TREES

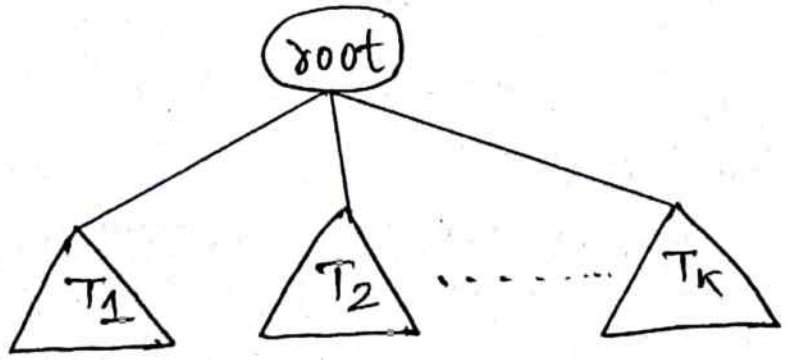
- Trees are non-linear data structure.
- Tree is a hierarchical data structure.
- Tree is a data structure which allows us to associate a parent-child relationship between various piece of data and thus allows us to arrange our records, data and files in a hierarchical fashion.

Definition of Tree

A tree (T) is a collection of elements called "nodes" such that the nodes have a parent-child relationship that satisfy the following:

(1) If T is not empty, then T has a special node called <u>root</u>. The root has no parent.

(2) The root can have zero or more non-empty subtrees $T_1, T_2, \ldots T_K$ each of whose roots are connected by a directed edge from root of T.



Scanned by CamScanner

Basic Terminology

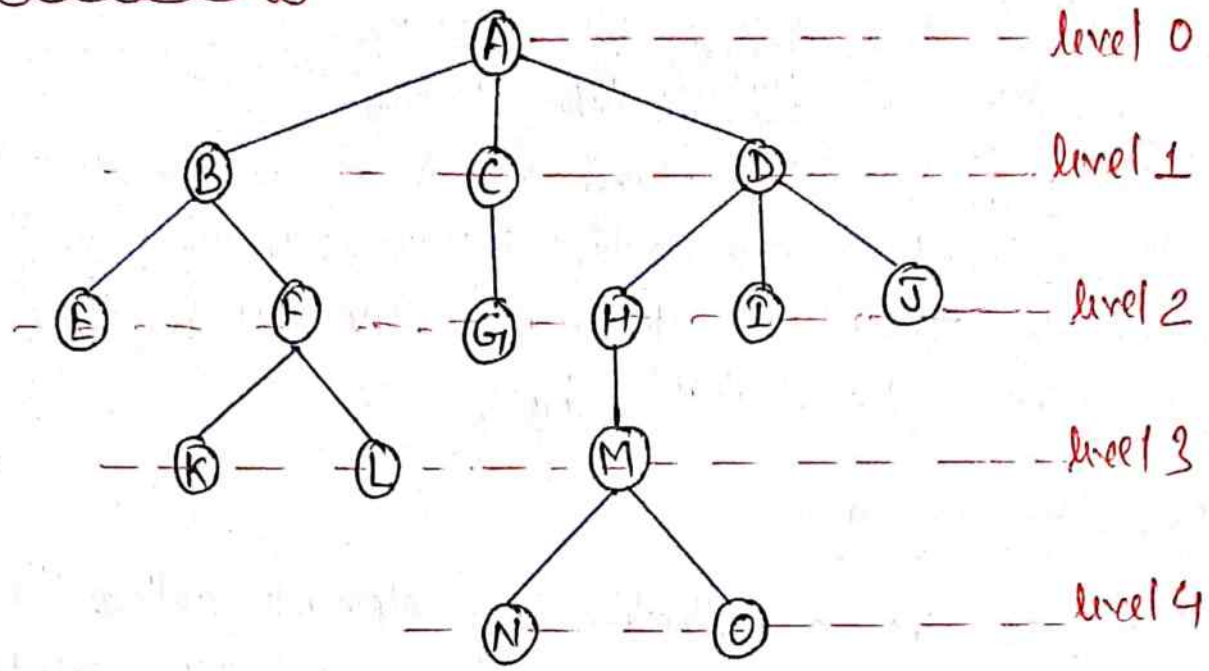


figure. Tree

Node :-> Each element of tree is called a node. A node is represented by circle.

Edge :-> The line connecting the nodes are called edges.

Parent Node :-> The immediate predecessor of a node is called parent node.

For example. Node A is parent of Node B, C, D.

Child Node :-> All the immediate successors of a node are called its child node.

For example. Node B, C, D are the children of node A.

Sibling Node :-> Nodes with same parent are called sibling Nodes.

For example: Node H, I, J are sibling Nodes.

Scanned by CamScanner

Ancestor Node :-> Ancestor of a node is its parent, grand parent etc.

For example: Ancestors of Node H are Node D, A.

Descendant Node :-> Descendant of a node is its child, grand child etc.

Internal Node :-> A node with atleast one child.

For example :-> Internal nodes are A, B, C, D, F, H, M

External Node :-> (Leaf Node) A node having no children.

For example :-> External nodes are E, K, L, G, N, O, I, J

Path :-> It is the number of successive edges from source node to destination node.

For example: The path length from node D to N is 3. (D-H, H-M, M-N).

Level :-> Level of any node is defined as the distance of that node from the root.

For example: Root node is at a distance 0 from itself. So it is at level 0.

The immediate children of the root are at level 1 and so on.

Root node A is at level 0

Nodes B, C, D are at level 1

Nodes E, F, G, H, I, J are at level 2

Nodes K, L, M are at level 3

Nodes N, O are at level 4

Scanned by CamScanner

Degree :-> The degree of a node is the number of children of that node.

Note: Nodes with degree 0 are called leaf nodes.

For example:

Degree of Node A is 3
Degree of Node M is 2
Degree of Node C is 1
Degree of Node E is 0. (∵ E is leaf node).

Height of tree :->

Height of tree = Height of root

Height of a node :-> It is the number of edges in longest path from the node to a leaf node.

For example :-> Height of Root node A is 4.

Height of node B is 2.
Height of node M is 1.
Height of node N is 0.

Note: Height of tree with one node = 0.

Depth of a node :-> It is the number of edges in path from root to that node.

For example :-> Depth of Root node A is 0.

Depth of node B is 1.
Depth of node M is 3.
Depth of node N is 4. (Maximum)

Note :-> Maximum Depth value is the Height of Tree.

Scanned by CamScanner

Binary Tree

A tree is called Binary Tree if each node of the tree can have maximum of two children. One child is called the "left" child and other is called the "right" child.

i.e. In a binary tree, a node can have degree 0, 1 or 2.

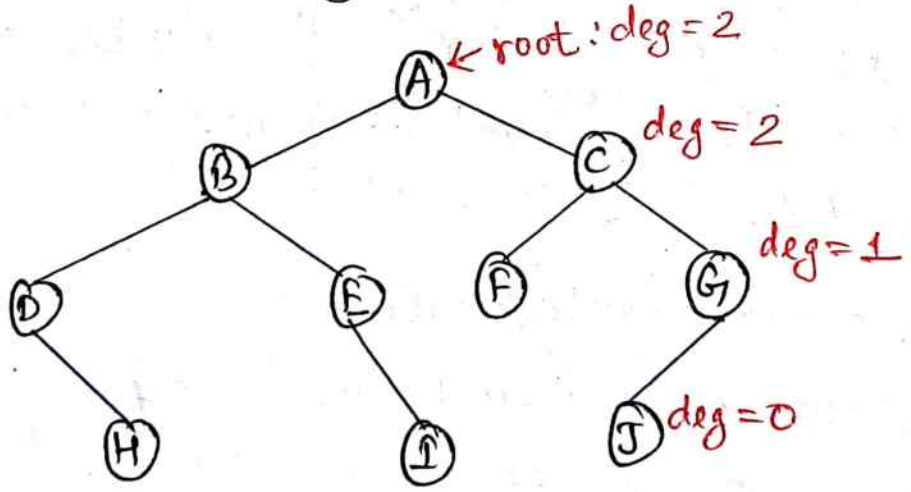


figure. A Binary Tree

Scanned by CamScanner

Q1. Prove that in a binary tree, the number of leaves is equal to the number of nodes with two children + 1.

