Types of Binary Tree

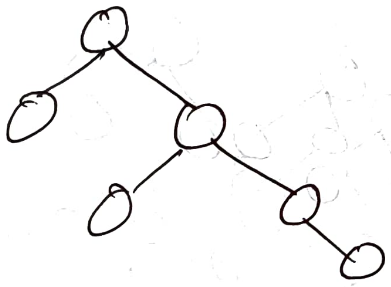
A tree is a binary tree if and only if:

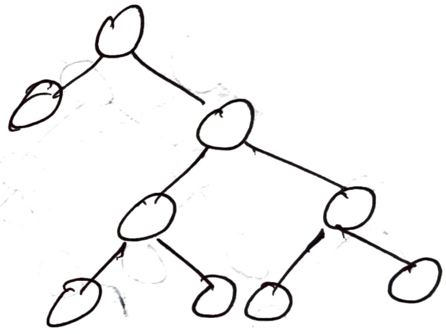
(1) It has a root node, which may not have any child node. (0 child nodes, NULL tree).

(2) A root node may have one or ~~more~~ two child nodes. Each node forms a binary tree itself.

(3) The number of child nodes can not be more than two.

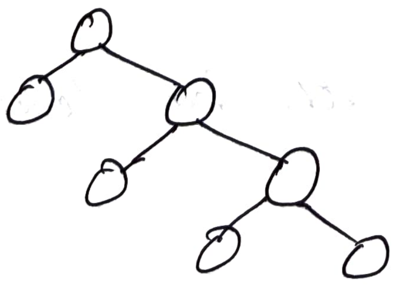
(4) It has unique path from root to every other node.

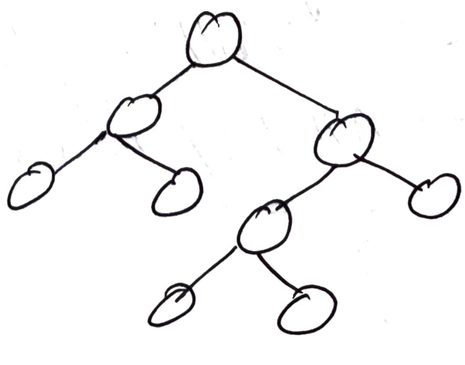




1) Strictly binary tree

A binary tree is a strictly binary tree if and only if each node has exactly two child nodes or no nodes.

