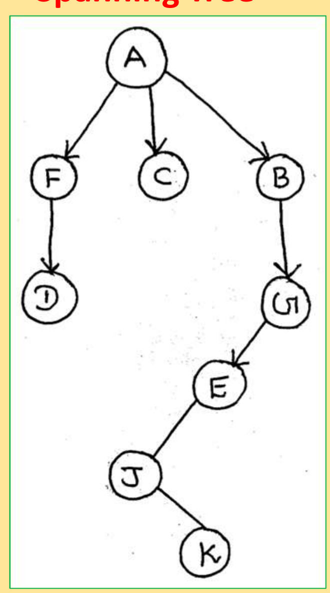
Spanning Tree

Eg: 2



VISITED	VERTEX VISITED	ADJACENT	Q
T	A	B C F	A
T	F	D B C	A F
T	C	D B	A F C
T	B	G D	A F C B
T	D	G	A F C B D
T	G	E	A F C B D G
T	E	K J	A F C B D G E
T	J	K	A F C B D G E J
T	K	-	A F C B D G E J K

Spanning Tree



GRAPH REPRESENTATION IN MEMORY

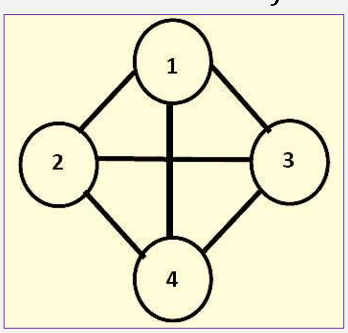
The most commonly used representations are

- Adjacency Matrix
- Adjacency List
- Adjacency Multilist
- Sequential Representation

1. Adjacency Matrix

- The graph can be represented in a matrix form.
- Let G=(V, E) be a graph with “n” vertices. The size of the adjacency matrix G is **n×n**.

- A[i , j] =1 indicate that an edge is present between the vertex v_i and v_j.
- A[i , j] = 0 indicate that an edge is not present in between the vertex v_i and v_j.



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

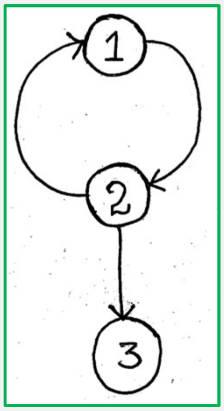
n=4

Size of matrix = 4×4

Degree of vertex v_1 = number of ones in row or column in 1. That is 3

- If the graph is undirected, the obtaining matrix is a symmetric matrix. So we can store the upper portion or the lower portion of the matrix in memory.

Consider the following directed graph



$$\begin{array}{cc} & \begin{array}{ccc}1 & 2 & 3\end{array} \\ \begin{array}{c}1\\2\\3\end{array} & \begin{bmatrix}0 & 1 & 0\\1 & 0 & 1\\0 & 0 & 0\end{bmatrix}\end{array}$$

The matrix is not symmetric. The row sum of the matrix indicate the "out degree" and the column sum indicate "in degree"

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 73

Out degree of V1 = 1

In degree of V1 = 1

Out degree of V2 = 2

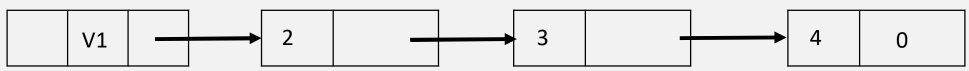
In degree if V2 = 1

- If we represented a graph (Directed or undirected) in the adjacency matrix form, the diagonal elements are always zero. So we can avoid the searching of diagonal elements.
- Number of search = n^2-n
- Time complexity = **$O(n^2)$**
- The searching time can be further reduced as O(n+e) where, n is number of vertices and e is number of edges. For reducing the time complexity of adjacency matrix, we use adjacency list.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 74

❖Adjacency List

- In adjacency list, head nodes and list nodes are present. The number of head node is equal to the number of vertex.
- Head node can be represented in a sequential manner or in an array form. Each head node contains a set of list nodes.



- V1 is the head node. The above figure represents, from V1, there is an edge present to V2, V3 and V4

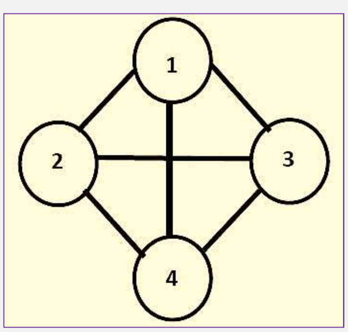


Fig : (a)

