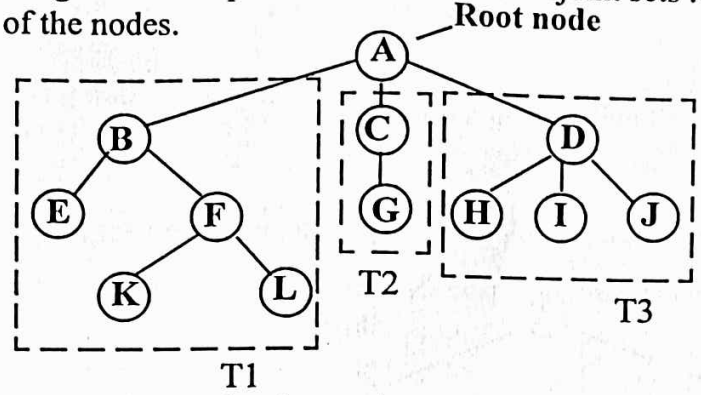
# UNIT-III
# TREES

**Introduction:** A tree is a finite set of one (or) more nodes . such that

* There is a specially designated node called root node.
* Remaining nodes are partitioned into " n " disjoint sets .T1,T2.....Tn are called sub trees of the nodes.

Ex:




**Root node:-** The first node of a tree is called as root node.Every tree must have a root node , which is only one.

ex:-In the above figure, 'A' is the root node.

**Parent:** Parent of a node is the predecessor of a node.

ex:- In the above diagram B is the parent node of E,F.

**Child:** The immediate successor of a node is called as child

ex:-In the figure H,I,J childrens of D

**leaf node:** A node with no children is called as leaf node

ex:-In the above diagram E, K, L , G, H ,I ,J,

**Non terminal nodes:-** A node with a child node is called as non terminal node,.

ex:-In the above example B,C,D,F. are non terminal nodes

**Sibling:-** The children of same parent node is called as sibling

ex:-In the above example H,I,J are siblings of parent D

**Edge:-** The link between a parent node and it's child is called as edge

ex:-In the above example ( A,B ) ( A,C) (A,D) ( B,E)..... are edges

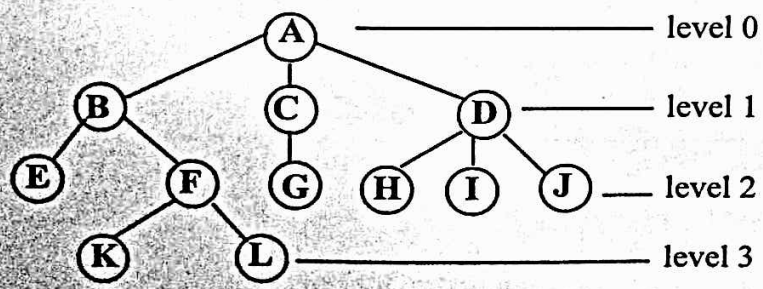
**Degree of a node:-** The no of child nodes for a given node is called as degree of a node.

ex:- From the above figure degree of a D is a 3, degree of F is 2

**Degree of tree:** Maximum no.of children that is possible for a node in a tree is known as degree of a tree.

In the above tree degree of tree is '3 '

**Level:-** Level is the length of the hirarchi from root node. root node is termed as in level " 0" . if a node is at level L then it's child is at level L+1, and parent is at level L - 1.This is true for all the nodes except the rootnode.



**Height:-** Maximum number of nodes that is possible in path starting from root node to leaf node is called as height of the tree.

**Ex:** In the above figure height of the tree is 4 i.e

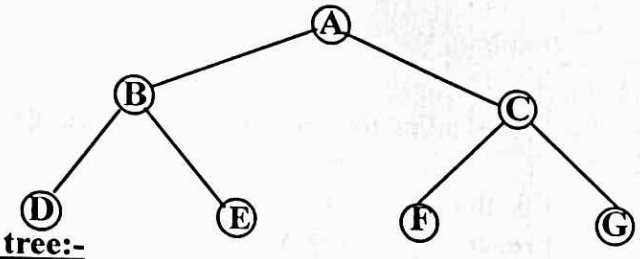
A => B=> F=> K

Degree of tree:-

**Generation:** All the nodes at particular level is said to be same generation

EX: in the above tree E,F,G,H,I,J are same generation

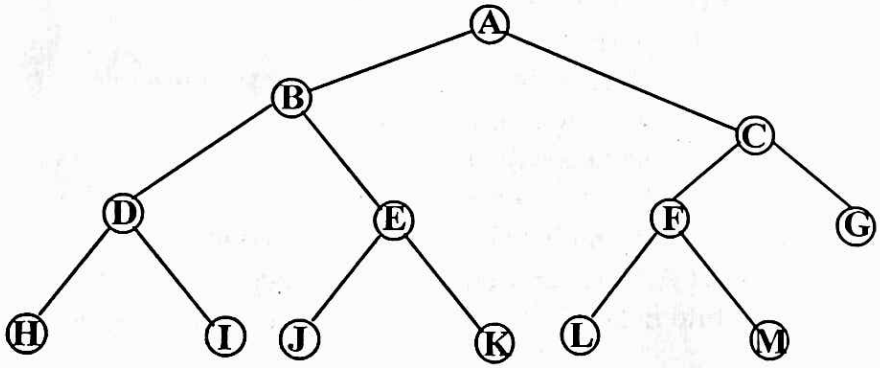
**Binary tree:** A binary tree is a tree ,in which a every node must have children less than or equal to 2 (0,1,2)




**Complete binary tree:-**

A binary tree is a said to be complete binary tree if all it's level except the last level have the maximum no.of the nodes.The bottom level is filled from left to right.