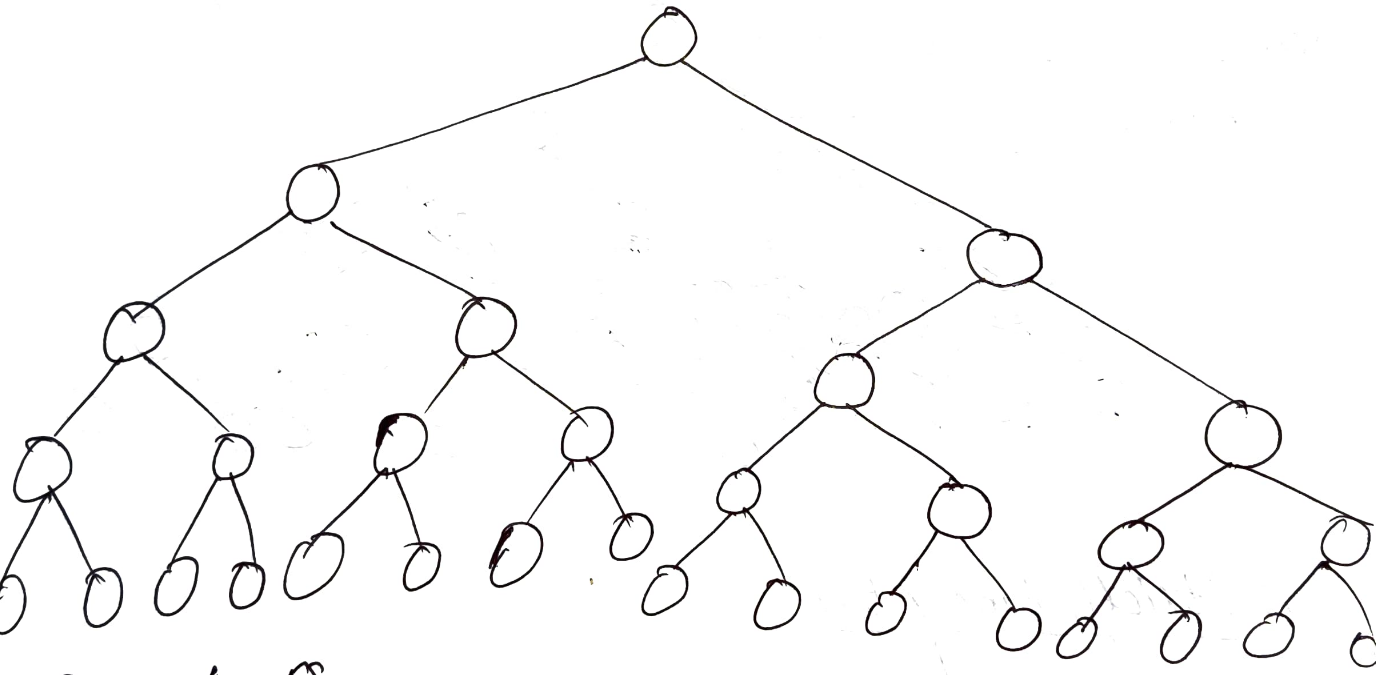




(2) Full binary tree

A binary tree of height h that contains exactly $2^h - 1$ elements is called a full binary tree.

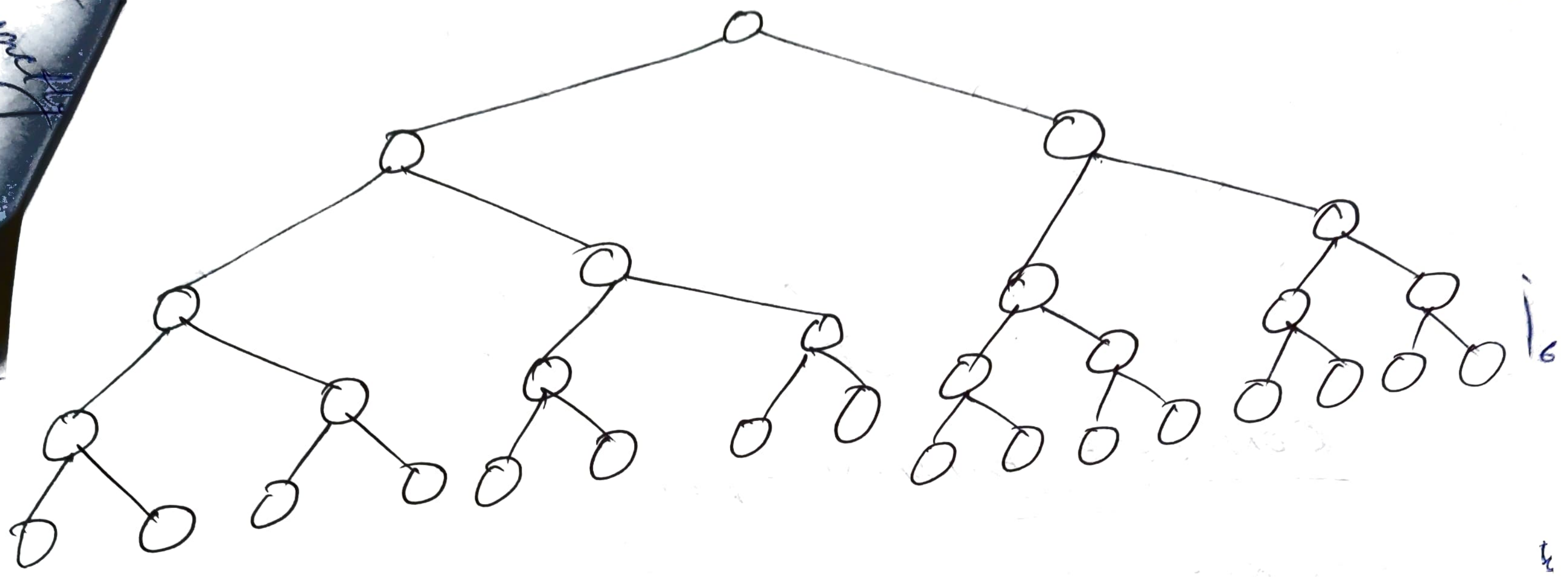
A binary tree is a full binary tree if and only if:
(1) Each non-leaf node has exactly two child nodes.
(2) All leaf nodes are at the same level.