Proof:-> Let n_0 be number of leaf nodes.

n_1 be number of nodes with degree 1.

n_2 be number of nodes with degree 2.

$\therefore$ n = total number of nodes

i.e $n = n_0 + n_1 + n_2 \longrightarrow (1)$

We know that, a binary tree with n nodes has exactly $n-1$ edges.

These edges may emerge from the node that has degree one or the nodes that has degree two.

i.e $n-1 = 2n_2 + n_1$

$\Rightarrow n = 2n_2 + n_1 + 1 \longrightarrow (2)$

From equ (1) and equ (2)

$n_0 + n_1 + n_2 = 2n_2 + n_1 + 1$

$\boxed{n_0 = n_2 + 1}$

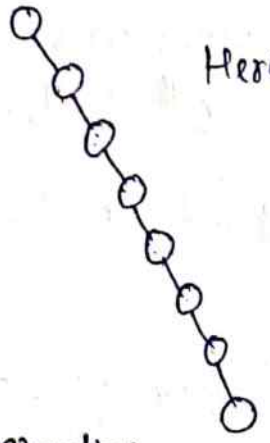
$\therefore$ No. of leaves is equal to no. of nodes with degree 2 + 1.

Scanned by CamScanner

Q2. If a binary tree contains n nodes, then its maximum height possible is n and minimum height possible is $\lceil \log_2 (n+1) - 1 \rceil$.

Proof:-> Let $n = 8$

case(1)



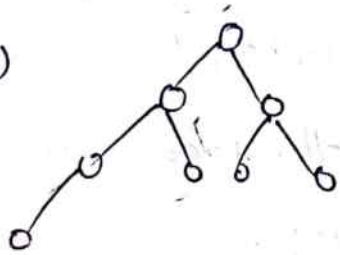
Here, the height of given binary tree is 7

$\therefore$ Maximum height possible is $n-1$

$\therefore$ $O(n)$ is Maximum Height.

fig. Binary tree

case (2)



Here, the height of given binary tree is 3

There should be at least one element at each level, so the height cannot be more than n.

We know that, a binary tree of height h can have maximum $2^{h+1}-1$ nodes.

So, the number of nodes will be less than or equal to this maximum value.

$$\therefore \quad n \leq 2^{h+1}-1$$

$$\Rightarrow 2^{h+1} \geq n+1$$

take log on both sides

$$\log(2^{h+1}) \geq \log(n+1)$$

$$\Rightarrow h+1 \geq \lceil \log_2 (n+1) \rceil \quad (h \text{ is an integer})$$

So, the minimum height possible is $\lceil \log_2 (n+1) - 1 \rceil$.

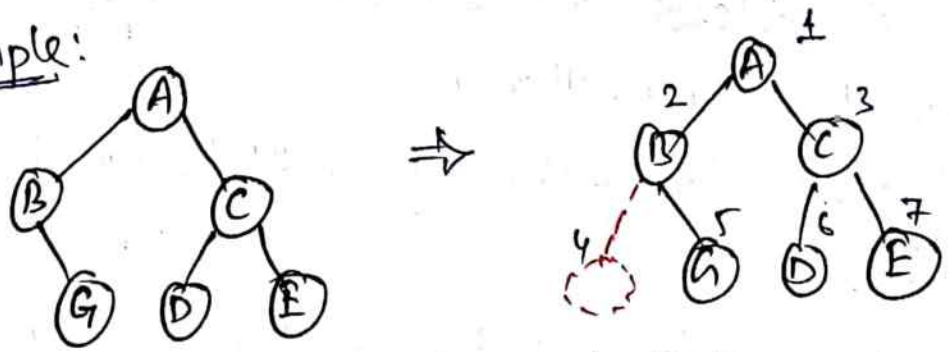
Scanned by CamScanner

Representation of Binary Tree

(1) Array Representation of Binary Tree

- Binary Trees are stored in memory in the form of array. which is called sequential representation.
- In this representation, the nodes of the trees are stored level-by-level starting from the 0th level.
- The root node of the tree is stored in the first memory location.
- In order to represent a tree in memory, the nodes are numbered sequentially level by level from left to right and even empty nodes are numbered.
- The number appearing against the node will work as indices of the node in the array.

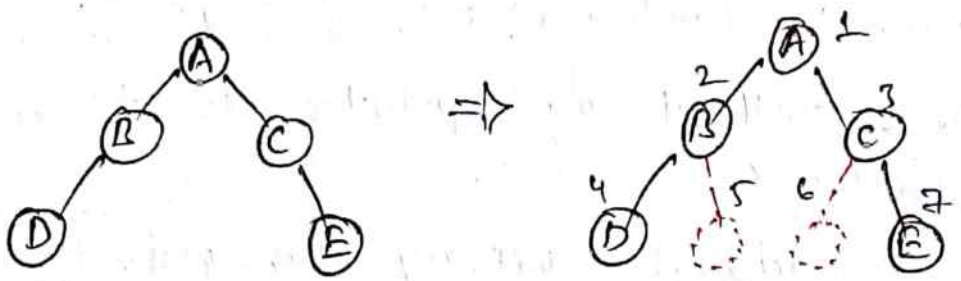
Example:



∴ The array representation of above binary tree is

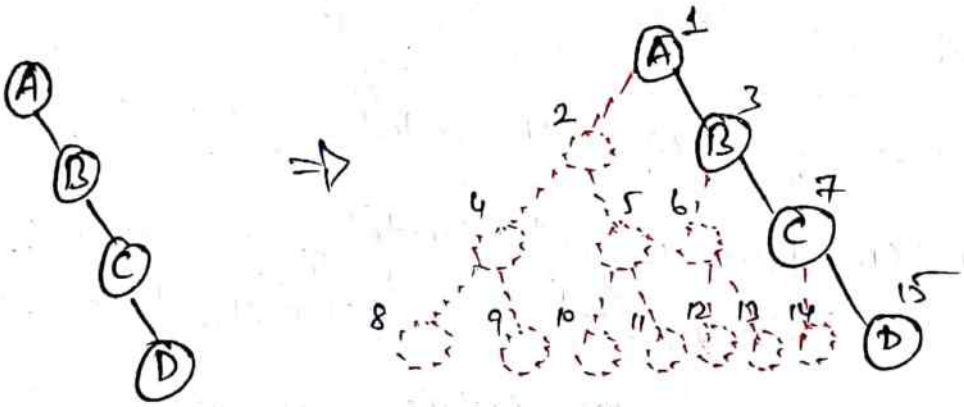
A	B	C		G	D	E
1	2	3	4	5	6	7

Scanned by CamScanner



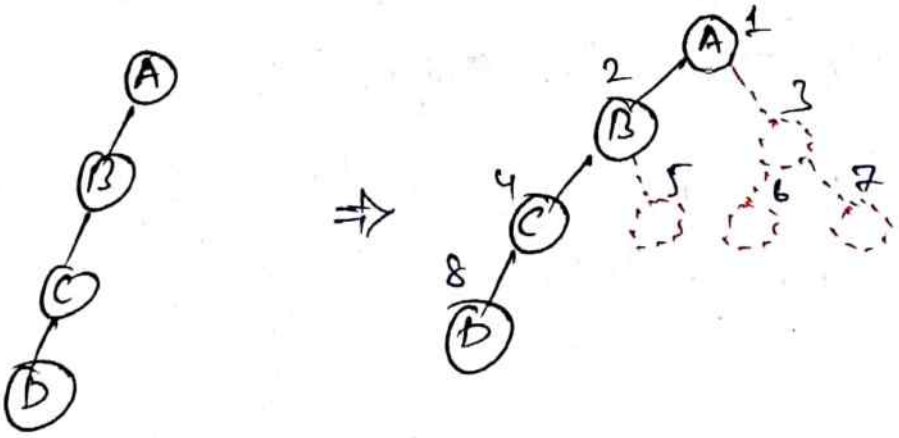
∴ Array representation of above Binary Tree is

A	B	C	D			E
1	2	3	4	5	6	7



∴ Array representation of above Binary Tree is

A		B				C								D
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



∴ Array representation of above Binary Tree is

A	B		C				D
1	2	3	4	5	6	7	8

Scanned by CamScanner

Advantages of Array Representation

(1) Data are stored without any pointer to its successor or predecessor.