EX:

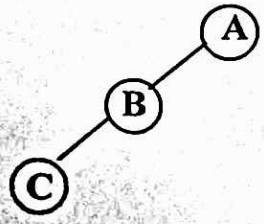



**Properties of binary tree:**

* In any binary tree maximum no.of nodes in a level L is 2L where L>= 0
* Mxamium no.of nodes possible in a binarytree of height ' h ' is 2h-1.
* Minimum no.of nodes possible in a binary tree of height ' h ' is 2h.

**Left skewed binary tree:-**

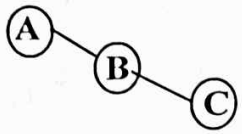
A binaty tree having left sides node only is called as left skewed binary tree



***Right skewed binary tree:-**

A binary tree having right side node only is called as right skewed binary tree.

ex:-




**note:-** For any non empty full binary tree, if n is the number of nodes and c is number of edge then n = e+1

**Explain Representation of binary tree ?**

There are two common methods used for representing binary trees.They are

* Linear representation using arrays .
*Linked representation using pointers

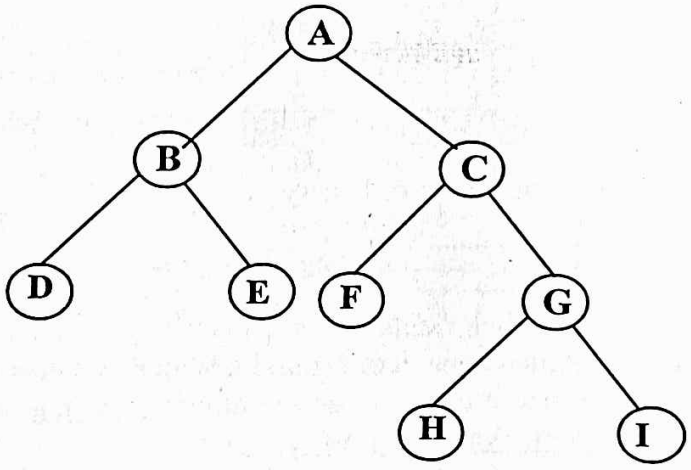
**Linear representation using arrays:**

This type of representation is static i.e is a block of memory for an array is to be allocated before going to store the actual tree values.

In this representation the nodes are stored level by level i.e; starting from the zeroth level where only root node is present.The root node is stored in the first memory allocation. In this representation the array index starts from one following the rules can be used to decide the location of any node of a tree in the array.

1. The root node is at location one
2. For any node with index ' i '
   a) PARENT ( i ) = [i / 2]; for the node when i=1 there is no parent.
   b)LCHILD ( i ) = 2*i ; if 2*i > n then ' i ' has no left child
   c)RCHILD ( i ) = 2*i +1 ; if 2*i+1 > n then ' i ' has no right child

Example:-




(a) Binary Tree

| A | B | C | D | E | F | G | ..... | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 9 10 11 12 13 | 14 | 15 |

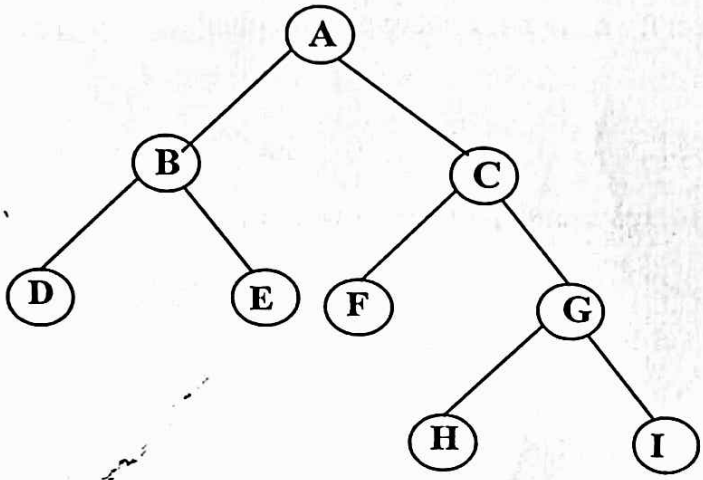
(b) Array representation of binary tree

A binary tree of height 'h' can have atmost $(2^h - 1)$ nodes.so the size of the array to fit a binary tree is $(2^h - 1)$ locations.

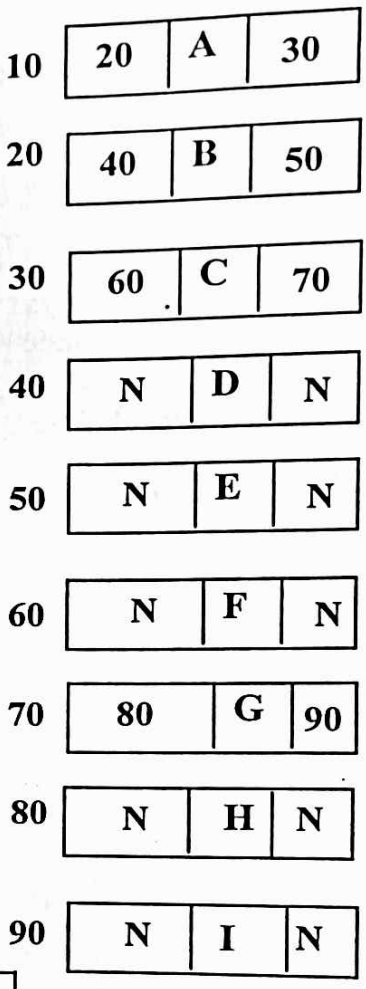
Example:- If h=3, the size of the array is $2^3 - 1 = 8 - 1 = 7$ locations.