(3) perfect binary tree

A binary tree is a perfect binary tree if and only if
- is a full binary tree.
- All leaf nodes are at the same level.

Eg



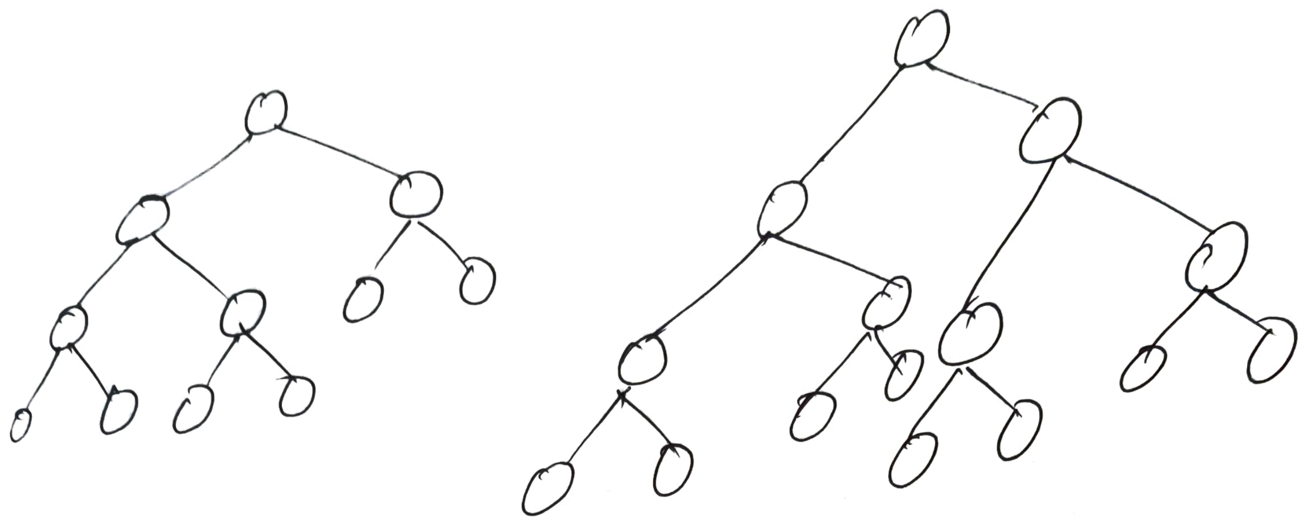
(4) <u>complete binary tree</u>

A binary tree is a complete binary tree of height h (we take root node as 0) if and only if:

→ level 0 to h-1 represent full binary tree of heigh h-1

→ one or more nodes in level h-1, then it may have 0 or 1 child nodes.

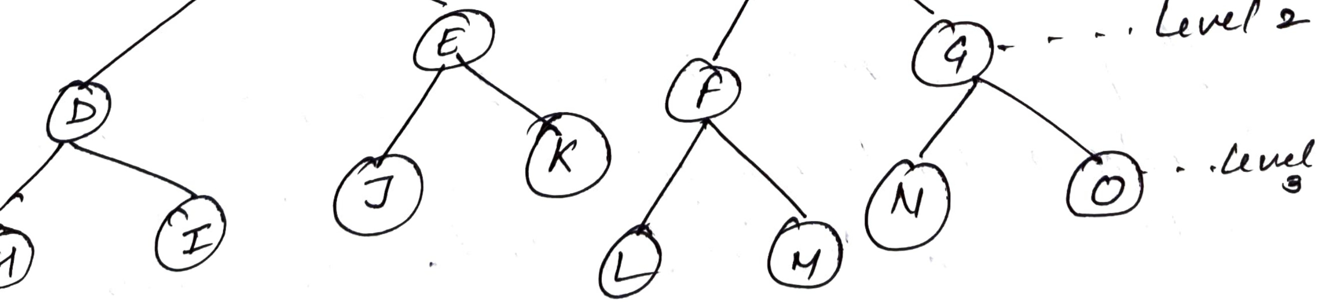
→ if j, k are nodes in level h-1, then j has mor child nodes than k if and only if j is to the left of k ie the last level (h) can be missing leaf node, however the ones presented must be shifted to the left.