(2) Any node can be accessed from any other node by calculating the index.

(3) Programming languages which do not support dynamic memory allocation, use only this type of representation for a tree.

(4) Simplicity

Disadvantages of Array Representation

(1) Array representation is not suitable for normal binary tree but it is only ideal for complete binary tree.

(2) Here most of the array entries are empty i.e. unnecessary more memory is wasted.

(3) There is no way possible to enhance the tree structure.

(4) Additions and deletions of nodes are inefficient because of the data movements in the array.

To overcome, all these problems, we have linked-list representation of binary tree.

Scanned by CamScanner

NOTE :→ The following rules are used to decide the location of any node.

1. The root node is at location 1.
2. For any node with index i, $1 < i \leq n$
   (a) Parent $(i) = \lfloor i/2 \rfloor$
   (b) Left child $(i) = 2i$
   (c) Right child $(i) = 2i + 1$

Linked-List Representation of Binary Trees

In a linked-list representation, a node in a tree contains three fields.

(i) Left Link (Left) :→ It contains address of left subtree.
(ii) Info (Data) :→ It contains the info value.
(iii) Right Link (Right) :→ It contains address of right subtree

left	data	right

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

Note :→ When a node has no children, the corresponding pointer fields are NULL.

Scanned by CamScanner

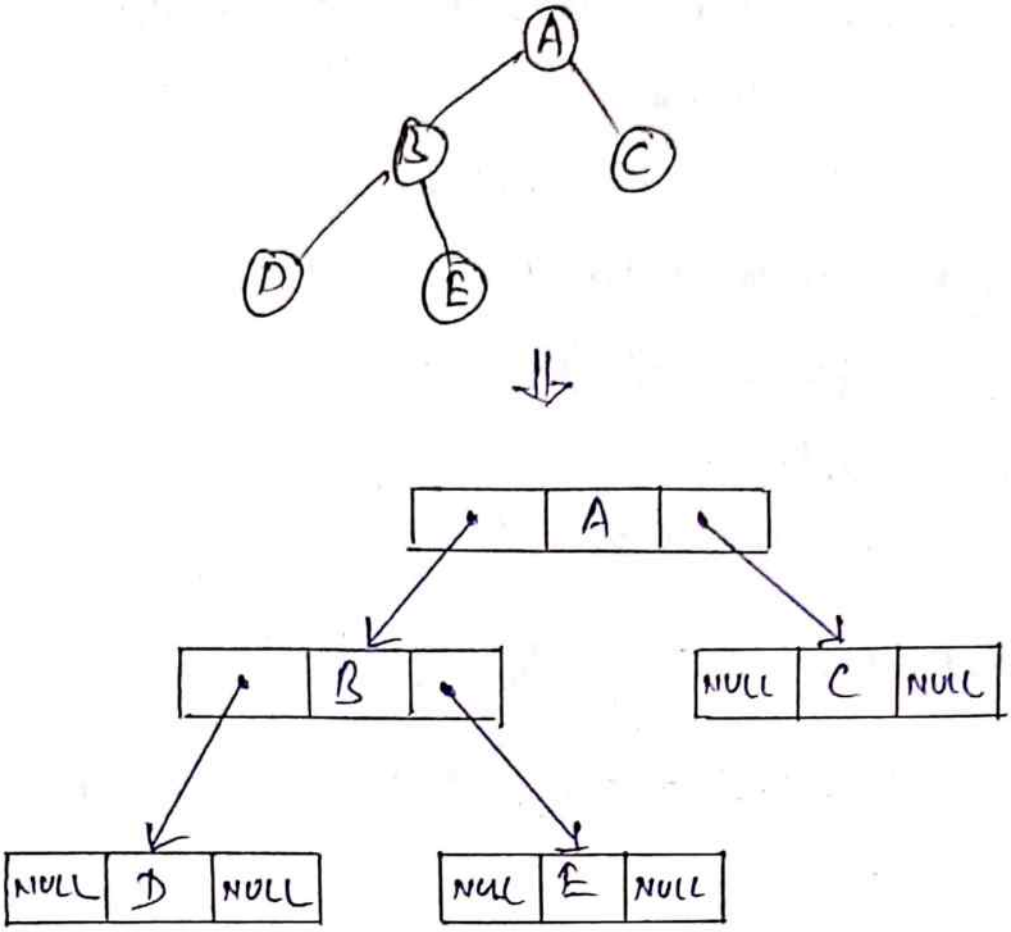


figure. Linked-list representation of Given Binary Tree.

This type of representation is dynamic i.e. block of memory required for the tree need not be allocated before. They are allocated only on demand.

Advantages

(1) Insertion and deletions involve no data movement.
(2) No wastage of memory.
(3) Enhancement of the tree is possible.

Disadvantages

(1) In this, pointer fields are involved which occupy more space than data fields.
(2) Programming language which do not support dynamic memory allocation have difficulty in implementing the binary tree.

Scanned by CamScanner

Binary Tree Traversals

- Traversing a tree is the process of visiting each node of the tree exactly once.
- Three different type of Binary Tree traversals are :->
  (1) Pre-order traversal
  (2) In-order traversal
  (3) Post-order traversal

Pre-Order Traversal (Root - Left - Right)

Steps are :->

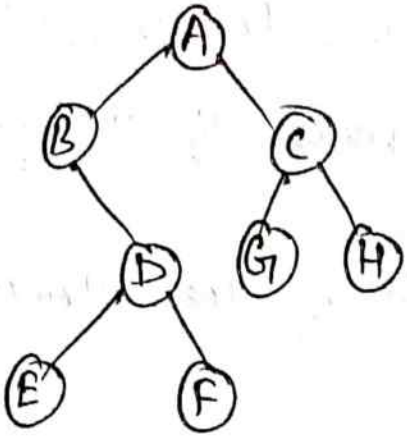
(i) Visit the root
(ii) Traverse the left sub-tree of root in pre-order.
(iii) Traverse the right sub-tree of root in pre-order.

Function is :->

```
void preorder (struct node * root)
{
    if ( root != NULL)
    {
        printf ("%d \n", root -> data);
        preorder ( root -> left);
        preorder (root -> right);
    }
}
```

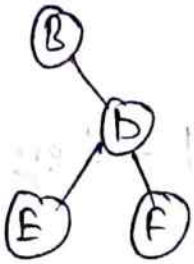
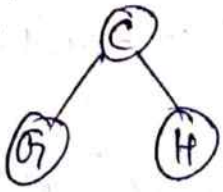
Scanned by CamScanner

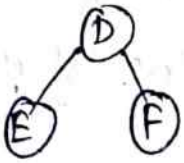
Q1.



Find pre-order traversal.

Soln

Step(1) A + pre-order of  + pre-order of 

Step(2) A + [B + preorder of ] + [C + preorder of (G) + preorder of (H)]

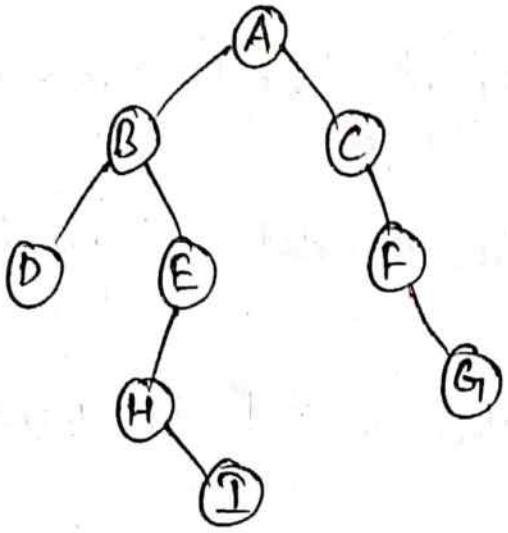
Step(3) A + B + [D + preorder of (E) preorder of (F)] + [C + G + H]

Step(4) A B D E F C G H

∴ pre-order traversal is A B D E F C G H.