**Linked representation of binary trees:-**

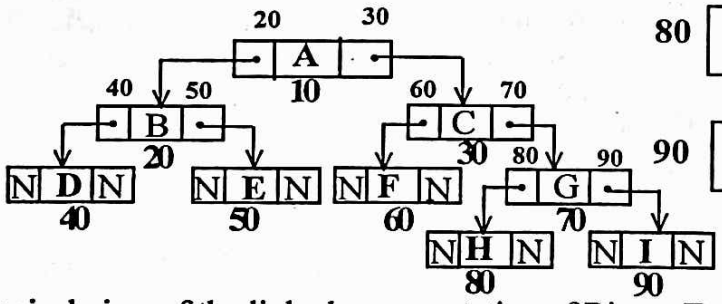
In linked representation of binary trees, we use node. The general representation of node as shown below




(a) Binary Tree




(B) BINARY TREE AND ITS VARIOUS NODES (PHYSICAL VIEW)




(c) Logical view of the linked representation of Binary Tree


In the above representation two link fields are used to store address of left child and right child of a node. Data is the original content of the node. With this representation, if we know the address of the root node then it is easy to access other nodes in a binary tree.

**Advantages of sequential representation of a binary tree :-**

**1)Data are stored only without any pointers to their success or ancestor which are mentioned implicently.**

**2)Any node can be accessed from any other node by calculating the index and this is efficient from execution point of view.**

**Disadvantages of sequential representation of a binary tree:-**

1)Other than full binary tree, majority of the array entires may be empty.
2)It allows static representation there is no way to enhance the tree structure.

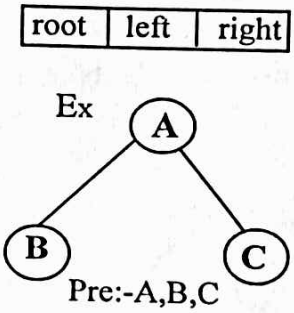
**Explain Tree traversal methods ?**

**Tree traversal methods** Traversing means to visit each and every node in the tree exactly once. There are three types of traversal methods. They are

1) Pre order traversal
2) In order traversal
3) post order traversal

**Pre-order traversal:-** In this traversal, root is first visited and then left sub-tree in pre-order traversal and then right sub-tree in pre order traversal. It can be defined as follows

* Visit the root node 'R'
* Traverse the left sub-tree of 'R' in pre-order
* Traverse the right sub-tree of 'R' in pre-order.

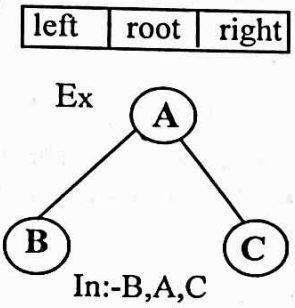



Algorithm:

```
pre-order(root)
step1: start
step2: if(root) then
          visit " root.data "
          pre-order(root.l link)
          pre-order(root.r link)
step3: stop
```

**In order tranversal :-**In this traversal, first visit the left sub-tree in order traversal and then visit the root node. and then visit the right sub-tree of a root node in order traversal. It is defined as follows

* Traverse the left sub-tree of the root node 'R' in Inorder.
* Visit the root node 'R'.
* Traverse the right sub-tree of the root node 'R' in Inorder.

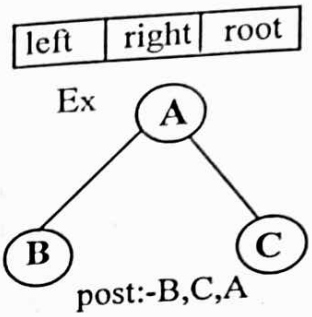



Algorithm:

```
in-order(root)
step1: start
step2: if(root) then
          in-order(root.l link)
          visit " root.data "
          in-order(root.r link)
step3: stop
```

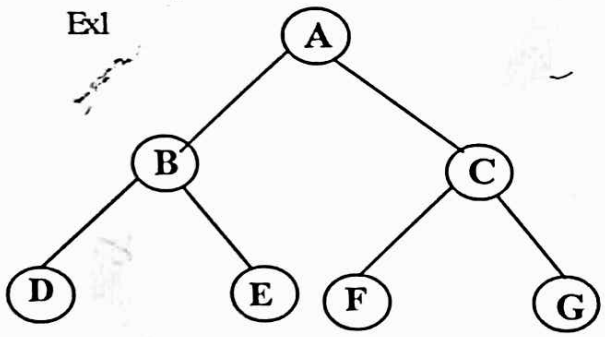
**Post-order Traversal:-** In this Traversal , first visit the left sub-tree and then right sub-tree, and at last visit the root node.It is defined as below

* Traverse the left sub-tree of the root node 'R' in post order
* Traverse the right sub-tree of the root node 'R' in post order
* Visit the root node 'R'.



