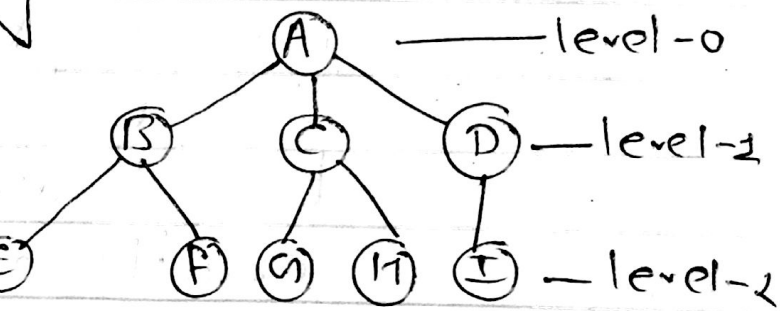
Tree (9)

= A tree is a non linear data structure. formally it is defined as, A tree is a finite set of one or more node such that

i) There is a special data item (node) called the root of the tree.

ii) And its remaining data item are partitioned into a number of disjoint set of node each of which is a tree and they are called subtree.

In big.



A is root and remaining node are partitioned into 3 disjoint set {B, E, F}, {C, G, H} and {D, I}, i.e No item is in more than one set.

Application of tree

1. The tree is used to create directory structure for stored file by OS.
2. It can be used for encoding message using Hoffman Tree.
3. It can be used for sorting data using a heap tree.

(4) It can be used for evaluating arithmatic expression.

Tree Terminologies

- Root: A root is the first in the hierarchial arrangement of data item. In big, A is the root.
- Node: A node that point to another nodes is parent of those nodes while the nodes pointed to are the children. Every node has exactly one parent.
- Degree of Node: It is the number of subtrees of a node. The degree of node A, B, C and D are 3, 2, 2 and 1 respectively.
- Level: The entire tree is levelled in such a way that the root is always ~~levelled~~ at level 0 and as we move down the tree, the level

of node increases in such a way that if a parent is at level n then its children are at level n+1.

In the above tree, A is at level-0, B, C and D are at level 1 and E, F, G, H and I are at level 2.

- Leaf Node: A node with no children is called leaf node/ terminal Node or External Node. Such node has zero degree. E is leaf Node.
- Non-terminal Node: Any node except the root node whose degree is not zero is called Non-terminal Node/ Internal Node. B, C and D are Non-terminal Node.
- Sibling / Brothers: The children nodes of a given parent node are called sibling. E and F are siblings of parent B, G and H are siblings of C.
- Edge: It is the connecting line between two nodes.
- Depth: The depth of the tree is the maximum level of any leaf in a tree. Depth is one more than the maximum level of tree.

In above figure maximum level = 2 and depth = 3. The depth is also called as height of tree.

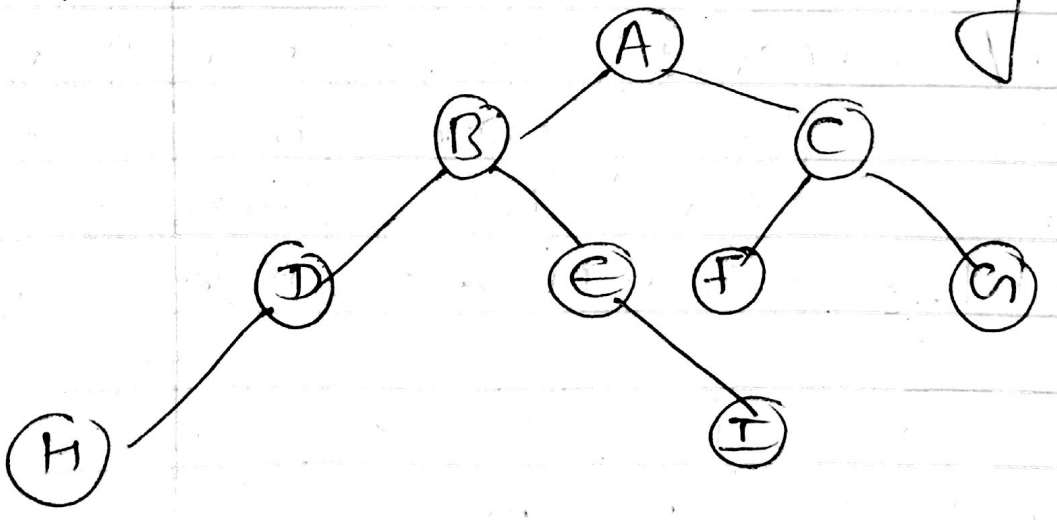
- Subtree: A portion of tree, which can be viewed as a tree itself is called subtree.

{B, E, F}, {C, G, H}, {D, I},

are the subtrees.

- All the leaf nodes are subtrees but all the subtrees are not leaf node.