Scanned by CamScanner

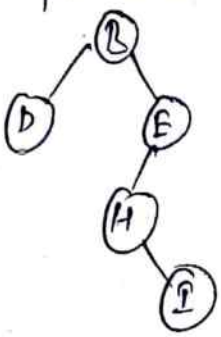
Q2



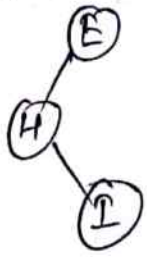
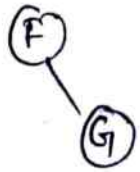
Find Pre-order traversal

Soln

step(1)

A + pre-order of  + pre-order of

step(2)

A + [B + pre-order of + pre-order of ] + [C + pre-order of ]

step(3)

A + B + D + [E + pre-order of] + C + [F + pre-order of]

step(4)

A B D E H I C F G

∴ pre-order traversal is A B D E H I C F G

Scanned by CamScanner

In-Order traversal (Left-Root-Right)

Steps are

(i) Traverse the left subtree of root in inorder.

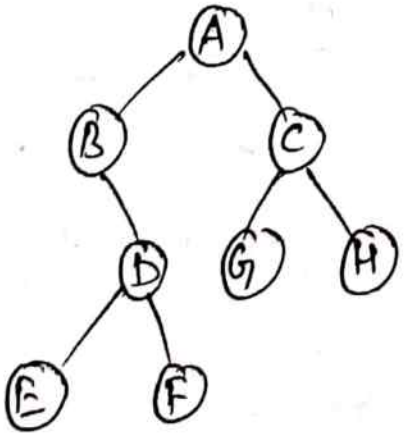
(ii) Visit the root.

(iii) Traverse the right subtree of root in inorder.

Function is

```
void inorder (struct node *root)
{
    if (root != NULL)
    {
        inorder (root -> left);
        printf ("%d \n", root -> data);
        inorder (root -> right);
    }
}
```

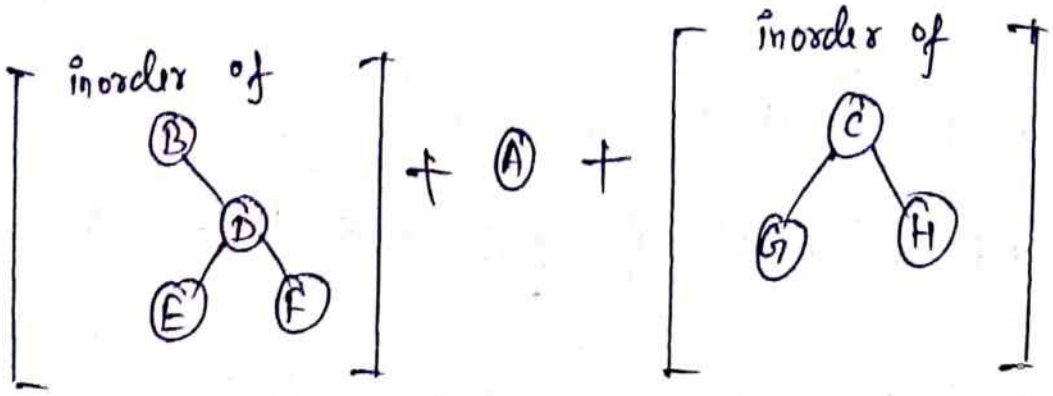
Q1.



Find inorder traversal.

Soln

Step(1)



Scanned by CamScanner

Step (2)

$$\left[B + \text{inorder of } \begin{array}{c} D \\ E \quad F \end{array} \right] + A + \left[\text{inorder of } G + C + \text{inorder of } H \right]$$

Step (3)

$$B + \left[\text{inorder of } E + D + \text{inorder of } F \right] + A + \left[G \; C \; H \right]$$

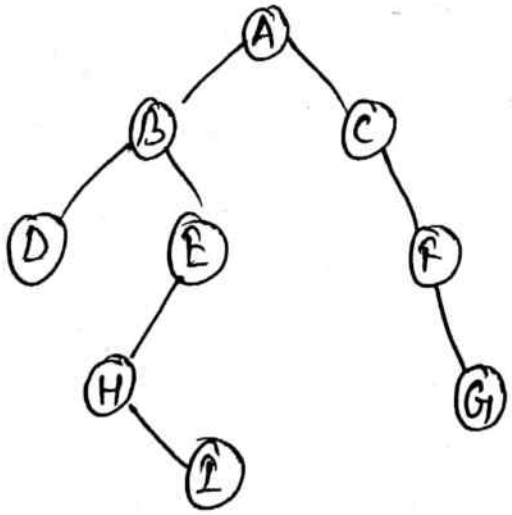
Step (4)

$$B + \left[E \; D \; F \right] + A + \left[G \; C \; H \right]$$

Step (5) B E D F A G C H

∴ inorder traversal is B E D F A G C H

Q2.



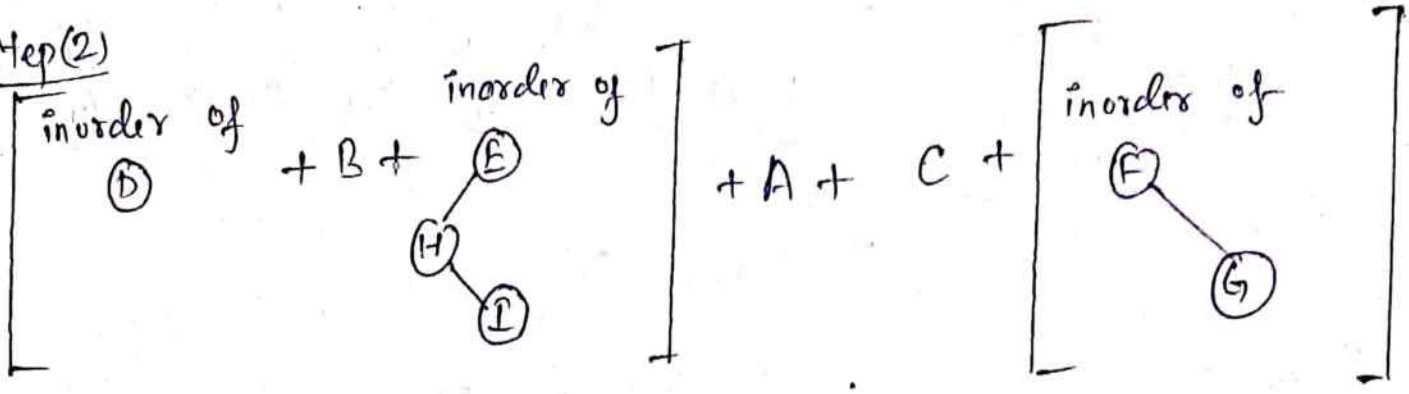
Find inorder traversal.

Soln Step (1)

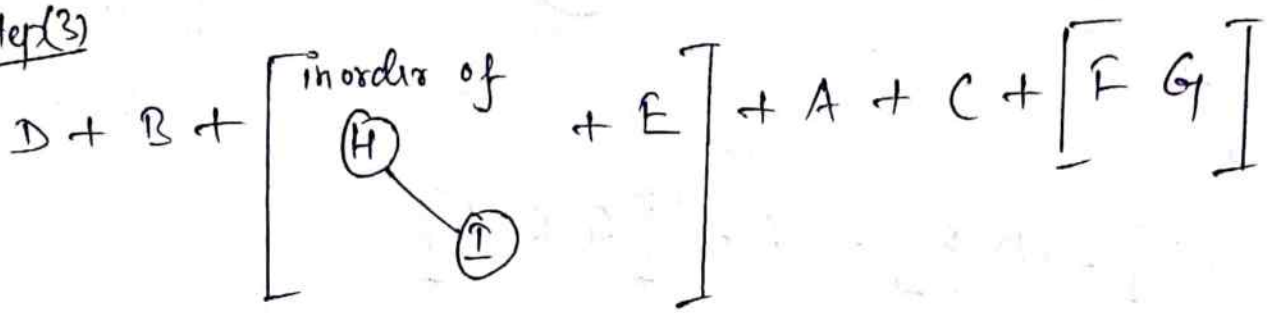
$$\left[\text{inorder of } \begin{array}{c} B \\ D \quad E \\ H \\ I \end{array} \right] + A + \left[\text{inorder of } \begin{array}{c} C \\ F \\ G \end{array} \right]$$

Scanned by CamScanner

Step(2)



Step(3)



Step(4)

D B H I E A C F G

∴ inorder traversal is D B H I E A C F G

Scanned by CamScanner

Post order Traversal (Left - Right - Root)

Steps are :->

(i) Traverse the left subtree of root in postorder.