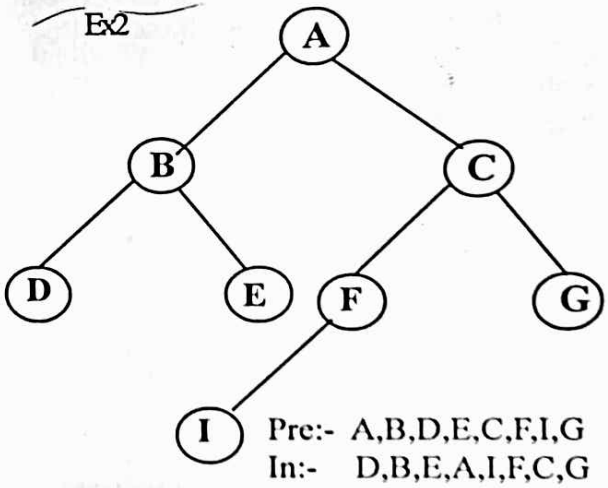

Algorithm:
post-order(root)
step1: start
step2: if(root) then
post-order(root.l link)
post-order(root.r link)
visit " root.data "
step3: stop

Ex1




pre:- A,B,D,E,C,F,G
In:- D,B,E,A,F,C,G
Post:- D,E,B,F,G,C,A

Ex2

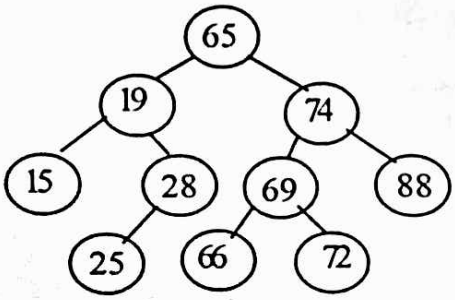



Pre:- A,B,D,E,C,F,I,G
In:- D,B,E,A,I,F,C,G
Post:- D,E,B,I,F,G,,C,A

**Explain Binary search tree ( BST ) ?**

A binary tree 'T' is called as binary search tree if each node 'N' of 'T' satisfy the following property.
"The value at 'N' is greater than every value in the left subtree of 'N' and value at 'N' is less than every value in the right subtree of 'N' ".

Ex:-



## Creation of Binary search tree:

**Binary search tree having two characteristics:**

(1) All the nodes in a left sub tree having values less then root node.

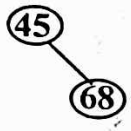
(2) All the nodes in a right sub tree having values greater then root node.

**Suppose we want to construct binary search tree for following set of data:**

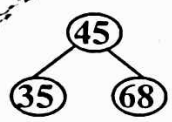
**45 68 35 42 15 64 78**

**Step 1:** First element is 45 so it is inserted as a root node of the tree.

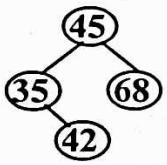

**Step 2:** Now we have to insert 68. First we compare 68 with the root node which is 45. Since the value of 68 is greater then 45 so it is inserted to the right of the root node.



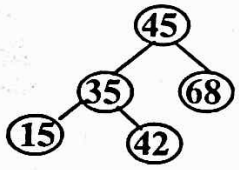

**Step 3:** Now we have to insert 35. First we compare 35 with the root node which is 45. Since the value of 35 is less then 45 so it is inserted to the left of the root node.



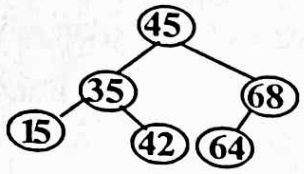

**Step 4:** Now we have to insert 42. First we compare 42 with the root node which is 45. Since the value of 42 is less then 45 so it is inserted to the left of the root node. But the root node has already one left node 35. So now we compare 42 with 35. Since the value of 42 is greater then 35 we insert 42 to the right of node 35.



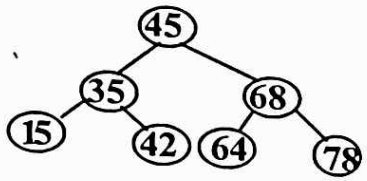

**Step 5:** Now we have to insert 15. First we compare 15 with the root node which is 45. Since the value of 15 is less then 45 so it is inserted to the left of the root node. But the root node has already one left node 35. So now we compare 15 with 35. Since the value of 15 is less then 35 we insert 15 to the left of node 35.




**Step 6:** Now we have to insert 64. First we compare 64 with the root node which is 45. Since the value of 64 is greater then 45 so it is inserted to the right of the root node. But the root node has already one right node 68. So now we compare 64 with 68. Since the value of 64 is less then 68 we insert 64 to the left of node 68.




**Step 7:** Now we have to insert 78. First we compare 78 with the root node which is 45. Since the value of 78 is greater then 45 so it is inserted to the right of the root node. But the root node has already one right node 68. So now we compare 78 with 68. Since the value of 78 is greater then 68 we insert 78 to the right of node 68.