COMPLETE BINARY TREE

As we know the depth of a binary tree is the maximum level of any leaf in the tree. This equals the length of the longest path from root to any leaf.

A complete binary tree of depth d is strictly binary tree all of whose leaves are at level d.



If a binary tree contains m nodes at level I, it contains at most 2m nodes at level I+1.

Since a binary tree can contain atmost one node (root) complete binary tree of depth d is the binary tree of depth d that contains exactly 2^I nodes at each level I between 0 and d.

Thus, the total number of nodes in a complete binary tree of depth d equals the sum of the number of nodes at each level between 0 and d ie.

Number of nodes at level 0 is $2^0 = 1$

Number of nodes at level 1 is $2^1 = 2$

...

Number of nodes at level d is 2^d.

The total number of nodes in complete binary tree is

$$2^0 + 2^1 + 2^2 + \ldots + 2^d = \sum_{j=0}^{d} 2^j$$

By induction, it can be shown that this sum equals $2^{d+1} - 1$. Since all leaves in such a tree are at level d, the tree contains 2^d leaves and therefore $2^d - 1$ non-leaf nodes.

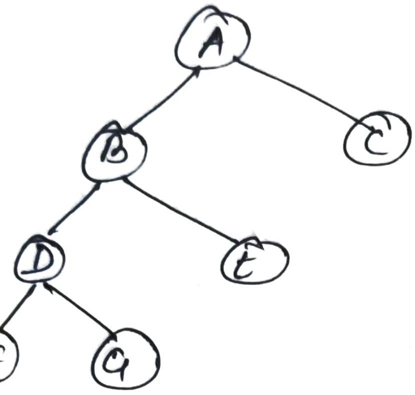
The significance of a complete binary tree is that it is the binary tree with the maximum number of nodes for a given depth.

Almost Complete Binary tree

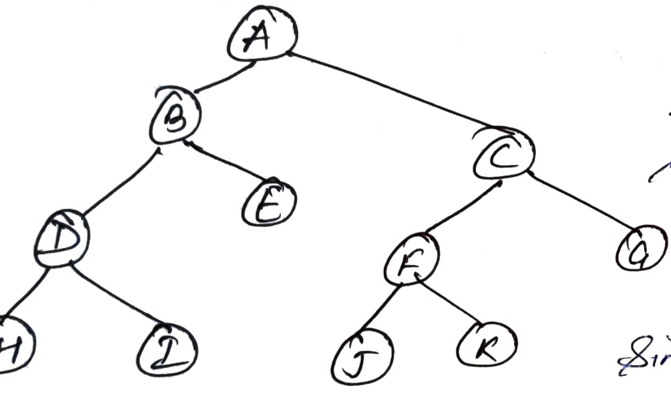
A binary tree of depth d is an almost complete binary tree, if

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

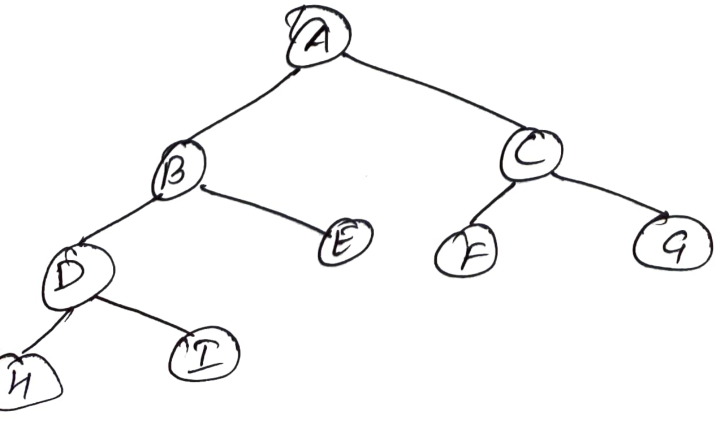
(2) for any node 'x' in the tree with a right descendent at level d, x must have ~~has~~ a left child and every left descendent of x is either a leaf at level d ~~or~~ has two childrens.