Consider the following tree.



a) depth = 4

b) Children of B = {D and E}

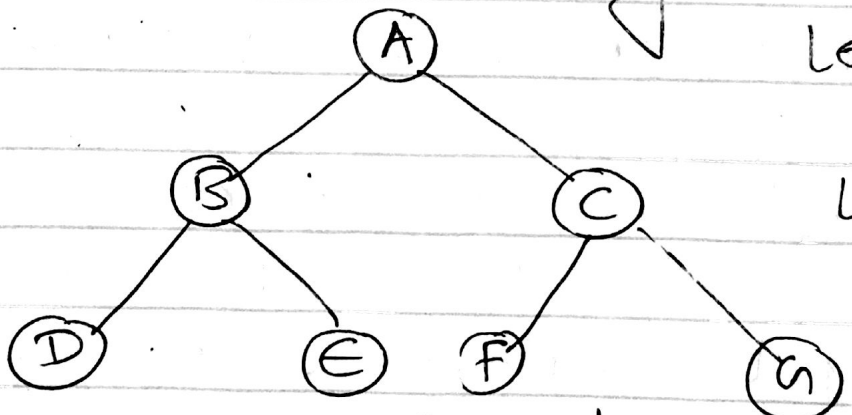
c) Parent of f = c
d) Level of node e = 2
e) Sibling of H = "It doesn't have sibling"
f) Sibling of D = NULL
g) leaf Nodes = { H, I, f and G }

Binary Tree

A binary tree is a set of zero or more nodes T such that

① There is a special data item called as root of the tree and

② The remaining nodes are partitioned into two disjoint set T_1 and T_2 each of which is a binary tree.

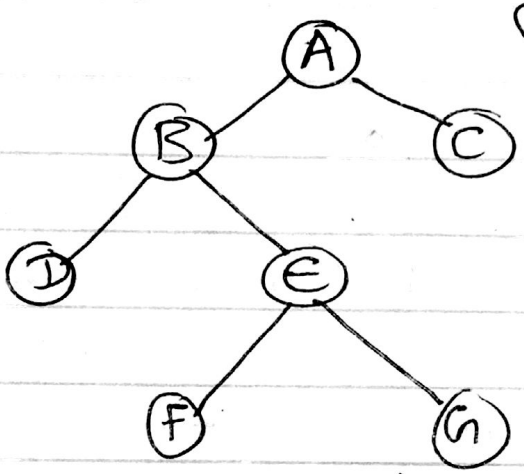


In a binary tree, the maximum degree of any node is atmost 2 i.e there may be a 0 degree, one degree and 2 degree.

TYPES

1 # Strictly Binary Tree

It every non leaf node in a binary tree has non empty left and right subtree then such tree is called strictly binary tree.



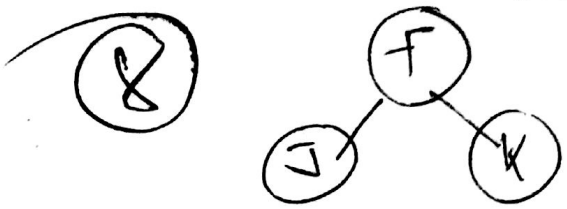
In the bigure, All the Non-terminal node such as B and E have non empty left and right subtree, so it is strictly binary tree.

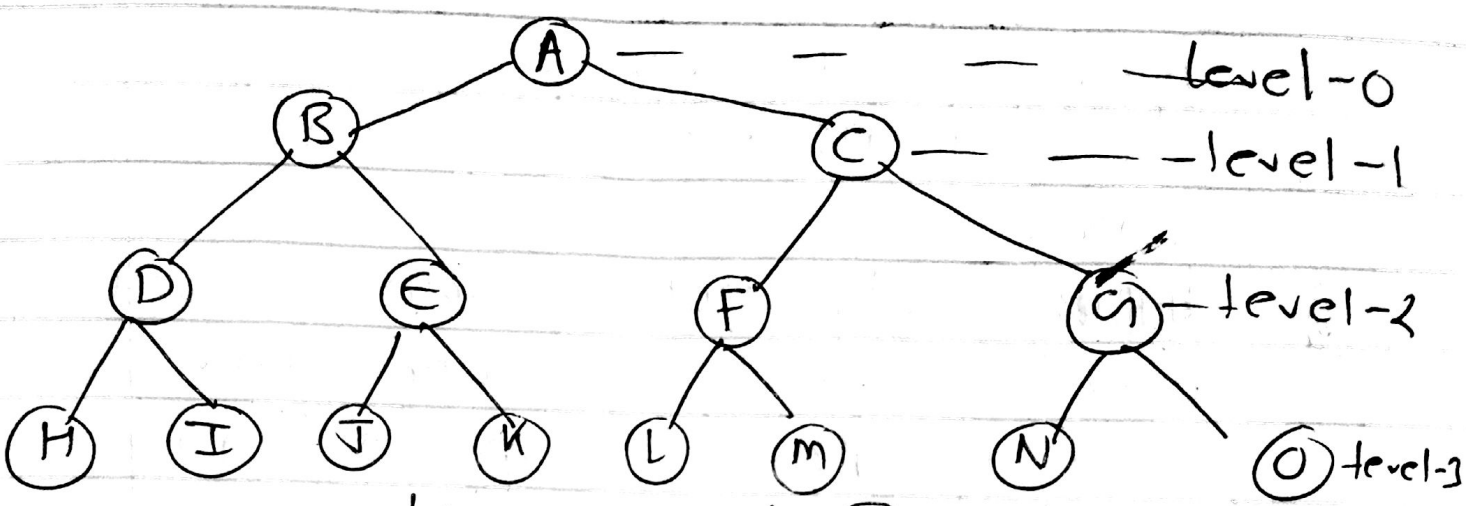
Q # — Complete / Full binary tree

A binary tree is complete binary tree if an only if

① Each Non-leaf node has exactly two children i.e must be strictly binary tree.