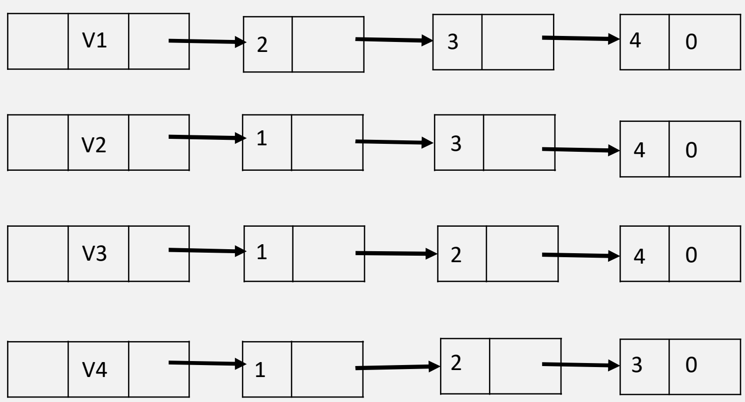
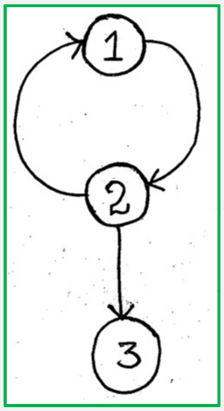
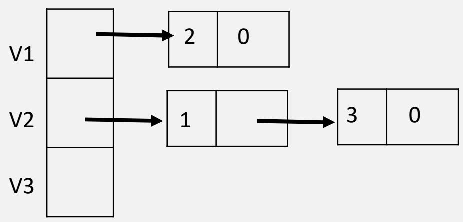


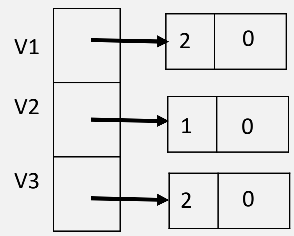
Fig : Adjacency list of (a)

- Consider the directed graph. Here 2 separate lists are used for representing “in degree” and “out degree”





Out degree representation using adjacency list



In degree representation using adjacency list

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 77

- Adjacency list representation is efficient when the graph is sparse.
- Consider an undirected graph of n vertices

  Number of head nodes = n

  Number of list nodes = 2e

For representing each node, we need the space of log n. For representing list node, the space needed is (log n + log e)

So, Total space requirement is

n log n + 2e (log n + log e)

In the above figure, there are 3 list nodes present with V1. So the degree of V1 is 3.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 78

❖Adjacency Multilist

- In the case of adjacency list, the total number of list node is 2e, because one edge contribute 2 nodes.
- For reducing the number of list nodes, we introduce adjacency Multilist. The list node structure is

N	Vertex 1	Vertex 2	Path 1	Path 2

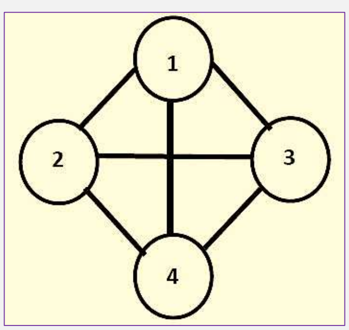
- So each head node contains only ‘e’ number of list nodes, not 2e.
- N is a mark bit used for analyze the edge.

N	1	2	N2	N4

N2 contains the next information about 1

N4 contains the next information about 2

Eg:



No.of edges = 6. So,
total number of list node= 6

1
2
3
4

N1	1	2	N2	N4
N2	1	3	N3	N4
N3	1	4	0	N5
N4	2	3	N5	N6
N5	2	4	0	N6
N6	3	4	0	0

- If 1, 2 is included , 2,1 is also embedded. So no need to write 2,1 separately.
- The corresponding path is

  V1 = N1→ N2 →N3

  V2 = N1 →N4 →N5

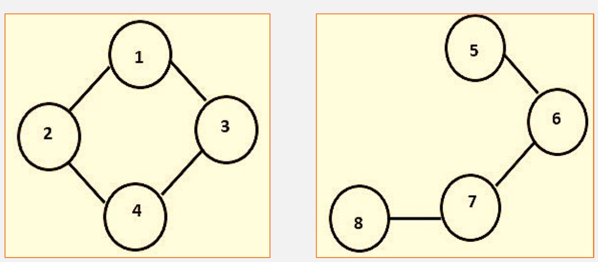
  V3 = N2 →N4 →N6

  V4 = N3 →N5 →N6

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 81

❖Sequential Representation

- Consider a graph of ‘n’ vertex and ‘e’ edges. The total space needed to represent sequentially is , n+2e+1

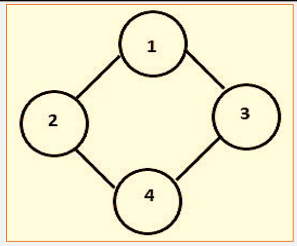


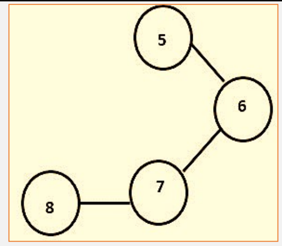
Total space needed is n+2e+1
= 8+(2×7)+1
=23

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 82

1	10
2	12
3	14
4	16
5	18
6	19
7	21
8	23
9	23
10	2
11	3
12	1
13	4

14	1
15	4
16	2
17	3
18	6
19	5
20	7
21	6
22	8
23	7





- Last space is 23 .
- Total number of nodes = 8
- So we use 9 for 23 (9^{th} representation is the end of graph)
- 10,11 contains the adjacent nodes of 1. (that is 2 and 3)
- 12,13 contains the adjacent nodes of 2. (that is 1 and 4)

and so on.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 83