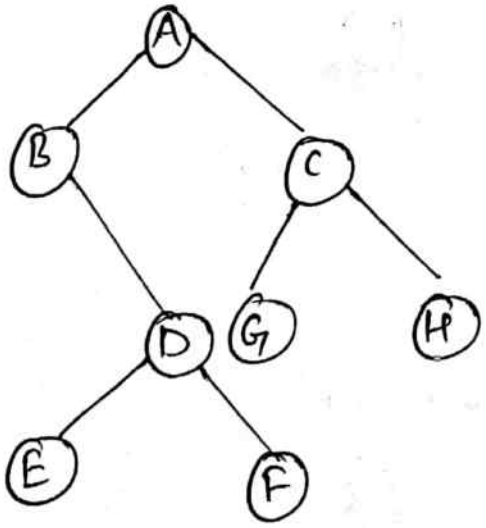
(ii) Traverse the right subtree of root in post-order.

(iii) Visit the root

Function is :->

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

Q1.

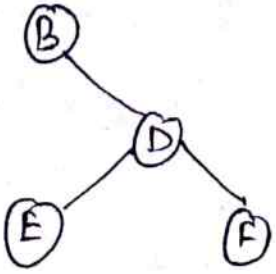


Find post-order traversal.

soln

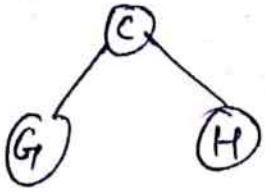
step(1)

post-order of



+

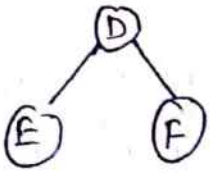
post-order of



+ A

Scanned by CamScanner

Step(2)

[postorder of  + B] + [postorder of (G) + postorder of (H) + C] + A

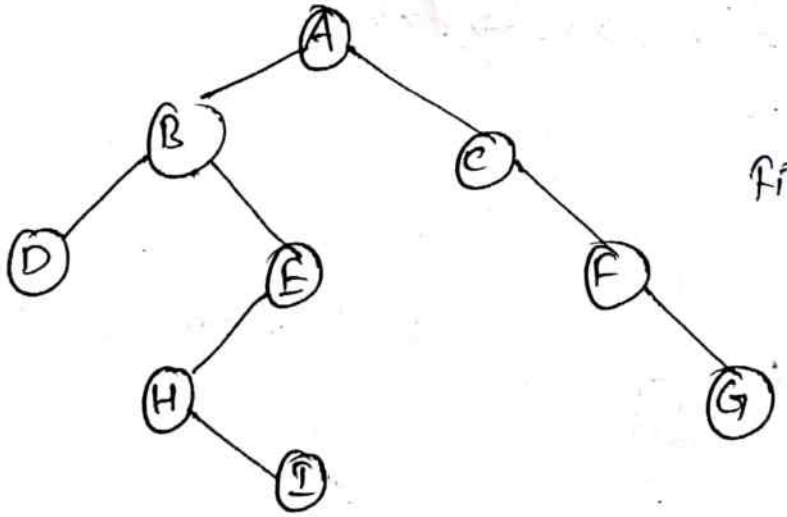
Step(3)

[postorder of (E) + postorder of (F) + D] + B + G + H + C + A

Step(4) E F D B G H C A

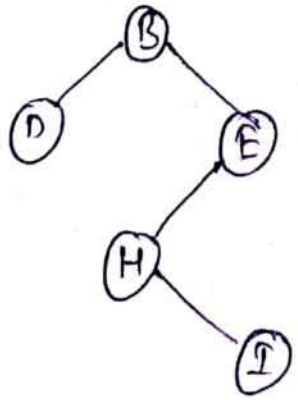
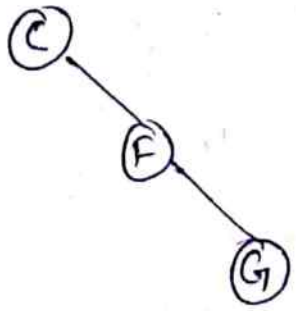
∴ post-order traversal is E F D B G H C A.

Q2



Find post-order traversal.

Soln

Step(1) post-order of  + post-order of  + A

Scanned by CamScanner

Step (2)

[postorder of (D) + postorder of (E, H, I) + B] + [postorder of (F, G) + C] + A

Step (3)

D + [post-order of (H, I) + E] + B + [postorder of (G) + F] + C + A

Step (4)

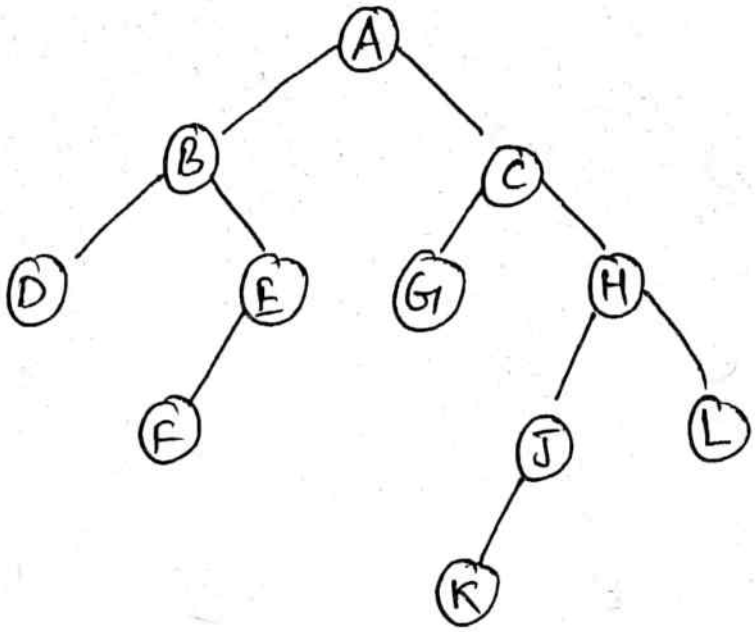
D + [postorder of (I) + H] + E + B + G + F + C + A