② All leaf nodes are at same level





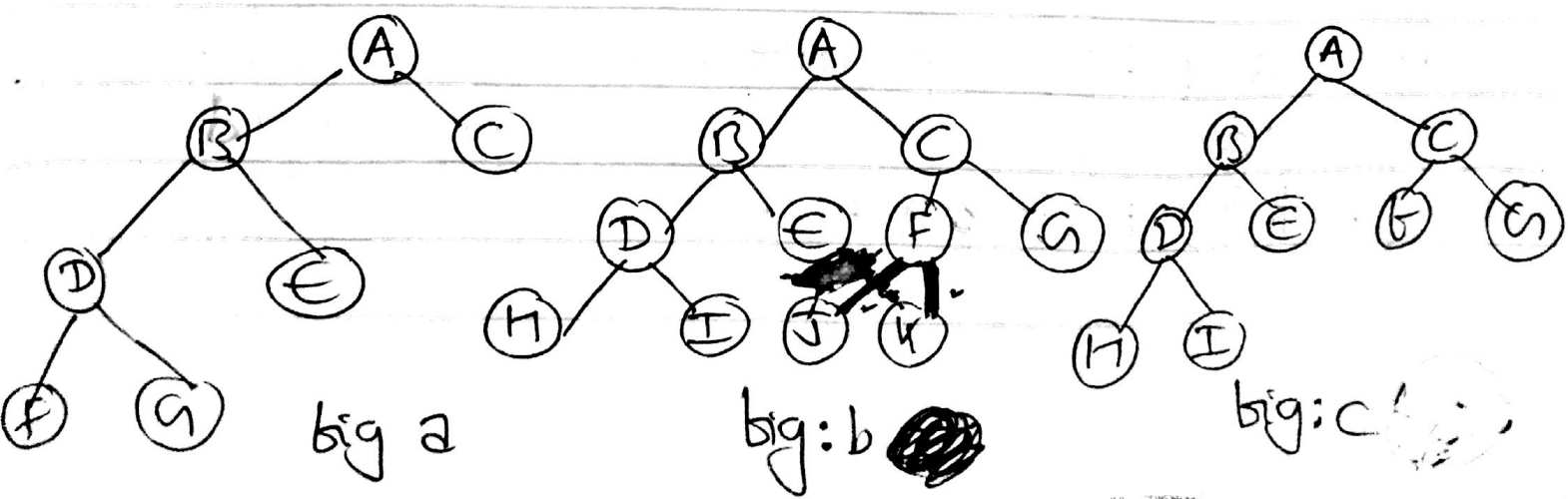
big: Complete Binary tree.

3# Almost Complete binary tree

— A binary tree of depth d is an almost complete binary tree if

a) Any node at level less than d-1 has two children.

b) For any node x, in the tree. with right descendent at level d, x must have left child and every left descendent have of x is either a leaf at level d or has two childven.



big a big: b big: c

big a, is not almost complele binary tree because it doesn't satisfy condition 1.

big b, is also not because it satisfy condition 1 but not condition 2.

big c, is almost complete binary tree.

4# Expression tree

An expression tree is a strictly binary tree in which leaf nodes contain opperands and the non leaf nodes contains operators, root node contain the operator that is applied to result of left and right subtree.

find exp tree for (a+b)*c

(1)

Eg: Construct expression tree for the following infix expression.

(a+b*c) / (+(c(d*e+f)*g))

Soln: Step 1: Separate operator and opperand

operator

(+*) + ((c*+)*))

Opperand

a b c d e f g

Step1: S :parale operator and opperand

(+*)+C(C*+)*)) abcdebt.