**ALGORITHEM: BST - Creation**

```
step-1: start
step-2: ptr= root, flag=false
step-3:while (ptr != null) and (flag=flase) do
        case: ( item<ptr.data)
              ptrl = ptr;
              ptr = ptr.Lchild;
        case:( item>ptr.data)
              ptrl = ptr;
              ptr = ptr.rchild;
        case: ( ptr.data = item)
               flag = true;
        print " item already exist";
               Exist;
        end cases
        end while
step-4: if (ptr=null) then
        {
        new= GET NODE (NODE);
        new.data = item;
        new.Lchild =null;
        new.rchild = null;
        }
step-5: if ( ptr. data < item ) then
        ptrl.rchild = new;
        else
          ptrl.Lchild = new;
        end if
step-6: stop
```

We can perform the following operations on binary search tree.

1) Searching a node
2) Inserting a node
3) Deleting a node

**Searching a node:-**

Searching a particular node is available (or) not in a binary search tree is called as searching operation.

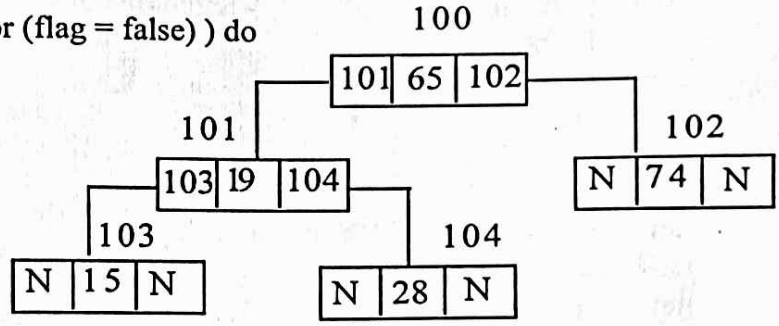
In a binary search tree 'T' , ' item' be the searching element. The searching procedure as follows.

searching starts from root node 'R'. If 'item' is less than the value in the root node 'R' , we proceed to its left child. if the 'item' is greater than the value in the root node 'R' , we proceed to its right child. This process will be continued until the item is found or not. If the item is found then it returns address of the node otherwise it display a message "item does not exit in the binary search tree".

Algorithm:- BST-search (item)

step 1 : start
step 2 : ptr = root , flag = False
step 3 : while ( (ptr ! = NULL) or (flag = false) ) do
case : ( Item < ptr.data )
ptr = ptr.child
case : ( Item = ptr.data )
flag = True
case : ( Item > ptr.data )
ptr = ptr.child
endcase
End while
step 4 : if ( flag = true ) then
print ("item has found",ptr)
else
print ("item does not exist in the tree");
end if
step 5 : Stop

For example use the following diagram

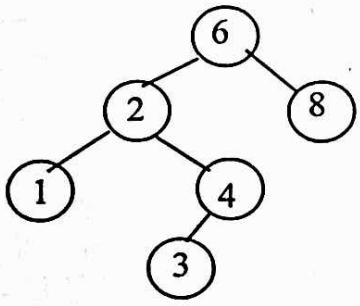



**Inserting a node:-**

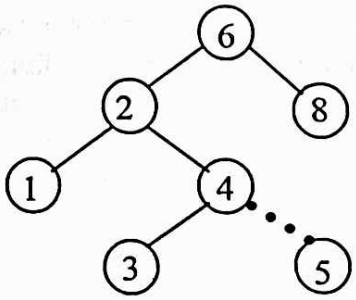
Inserting a node into binary search tree is called as insertion. To insert a node into binary search tree, the tree is to be searched starting from the root node. If item is not found then insert the new node at the dead end.

Example:-

Before insertion

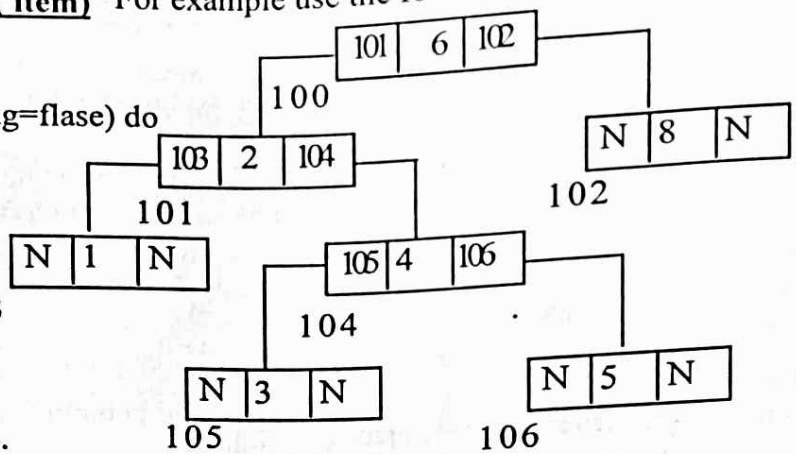



after insertion



In the above diagram , if we want to insert a node 5 in to binary search tree. So , search processor starts from root node like 6-2-4 then it halts when it finds that right child is null ( dead node). Simply we can insert the new node if the right link of the node '4'.