Step (5) D I H E B G F C A

∴ postorder traversal is D I H E B G F C A

Scanned by CamScanner

Que

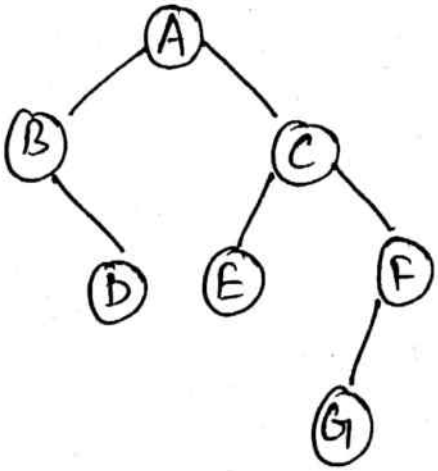


Pre-order : A B D E F C G H J K L

In-order : D B F E A G C K J H L

Post-order : D F E B G K J L H C A

Que



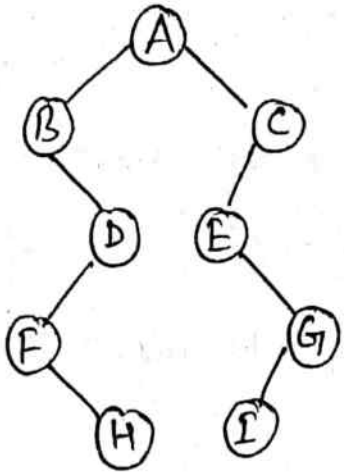
Pre-order : A B D C E F G

In-order : B D A E C G F

Post-order : D B E G F C A

Scanned by CamScanner

Que

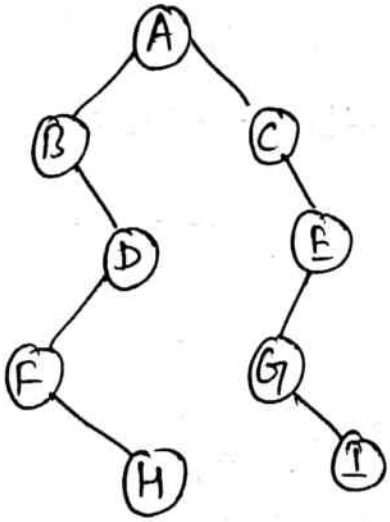


Pre-order : A B D F H C E G I

In-order : B F H D A E I G C

Post-order : H F D B I G E C A

Que

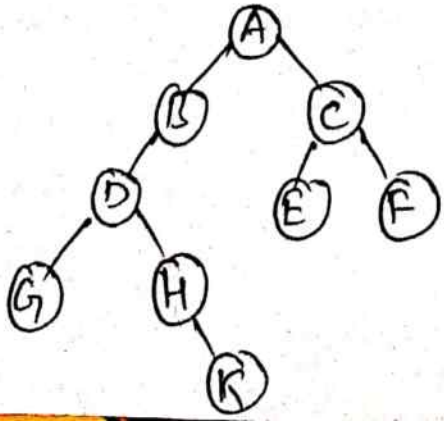


Pre-order : A B D F H C E G I

In-order : B F H D A C G I E

Post-order : H F D B I G E C A

Que



∴ Pre-order : A B D G H K C E F

In-order : G D H K B A E C F

Post-order : G K H D B E F C A

Scanned by CamScanner

Creation of Binary Tree Using Traversal

1. Given pre-order and in-order traversal.
2. Given post-order and in-order traversal.
3. Given pre-order and post-order traversal.

(1) Creation of Binary Tree from Pre-order and In-order

Q1. Construct the binary tree given the following traversals:

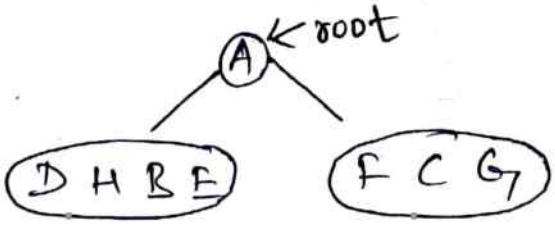
Pre-order : A B D H E C F G

In-order : D H B E A F C G

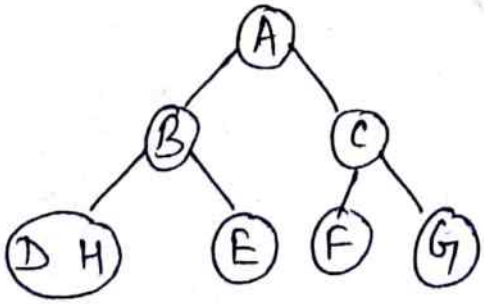
Soln Pre-order is Root-Left-Right

In-order is Left-Root-Right

Step(1)



Step(2)



Step(3)

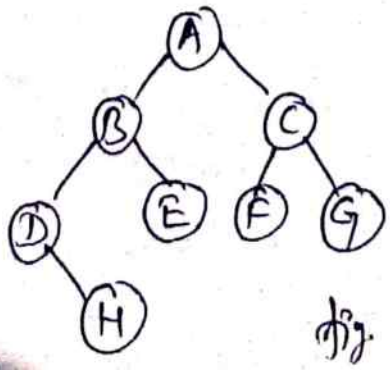


fig. Binary Tree

Scanned by CamScanner

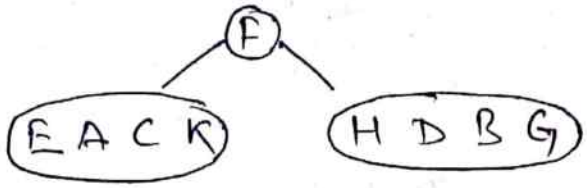
Q Draw the binary Tree:

Pre-order : F A E K C D H G B

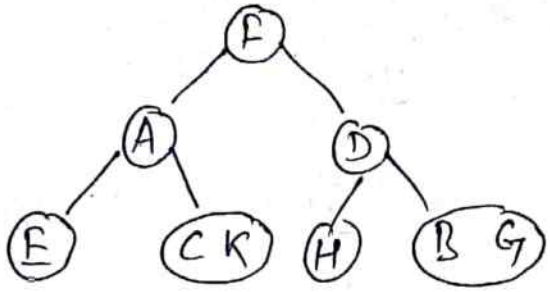
In-order : E A C K F H D B G

Soln

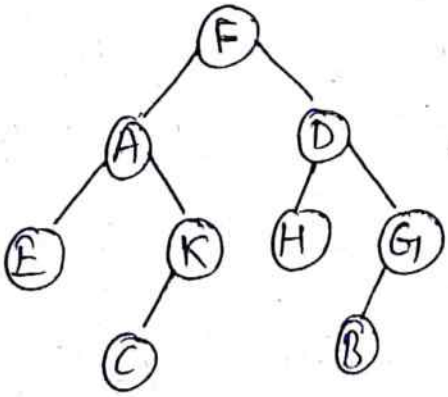
step(1)



step(2)



step(3)

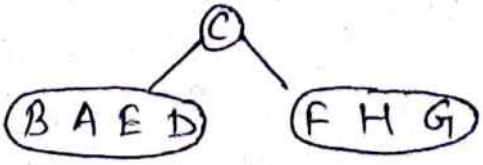


Q. Draw the binary Tree:

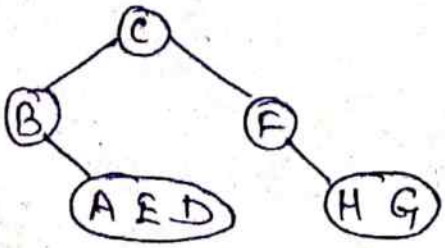
Pre-order : C B D E A F H G

In-order : B A E D C F H G

Soln step(1)

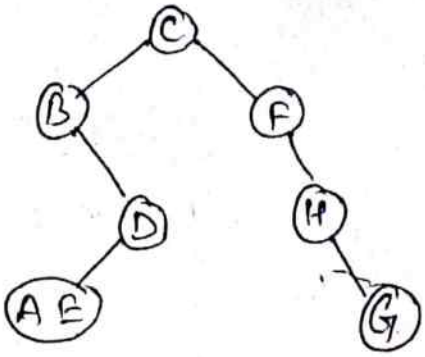


step(2)

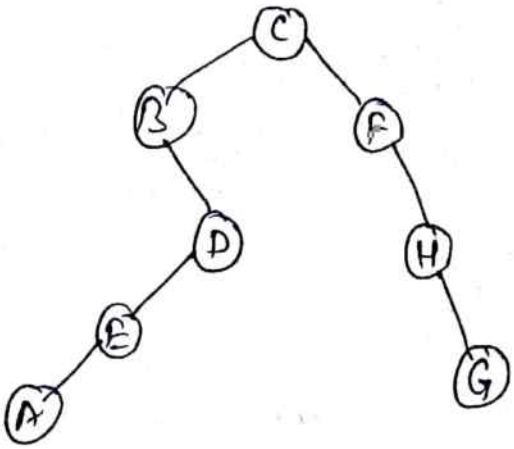


Scanned by CamScanner

Step(3)



Step(4)

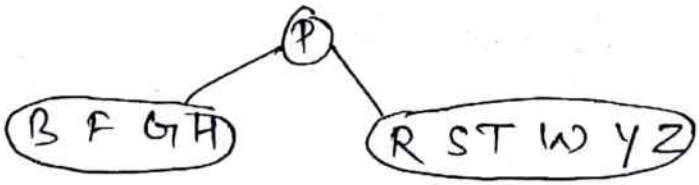


Q Draw the Binary tree :->

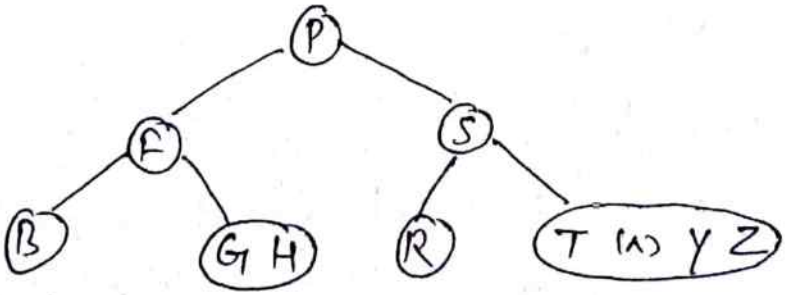
In-order : B F G H P R S T W Y Z

Pre-order : P F B H G S R Y T W Z

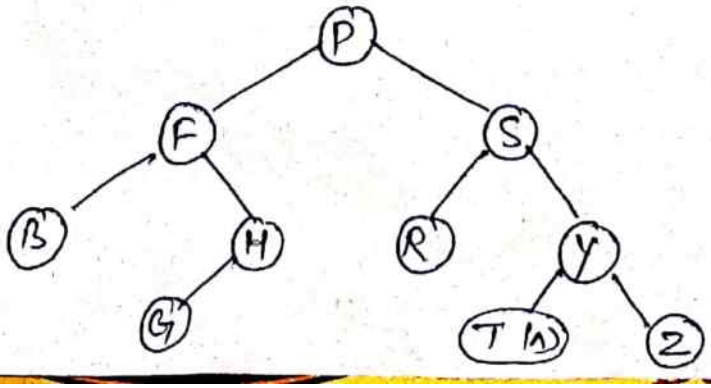
Soln step(1)