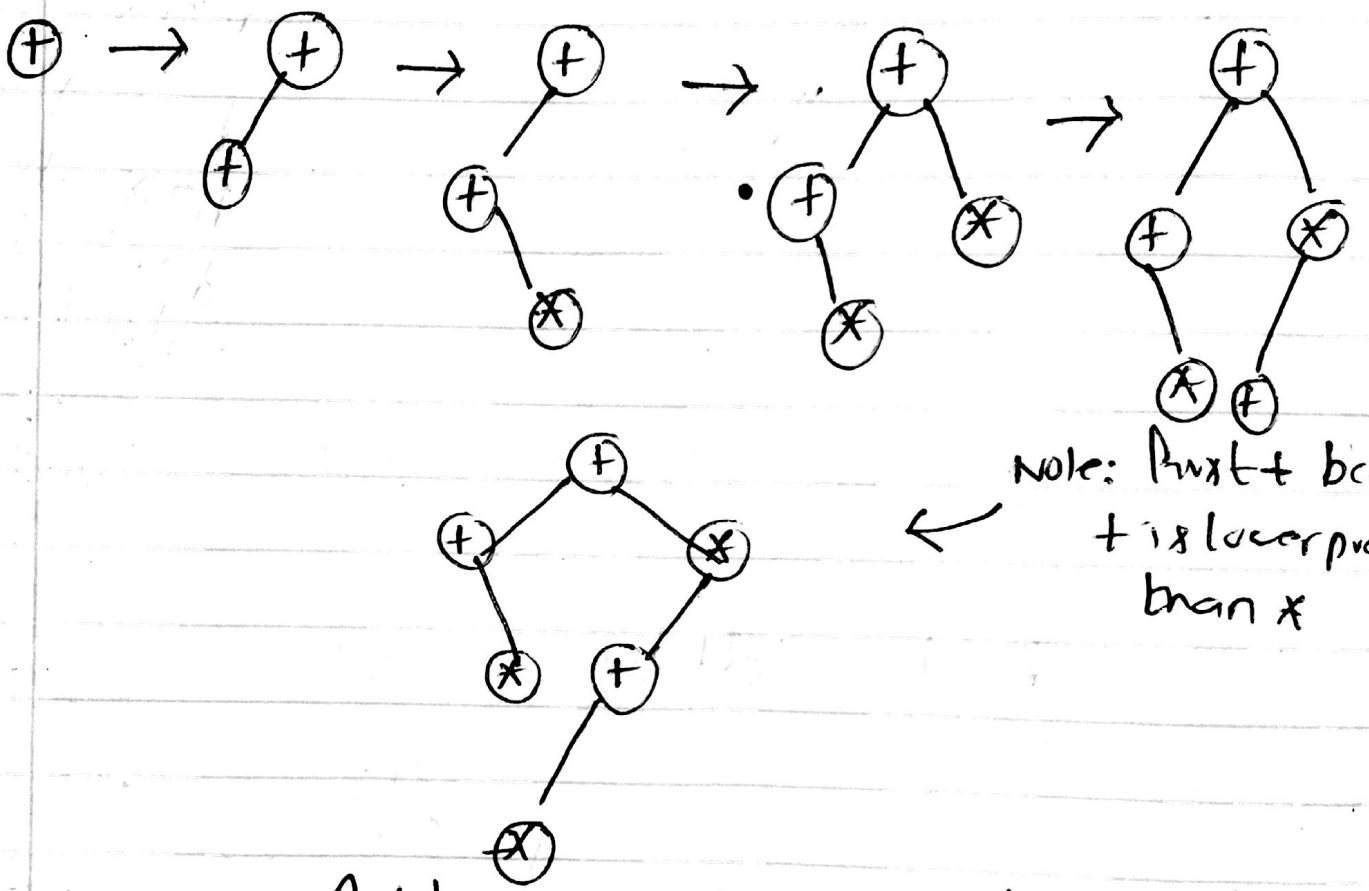
Step2: set precedence of operator.

\+
+* *
+

↑ higher

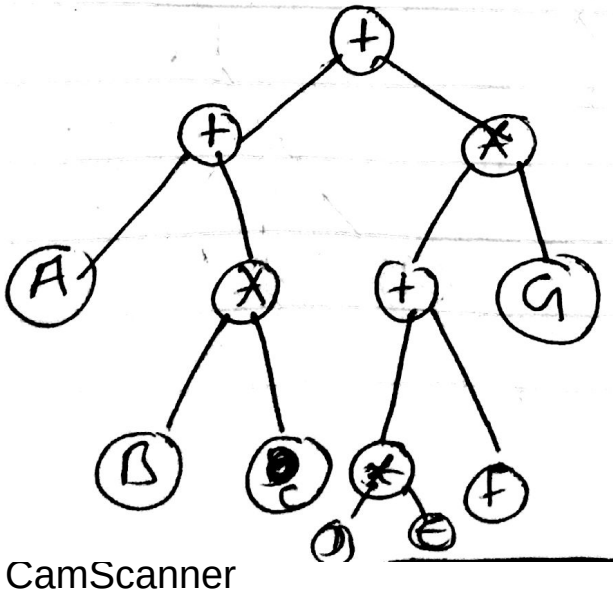
lower

Step3: Create expn tree with lowest priority operator as root.



Note: first + because + is lower prec. than *

Step4: Add opperand so as to satisfy strict binary tree.



∴ Inorder traversing

A+B*C+D*E+F*G

∴ Post order traversing

ABC*+DE*F+G*+

Construct Expression tree for the following infix expression

A + (B / C) * (D / E) - F

Soln:

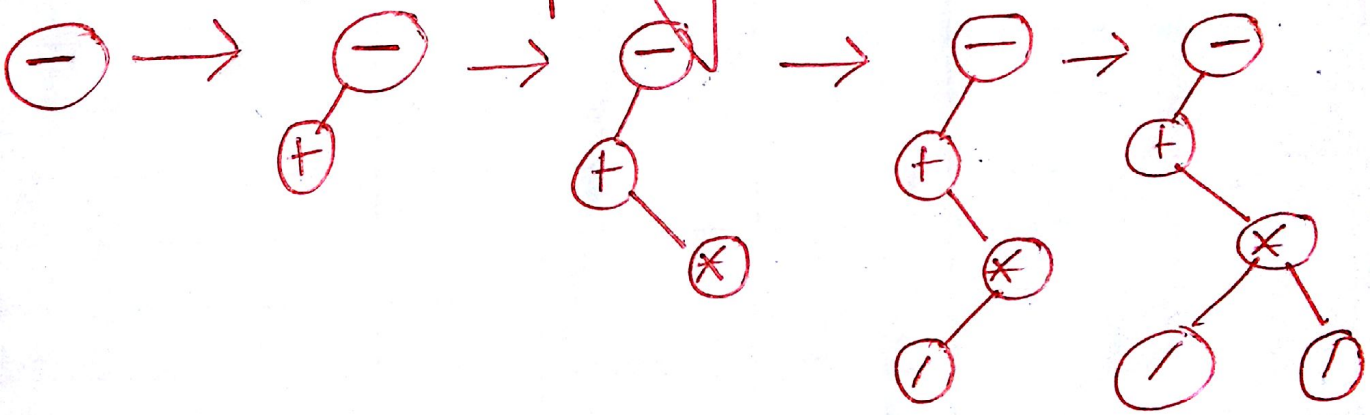
Step 1: Separate operators and operands

+ (/) * (/) - / A B C D E F

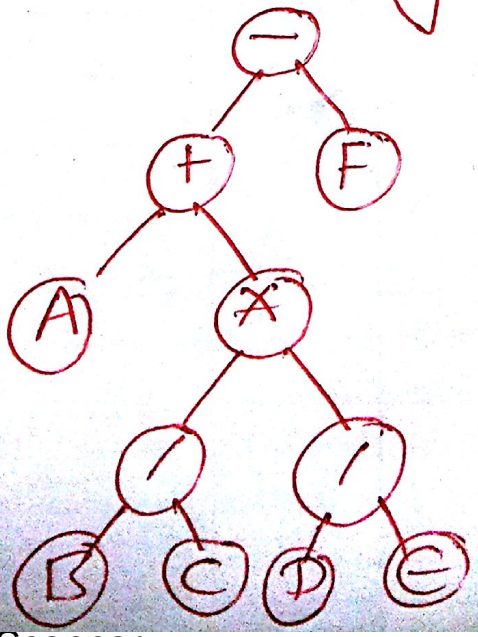
Step 2: Set precedence of operator.

/ , / , * , + , - (higher → lower)

Step 3: Create expression tree with lowest priority operator as root



Step 4: Add operand so as to satisfy strict binary tree.



#

(6)

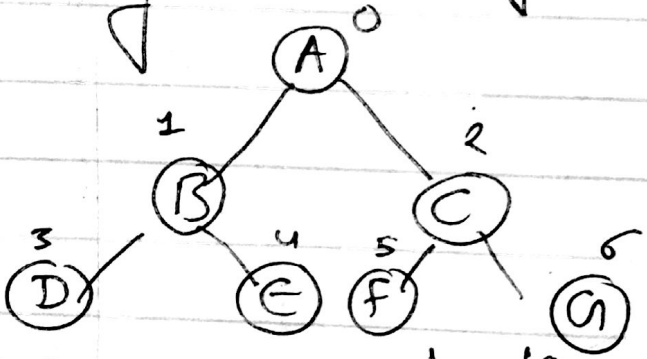
Representation of Binary tree.

→ Binary tree can be implemented using array as well as linked list.

(1) Array representation of Binary tree

— In this method, nodes of the binary tree are represented as elements in an array. The root node always lies at index 0.

eg:



The array representation of preceding binary tree is

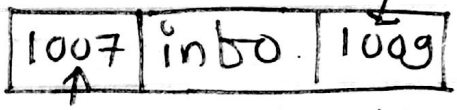
A	B	C	D	E	F	G
0	1	2	3	4	5	6

(2) Linked Representation of Binary tree

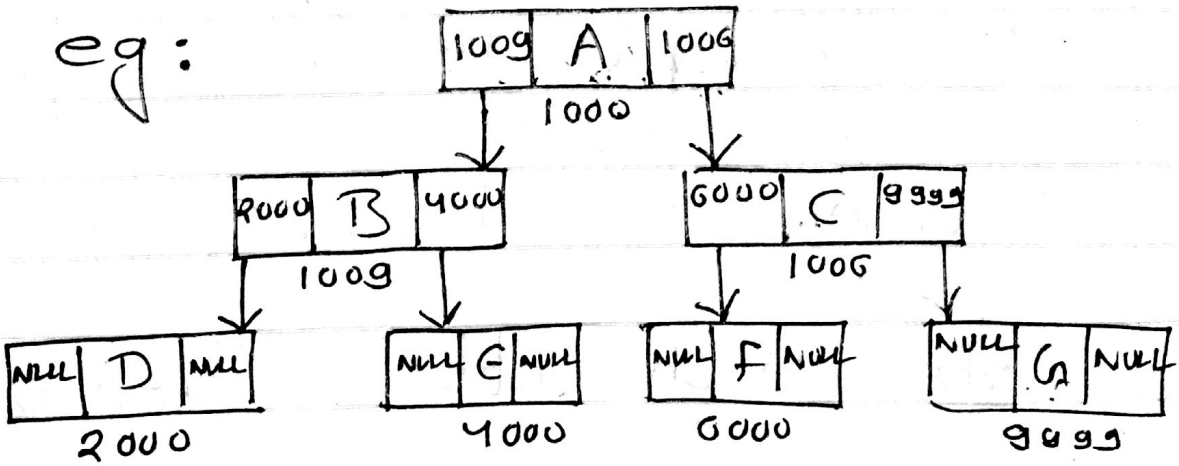
— A binary tree contain one root node and some terminal and non terminal node. The non termind

node have their left child and right child node. So we need Lchild and Rchild pointer to hold references to the left and right child respectively. But in case of terminal, the Lchild and Rchild pointer are set to NULL because terminal node have no children.