**ALGORITHEM: BST - insert ( item)** For example use the following diagram




```
step-1: start
step-2: ptr= root, flag=false
step-3:while (ptr != null) and (flag=flase) do
        case: ( item<ptr.data)
              ptrl = ptr;
              ptr = ptr.Lchild;
        case:( item>ptr.data)
              ptrl = ptr;
              ptr = ptr.rchild;
        case: ( ptr.data = item)
               flag = true;
        print " item already exist";
               Exist;
        end cases
       end while
step-4: if (ptr=null) then
        {
        new= GET NODE (NODE);
        new.data = item;
        new.Lchild =null;
        new.rchild = null;
        }
step-5: if ( ptr. data < item ) then
        ptrl.rchild = new;
        else
        `  ptrl.Lchild = new;
        end if
step-6: stop
```

**Deleating a node: -**

Delete a node from the binary search tree is called as deletion.suppose 'T' is a binary search tree and item is the information has to be deleated from 'T' . suppose 'N' be the node which contains the information item. the deletion is possible in two.

cases:

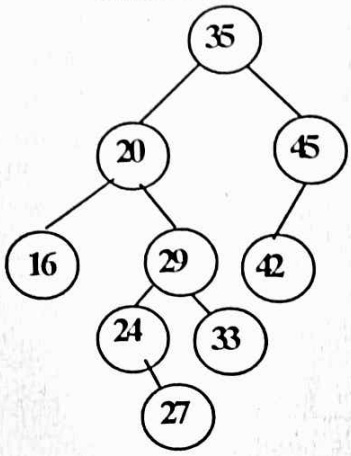
case 1: 'N' is the leave node ( or ) leaf node.

case 2:'N' has exactly one node

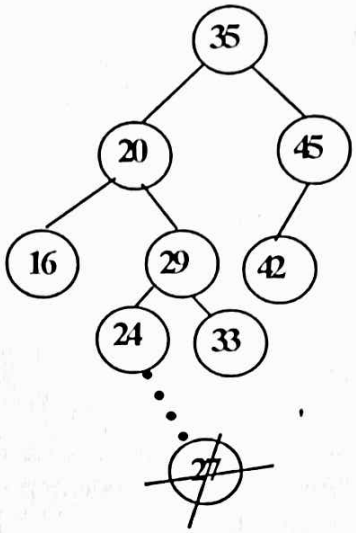
Examples:

case1

Before deletion



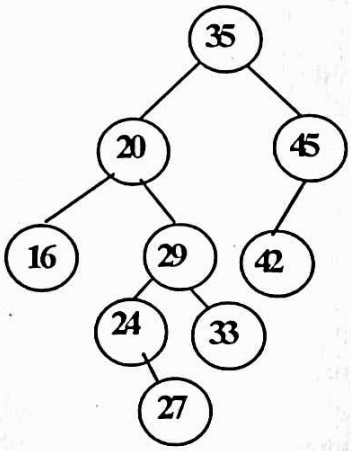

After deletion



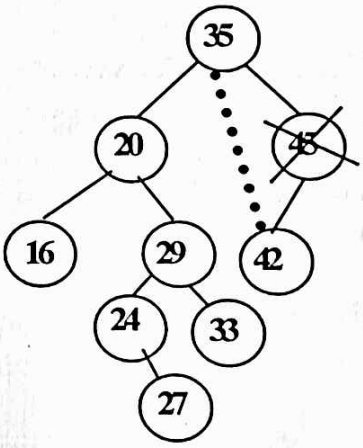

In the above case, '27' is deleted from tree by simply setting the address of 27 in the parent node of 27 by NULL.

case2:

Before deletion




After deletion




In the above case 45 is deleted from 'T' by simply replacing the address of 'N' in parent of '45' by the address of only child of 45

Algorithem: BST-delete ( item)

step-1: start

step-2: ptr = root, flag = false,

step-3: while ( ptr != Null) and ( flag= false ) do

```
        case: ( item< ptr . data )
              parent = ptr;
              ptr= ptr.Lchild;
        case: ( item > ptr, data )
              parent = ptr;
              ptr = ptr. rchild;
        case:( item = ptr. data)
```

```
        flag =true;
        End case;
        End while;
step -4 if ( flag = flase ) then
        print " item does not Exist : no deletion";
        Exist
        Exist if
step -5 if ( ptr.Lchild = Null ) and ( ptr.rchild = Null) then
        case =1
        else
         case = 2
        End if
step-6; if ( case = 1) then
        {
        if ( parent.Lchild = ptr) then
        parent.Lchild = Null;
        else
        parent.rchild = Null;
        end if
        return (node ptr):
        end if ;
step-7: if ( case =2 ) then
        {
        if ( parent . Lchild = ptr )then
        {
        if ( parent .Lchild = Null ) then
        parent . Lchild = ptr. r child;
        else
        parent.Lchild = ptr.Lchild;
        }
        else
        {
if ( parent . r child = ptr ) then
{       if ( ptr.Lchild = Null) then
        parent . rchild = ptr. rchild;
        else
          parent . rchild = ptr. Lchild;
        } }
        return ( node  ptr );
        }
step - 8 stop
```

**Applications of binary search tree:**

1. It stores data in hierachical manner .

Eg; file system

2.storing data naturally.