Step (2)



Step (3)



Scanned by CamScanner

Step (4)

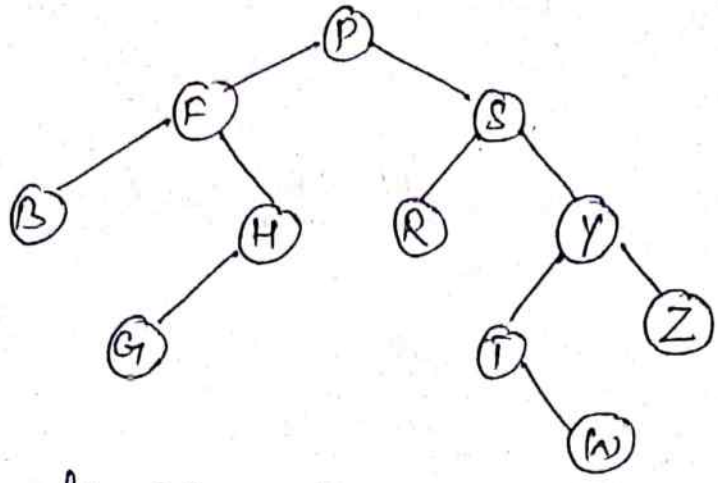


fig. Binary Tree

2) Creation of Binary Tree from Post order and Inorder Traversals.

Q Create a binary tree :

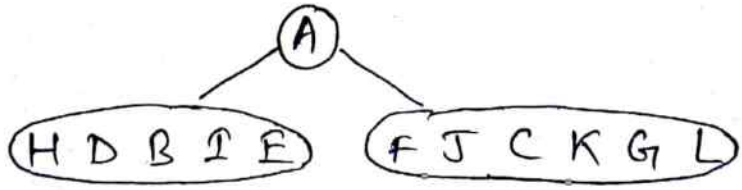
postorder : H D I E B J F K L G C A

Inorder : H D B I E A F J C K G L

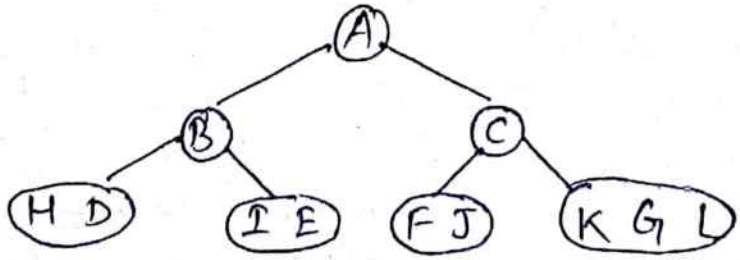
Soln Postorder is Left – Right – Root

Inorder is Left – Root – Right

Step(1)



Step(2)



Step(3)

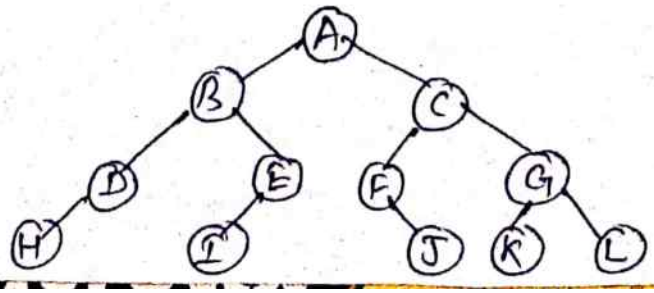


fig. Binary Tree

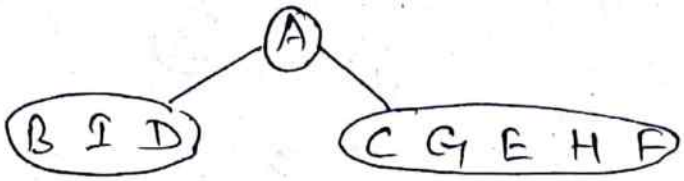
Scanned by CamScanner

Q. Create a Binary Tree:

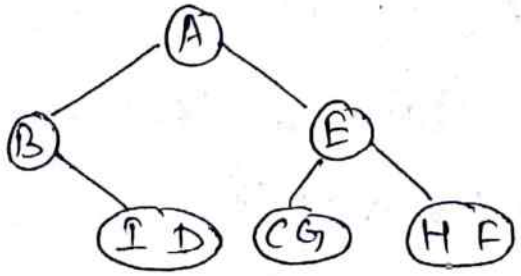
In-order : B I D A C G E H F

Post-order : I D B G C H F E A.

Soln step(1)



Step(2)



Step(3)

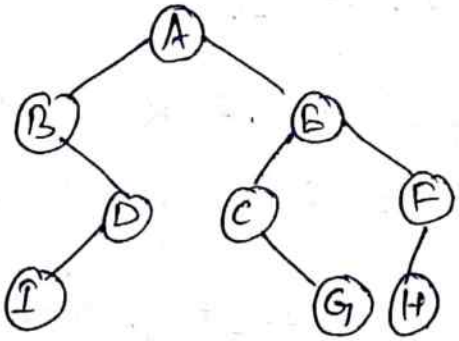


fig. Binary Tree

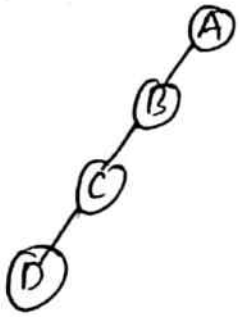
Scanned by CamScanner

3) Creation of Binary Tree from Pre-order and postorder traversals

We can construct a unique binary Tree from pre-order and in-order traversals as well as post-order and in-order traversals.

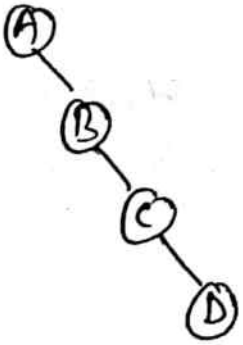
But preorder and postorder sequences don't provide enough information to create a unique binary tree.

For example:



preorder : A B C D

postorder : D C B A



preorder : A B C D

post order : D C B A

Therefore, it is not possible to construct a unique binary tree from preorder and post order traversals.

Q Construct a Binary Tree:

preorder : 1 2 4 5 3 6 8 9 7

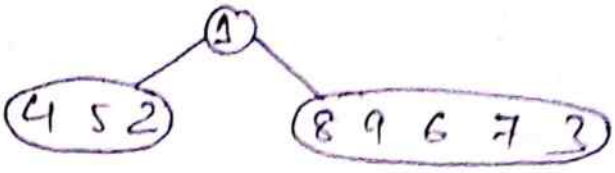
postorder : 4 5 2 8 9 6 7 3 1

Soln preorder is Root – Left – Right

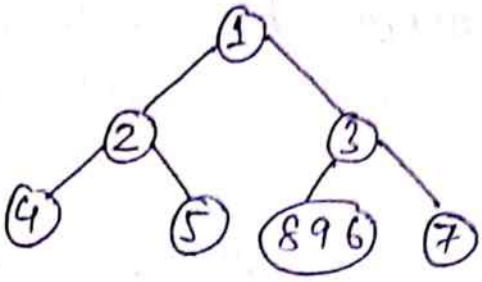
Postorder is Left – Right – Root

Scanned by CamScanner

Step(1)



Step(2)



Step(3)

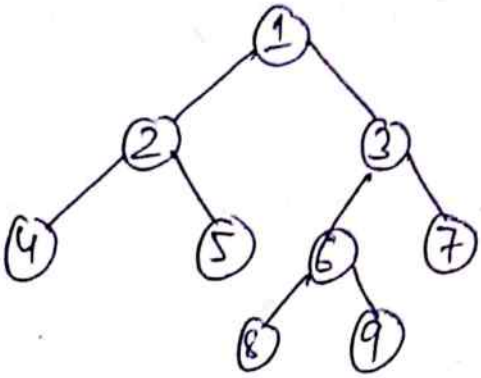


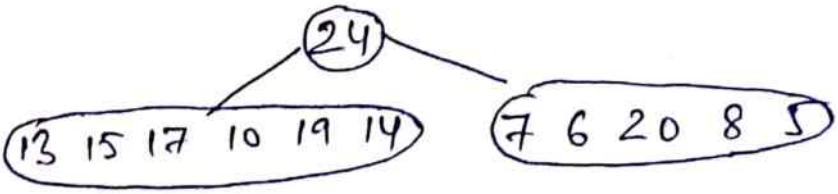
fig. Binary Tree.