So the structure of a Node in a tree is given below

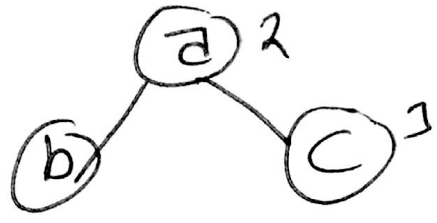


eg:



(3)

imp



Traversal

→ Binary tree traversal is a way in which each node in the tree is visited exactly once. The following are the possible orders in which a binary tree can be traversed

① LDR ② LRD ③ DLR ④ RDL
⑤ RLD ⑥ DRL

where L stands for traversing left
R stands for traversing right subtree
D stands for processing data

LDR order is also called as inorder.
LRD order is also called as postorder
DLR order is also called as preorder.
other are not used.

① Inorder traversal (LDR order)

Algorithm

- Traverse the left sub tree of root.(L)
- Process the root (D)
- Traverse the right subtree of root(R)

(9)

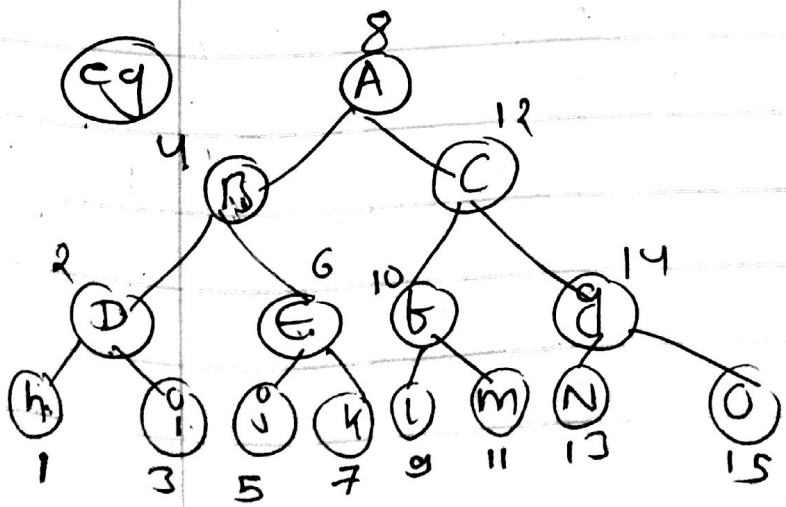
C- Implementation:

A node is defined as

```
struct node
{
    int info;
    struct node *left;
    struct node *right;
};
```

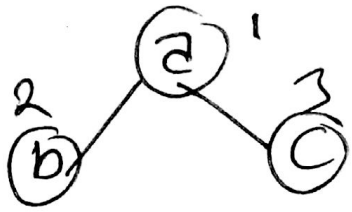
Now, the method for inorder

```
void inorder (struct node *tree)
{
    if (tree != NULL)
    {
        inorder(tree->left);
        printf("%d", tree->info);
        inorder(ptr->right);
    }
}
```



So the Node are visited as below,

H, D, I, B, J, E, K, A, L, F, M, C, N, G and O



(P) Preorder traversal (DLR)

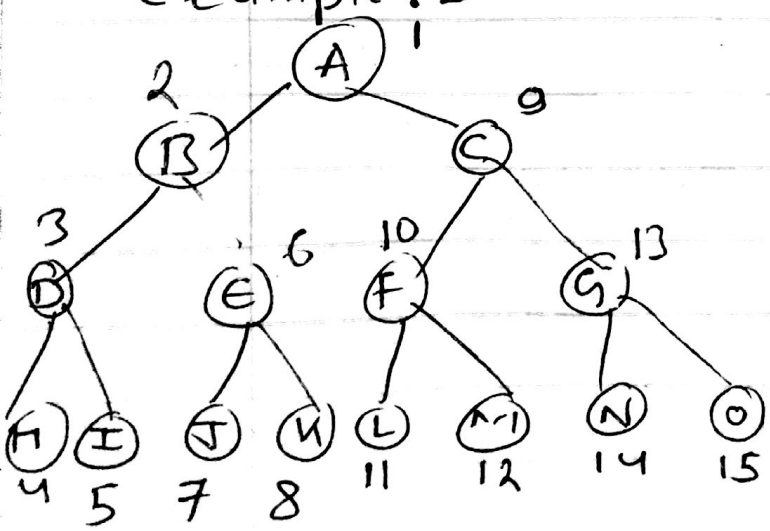
Algorithm

- Visit the root. (D).
- Traverse the left subtree of root. (L)
- Traverse the right subtree of root (R)

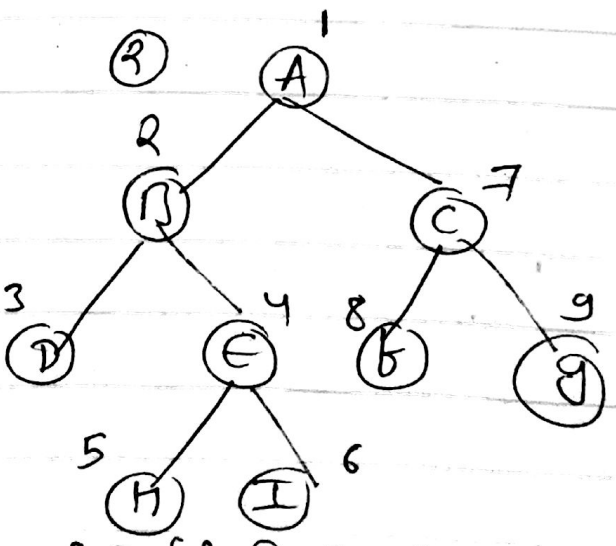
C-implementation

```
Void preorder (struct node * tree)
{
    if (tree != NULL)
    {
    printf ("%d", tree->info);
     preorder (tree->left);
     preorder (tree->right);
     }
}
```