It is not almost complete binary tree because it contains leaf nodes at level 1, 2, 3. So violating condition 1.



It satisfies condition 1, since every leaf is either at level 2 or at level 3. However, condition 2 is violated. Since A has a right descendent at level 3 (J) but also has a left descendent that is leaf at level 2 (E);



It satisfies both condition 1 and 2 and therefore is almost complete binary tree.

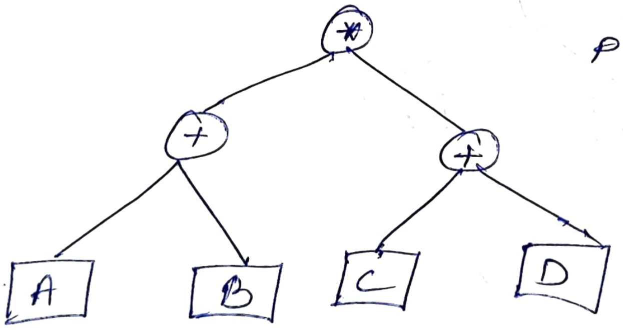
almost complete strictly binary tree with n leaves has 2n-1 nodes and an almost complete binary tree with n leaves that is not strictly binary has 2n nodes

Converting Algebraic Expression into Binary tree

The arithmetic expressions represented as binary trees are known as expression tree. In this root node is operator and the left and right children are operands. In case of unary operator the left child is not present and the right child is operand.

(A+B)*(C+D)

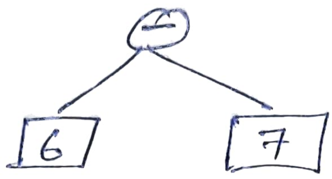
preorder : * +AB +CD
postorder : AB+CD+*



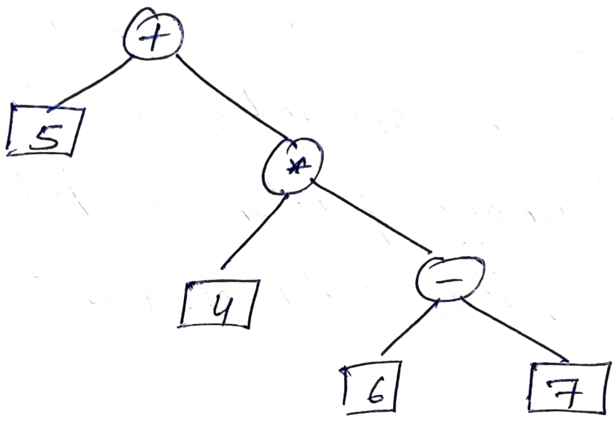
Node appearing in circular shape denote the operators and nodes appearing in square shape denote operands.

Q $(5+4*(6-7))/(5+8)$

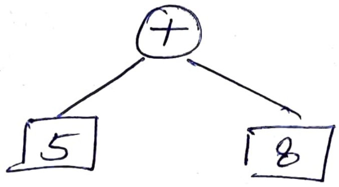
(1)



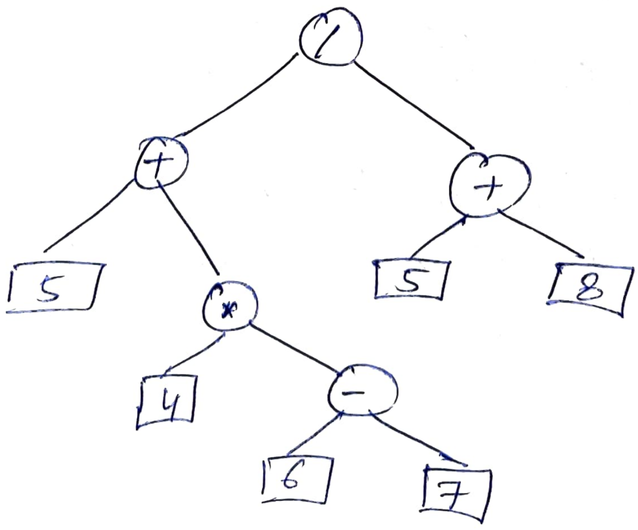
(2) $(5+4*(6-7))$



(3)