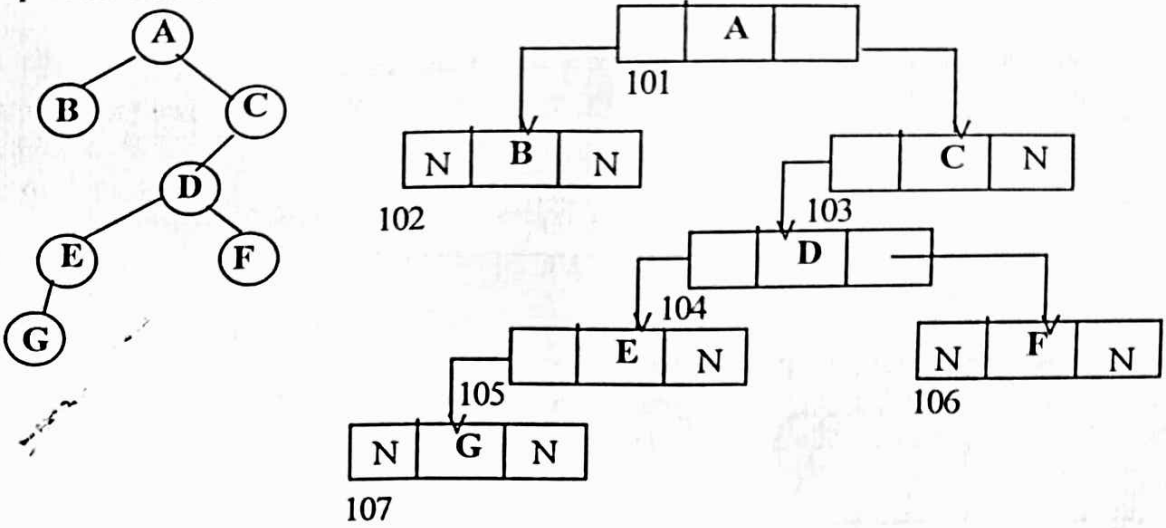
3.It is very easy to search a particular data in binary tree.

**Explain Threaded Binary trees ?**

In a binary tree more than 50% link fileds are with null values so, there is lot of memory waste in computer.

A clever way to utilise this null fields has been deviced by perlis and thornton. their idea is to store addresses of some nodes in unused link fileds. these extra addresses are called threads and the tree is known as threaded binary tree. In threded binary tree contains link to it's child or thread to some other node in the tree.

Representation of threaded binary tree:




For representing threaded binary tree, first decide to which node a thread should point. There are three ways to threaded a binary tree. They are:

1) In order threading
2) preorder threading
3) post order threading

**In order threading :-**

The threaded binary tree corresponds to in order transversal is called as inorder threading.For above tree, inorder traversal is B,A,G,E,D,F,C.

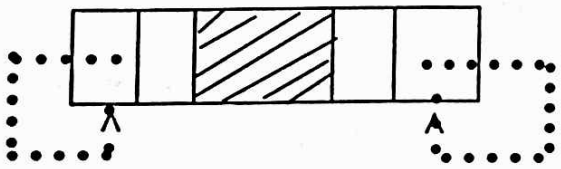
**Pre order threading:-**

The threaded binary tree corresponds to pre order traversal is called as pre order threading.

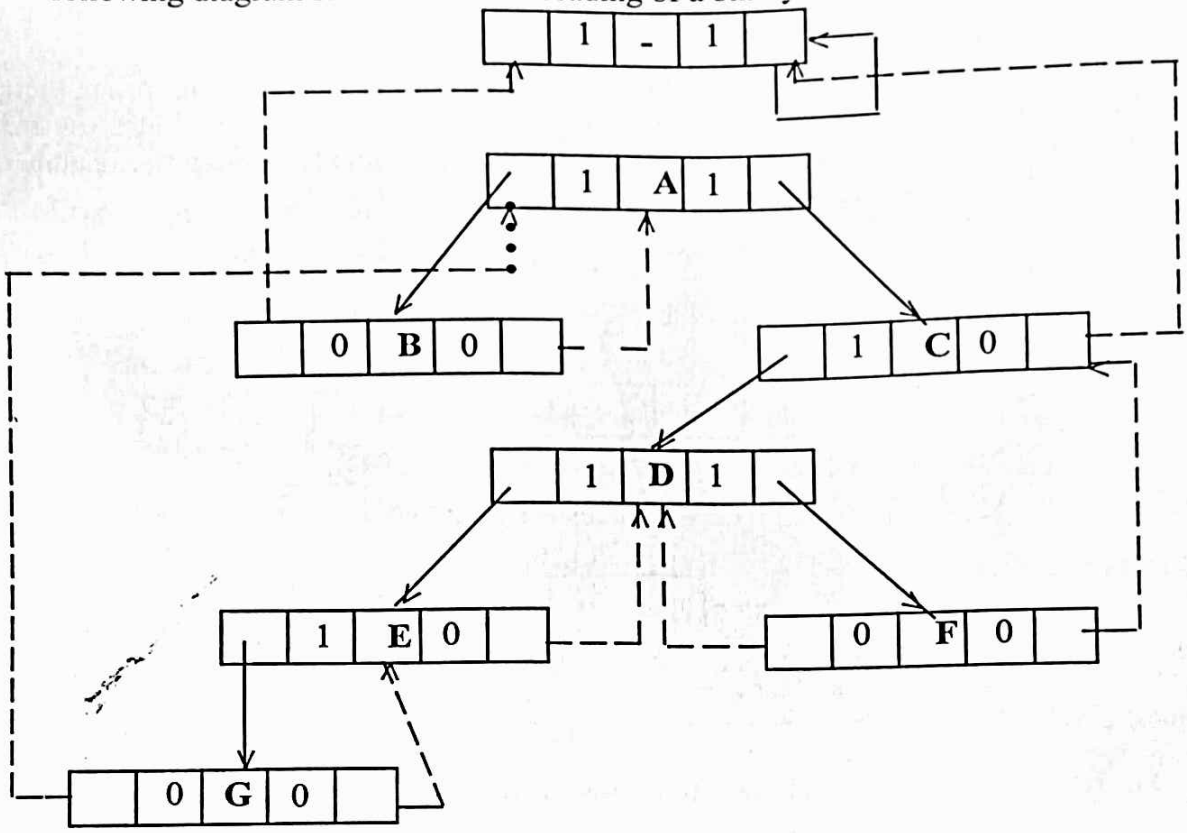
**Post order threading:-**

Threaded binary tree corresponds to post order traversal is called as post order threading.