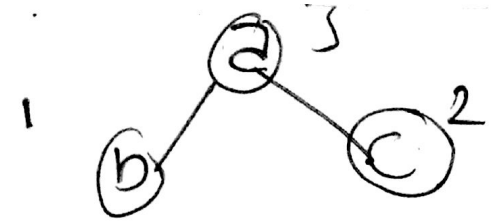
example: 1



I-B { A, B, D, H, I, E, J, K, C, F, L, M, G, N, O }



I-e { A, B, D, E, H, I, C, F, G }



(90)

(3) Post order traversal (LRD)

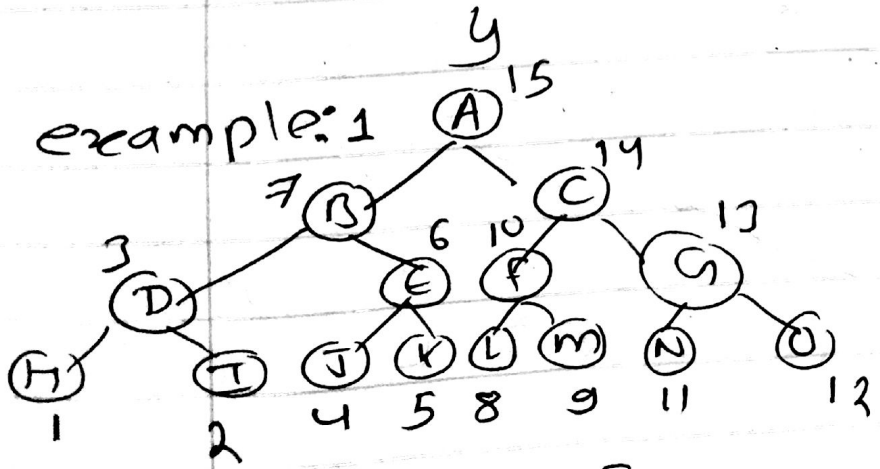
Algorithm

- Traverse the left subtree of root. (L)
- Traverse the right subtree of root. (R)
- Visit the root. (D)

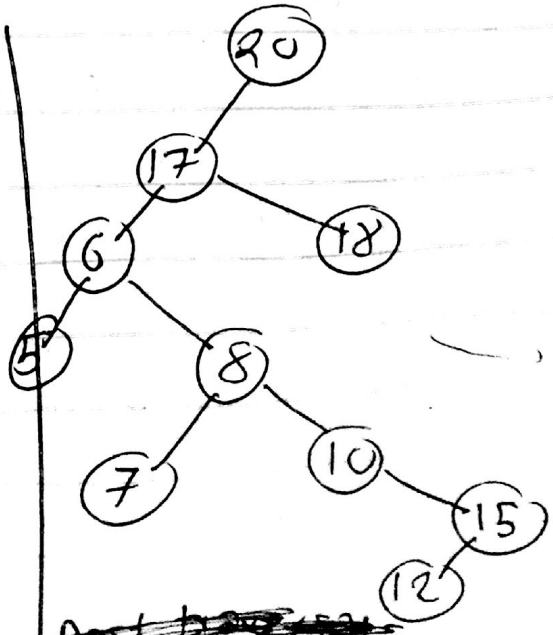
C- implementation

```
void postorder (struct node* tree)
{
    if (tree != NULL)
    {
    postorder (tree->left);
    postorder (tree->right);
    printf ("%d", tree->info);
    }
```

example: 1



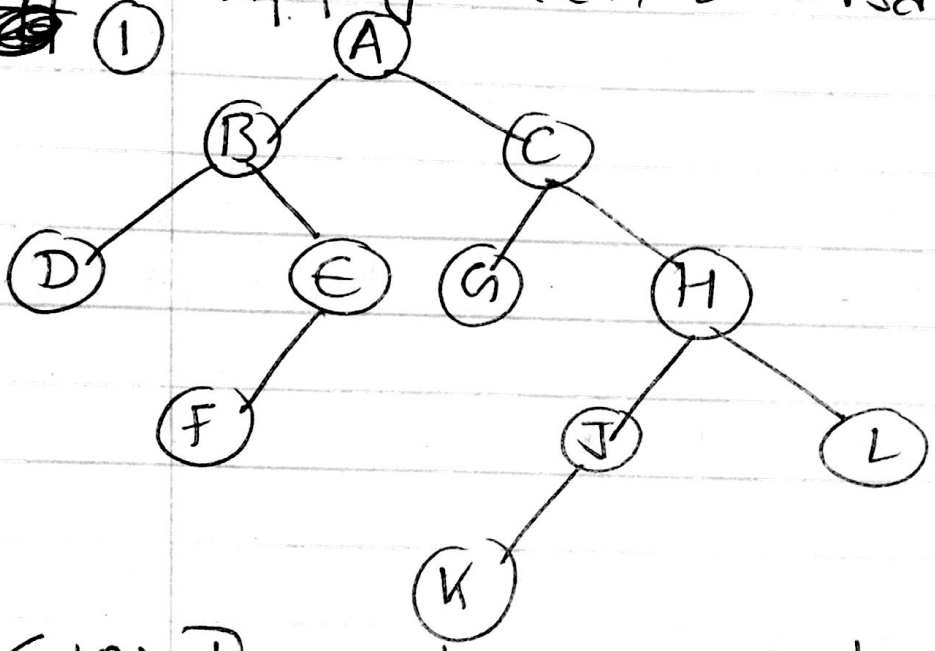
i.e { H, I, D, J, K, E, B, L, M, F, N, O, G, C, A }



The elements order in post order traversal is

{ 5, 7, 12, 15, 10, 8, 6, 18, 17, 20 }

Apply each traversal technique.