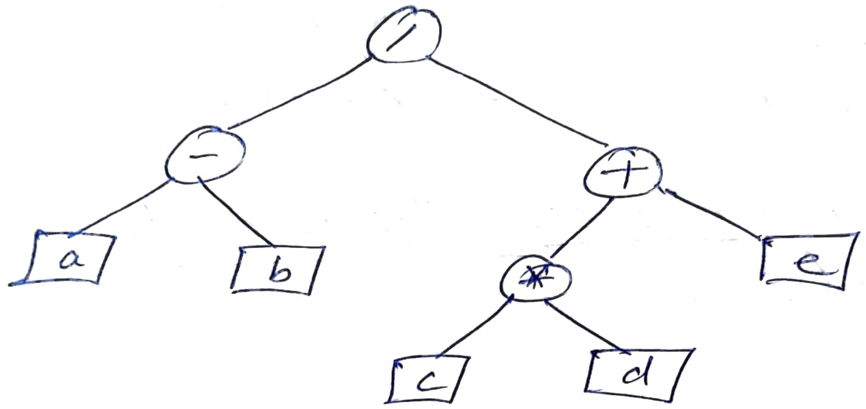
(4)



①

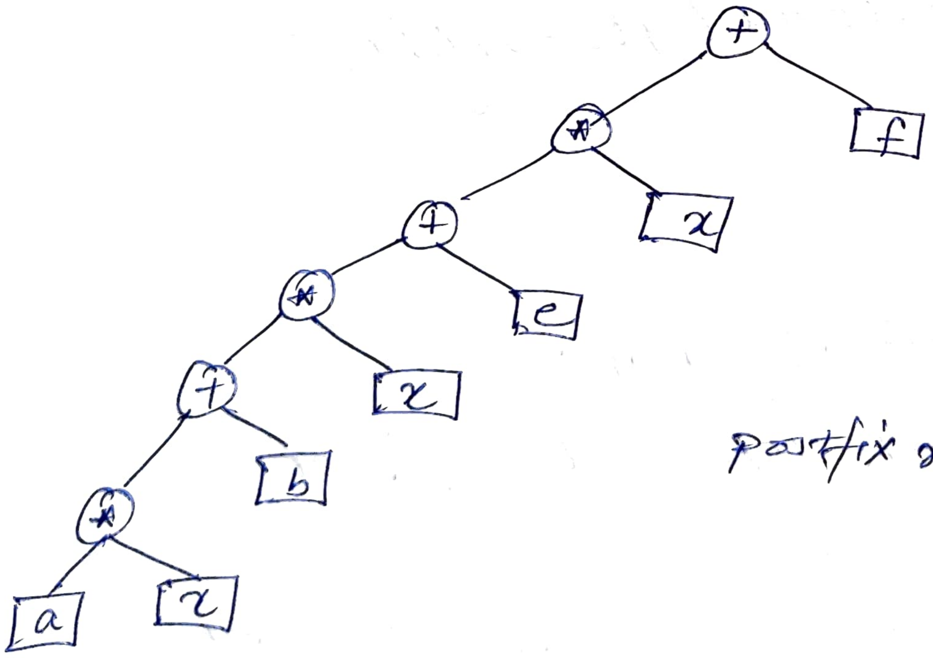
Construct a binary tree for the algebraic expression

$E = (a-b) / ((c*d)+e)$



8 $E = ((a*x+b)*x+e)*x+f$

Construct the expression tree. Find the equivalent postfix notation.



postfix notation: $ax*b+x*e+x*f+$

Traversing in Binary tree

(1) Preorder (NLR traversal)

a) Visit ~~node~~ the root

b) Traverse the left subtree of root

c) Traverse the right subtree of root

The function for preorder traversal is

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

(2) Inorder Traversal (LNR)

a) Traverse the left subtree of root.

b) Visit the root

c) Traverse the right subtree of root.

he function for inorder traversal is :

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

(3) postorder (LRN traversal)

(a) Traverse the left subtree of root.