The following figure shows an empty inorder threaded binary tree.



The following diagram shows inorder threading of a binary tree with a header node.




The above figure shows inorder threading of a binary tree.In the above figure links are represented by solid lines where as threads are represented by dotted lines ( or ) dashed lines.

The inorder traversal of a above binary tree is B,A,G,E,D,F,C

In order transversal of a binary tree, no left thread is possible for first node ( B )and no right thread is possible for last node ( C ).

To maintain the uniformity of setting of threads, to maintain a dummy node, this dummy node is called as header node. whose left link fileds is used to store the address of root node and the right link filed points to it self. Data content of header node is null and this node can be considered as the inorder predessor and successor of the first and last node respectively.

In representation of threaded binary tree, there is a problem of difference between a link and a thread.To do this, a simple node structure having LCHILD ( or ) RCHILD is not sufficient. it requires to extra fileds called 'LTAG' and 'RTAG'. The follwing diagram shows node structure in a threaded binary tree.

| LCHILD | LTAG | DATA | RTAG | RCHILD |
|---|---|---|---|---|

**The fields LTAG and RTAG will store either 0 or 1. The follwing assignment will be assumed to distingush between a link and a thread**

| | |
|---|---|
| LTAG = 0 | address of LCHILD is thread |
| LTAG = 1 | address of LCHILD is link |
| RTAG = 1 | address of RCHILD is thread |
| RTAG = 0 | address of RCHILD is link |

**Advantages of threaded binary trees :-**

1) In threaded binary tree any node can be accessable from any other node. Threads are move to upward where as links are downward. Thus in a threaded binary tree, one can move either directions.

2) The second advantage is, we can efficiently determined predessor and successor nodes storing from any node. In case of unthreaded binary tree, this task is more time consuming and different

**Disadvantage:-**

Insertions and deletions into threaded binary tree takes lot of time and it occupies more memory.

**Wriite a short notes on Heap Tree:**

A Heap tree is a specialized tree based data structure that satisfied the heap property .If B is a child node of A then value(A) is greater than or equal to value( B).This implies that an element with the greatest value is always in the root node, and so such a heap is called a Max-heap.The reverse is true for a Min-heap.

Example for Max-heap:-15,19,10,7,17,6

To construction of Max-Heap following the bellow steps

1.Create new node in the heap.

2.Assign value to the node

3.Compare value of child node with the parent node.

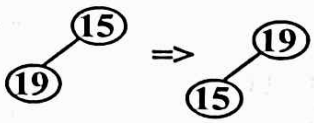
4.If parent < child then swap them

5.Repeat step 3 & 4 until heap property helds.

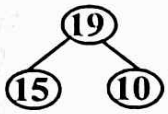
**Max-Heap tree** **Ex; 15,19,10,7,17,6**

Step 1:

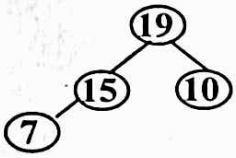

Step 2:



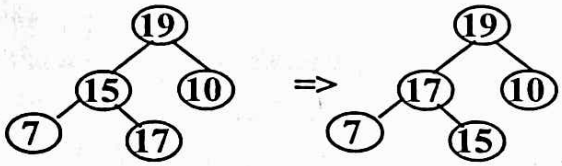

Step 3:



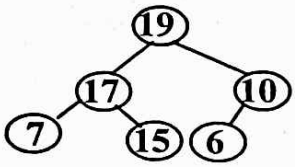

Step 4:




Step5:




Step 6



Example for Min-heap:-15,19,10,7,17,6
To construction of Min-Heap following the bellow steps

1.Create new node in the heap.

2.Assign value to the node

3.Compare value of child node with the parent node.

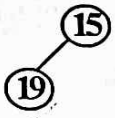
4.If parent > child then swap them

5.Repeat step 3 & 4 until heap property helds.

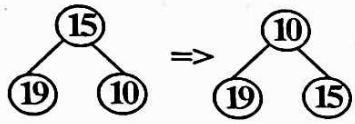
**MIN-Heap** **Ex; 15,19,10,7,17,6**

Step 1:

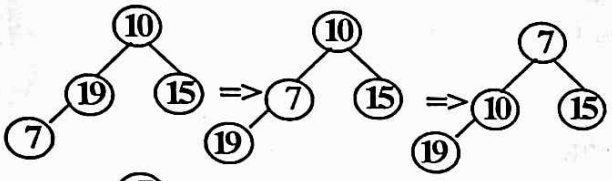

Step 2:



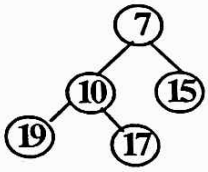

Step 3:




Step 4:




Step5:




Step 6