Soln: The nodes are visited in order as

{ D, B, F, E, A, G, C, K, J, H, L }

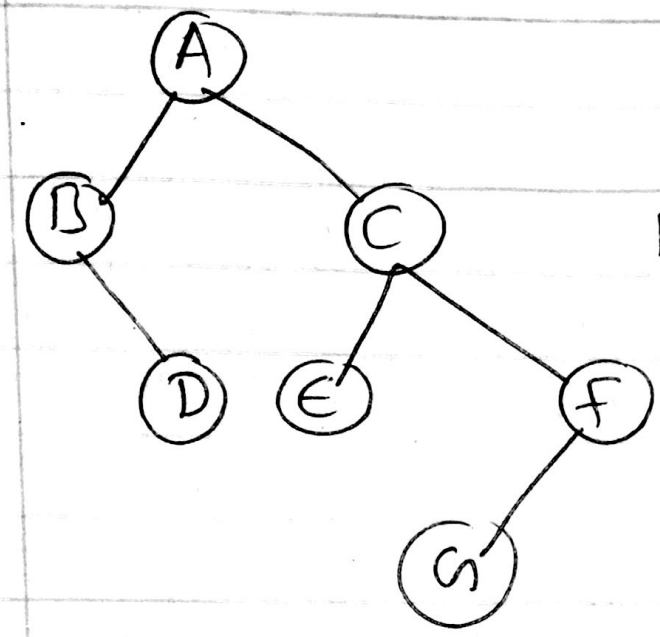
The nodes are visited in preorder as

{ A, B, D, E, F, C, G, H, J, K, L }

The nodes are visited in postorder as

{ D, F, E, B, G, K, J, L, H, C, A }

(9)



Inorder = { B, D, A, E, C, G, F }.

Preorder = { A, B, D, C, E, F, G }

Postorder = { D, B, E, G, F, C, A }

Binary Tree ADT

The binary tree T either empty or consists of a node called root of T together with at most two trees, called left subtree and right subtree of T, together with the following operations:

1. parent(n, T): Return parent node of the node n in tree T.
2. LeftChild(n, T): Return the left child node of the node n in tree T.
3. RightChild(n, T): Return the right child of node of node n is tree T.
4. Root(T): Return the node that is root of tree T.
5. MakeEmpty(T): Create an empty tree T.

Sibling(n, T), info(n, T) etc — .

Binary Search Tree (BST)

– A binary search tree is a binary tree that is either empty or each node contain a valve (key) and satisfy the following condition

i) All keys in the left subtree of the root are smaller than the key in the root.

ii) All keys in the right subtree of the root are greater than the keys in the root.

iii) The left and right subtree of the roots are again BST.

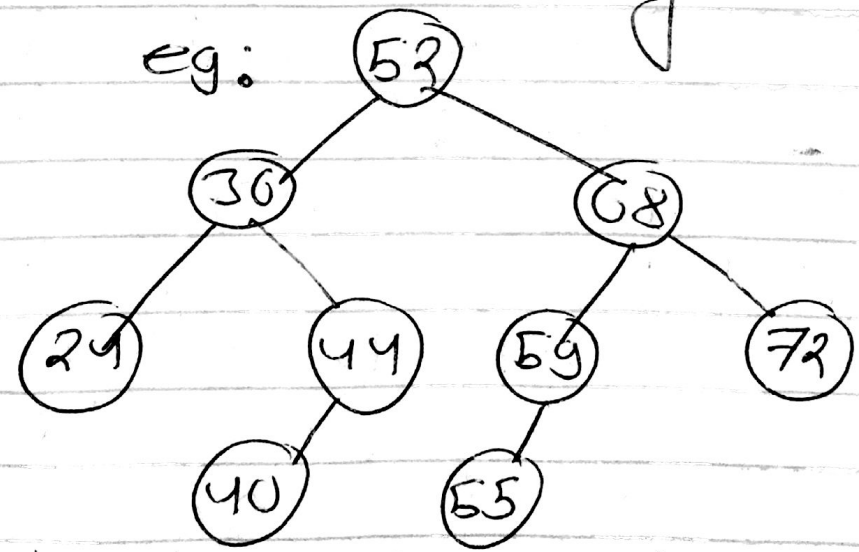


fig: Binary Search tree.

Advantage: It provide fast searching with efficient insertion and deletion as in linked list.

Insertion in BST

Algorithm

(1) - If the tree is empty, create a new node n storing the key x and make the new node n the root of this tree.
- Otherwise, compare x to the key stored in the Root R.

(2) - If x is identical to key in the root R, then stop i.e. do not change tree.

(3) - If x is smaller then the key in the root, insert the new item into left subtree of root

(4) If x is greater then the key in the root R, insert the new item into right subtree of root.

(5) Repeat step 2 to 4 until the leaf node is encountered where the data has to be inserted

example: Construct BST for the following numbers.

40 25 70 22 35 60 80
90 10 30