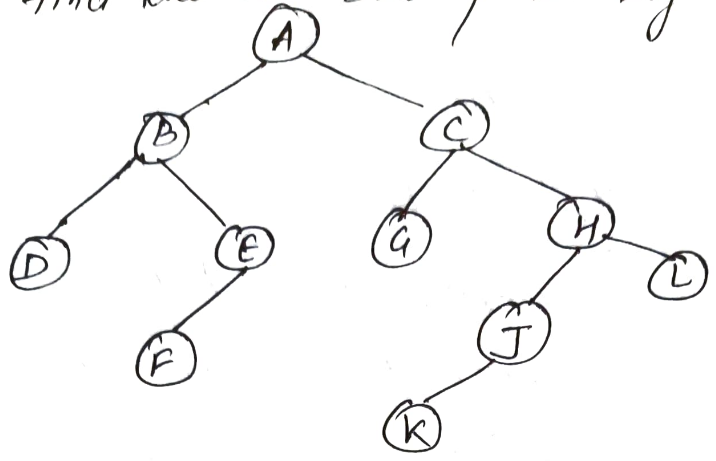
(b) Traverse the right subtree of root

(c) Visit the root.

The function for postorder traversal is :

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

Q find the traversal of Binary tree:



Inorder traversal (LNR)

DBFEAGC KJHL

Preorder traversal (NLR)

ABDEFCGHJKL

Postorder traversal (LRN)

DFEBGKJLHCA

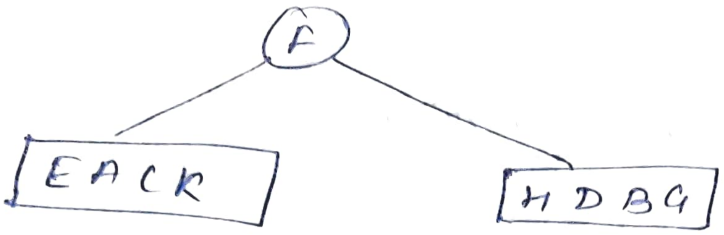
Q Draw a binary tree:

Preorder : F A E K C D H G B
Inorder : E A C K F H D B G

Sol^n Root node is $\textcircled{F}$ (preorder first node)

Inorder $\underbrace{[EA\ CK]}$ $\textcircled{F}$ $\underbrace{H\ D\ B\ G}$

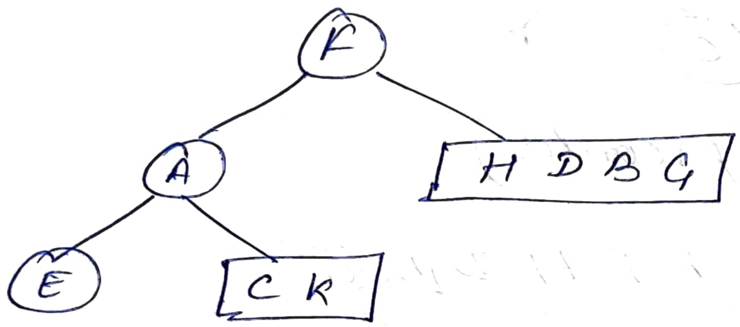
ie



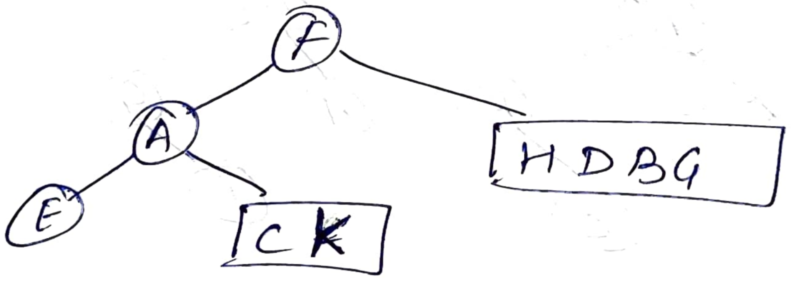
Next node is A (through preorder)

Inorder

$\underbrace{E}$ Ⓐ $\underbrace{C\ K}$ F H D B G

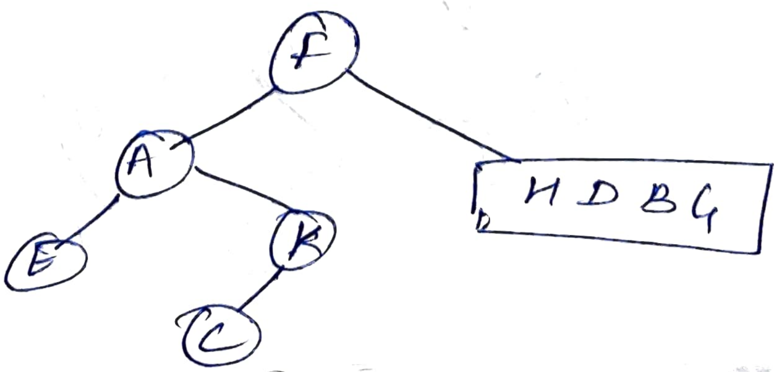


Now, next node is E (preorder)



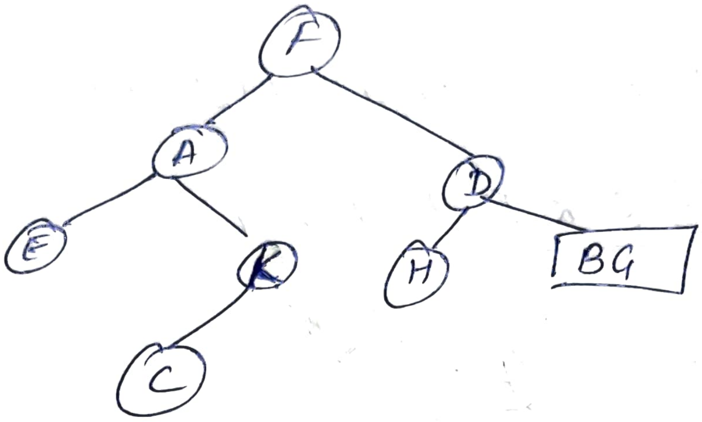
Now, Next node is K (preorder)

E A $\underbrace{C}$ Ⓚ F H D B G



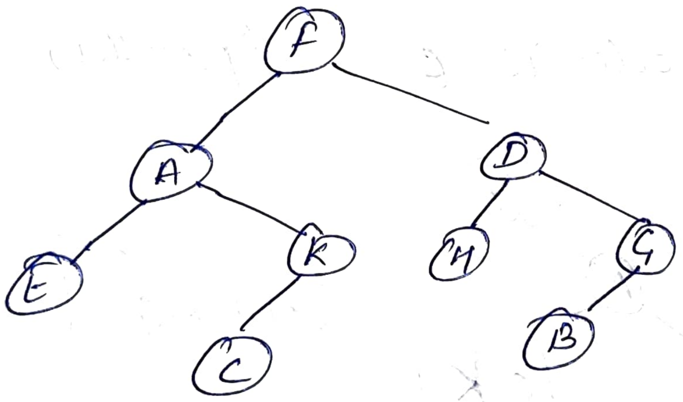
Now D, (preorder)

Inorder E A C K F $\underline{H}$ D $\underline{B\,G}$

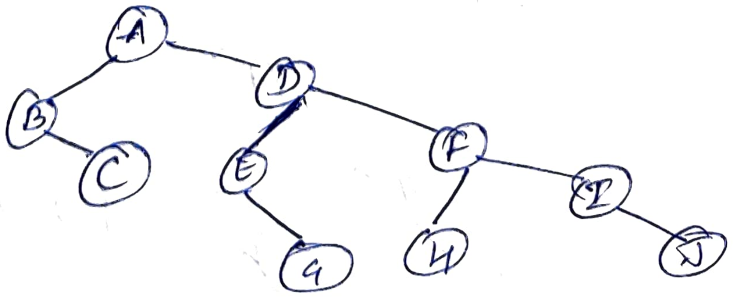


Now G,
Next node (preorder)

Inorder E A C K F H D $\underline{B}$ G



Q Create a binary tree
Inorder : B C A E G D H F I J
preorder : A B C D E G F H I J


A
B
D
C
E
F
I
G
H
J