Q Construct a Binary Tree

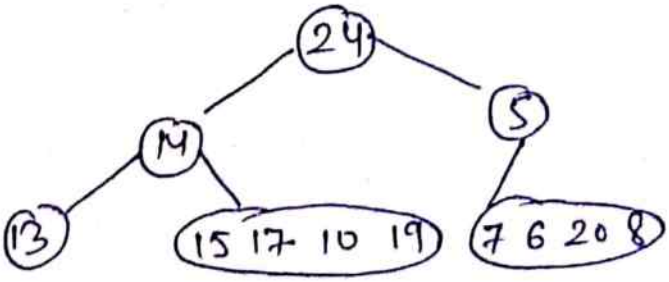
preorder: 24, 14, 13, 19, 17, 15, 10, 5, 8, 6, 7, 20

postorder: 13, 15, 17, 10, 19, 14, 7, 6, 20, 8, 5, 24

Soln step(1)

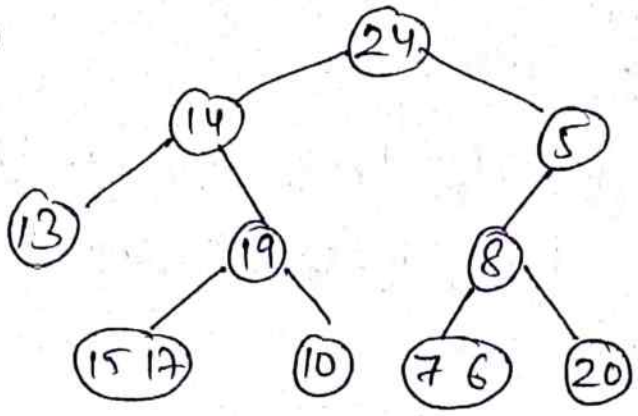


Step (2)



Scanned by CamScanner

Step (3)



Step(4)

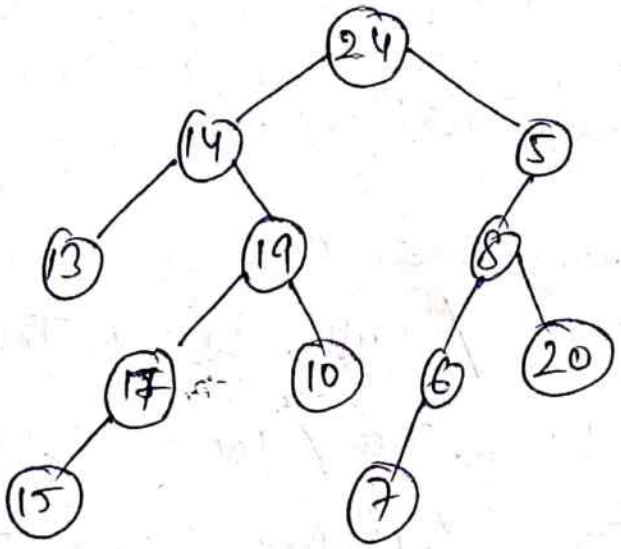


fig. Binary Tree

Scanned by CamScanner

Types of Binary Tree

In a Binary Tree, every node have atmost 2 child. or we can say degree of every node is atmost 2.

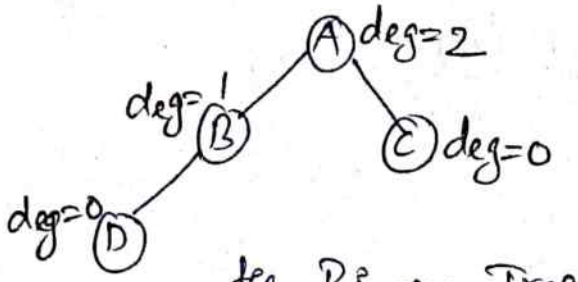


fig. Binary Tree

Different types of Binary Tree are:

(1) Strict Binary Tree / Fully Binary Tree
(2) Almost Complete Binary Tree / Incomplete Binary Tree
(3) Complete Binary Tree / Perfect Binary Tree
(4) Left - skewed Binary Tree
(5) Right - skewed Binary Tree

Strict Binary Tree

- In Strict Binary Tree, the degree of every node is either 0 or 2.
- We can say that in a strict Binary tree, a node can have either 0 children or 2 children.

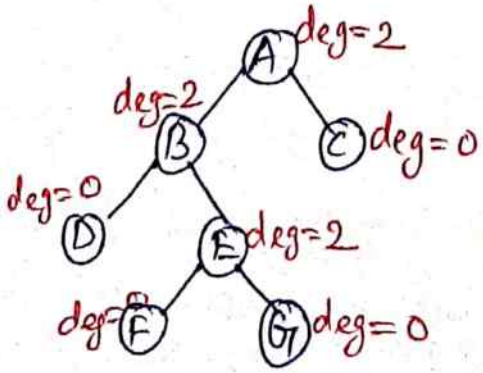


fig Strict Binary Trees

Scanned by CamScanner

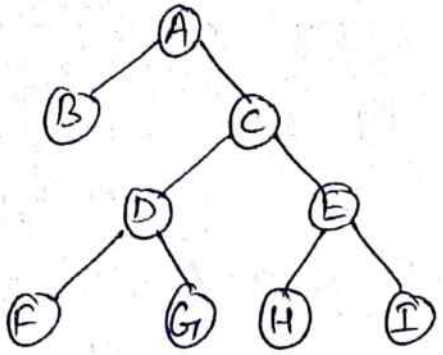


fig. Strict Binary Tree

Note :> A Strict Binary Tree is also defined as a tree in which every node must have two children except leaf nodes.

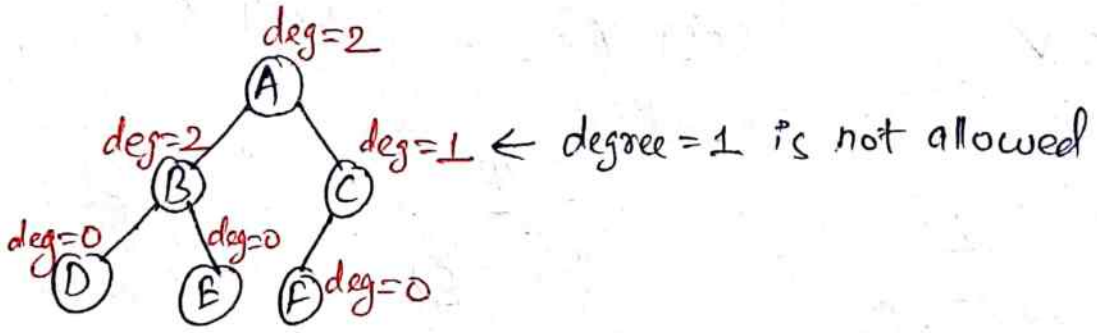


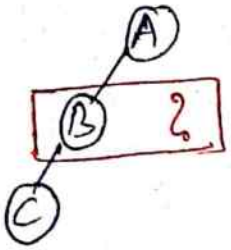
fig. Not a Strict Binary Tree

Note :> A Strict Binary Tree with n leaves always contain 2n−1 nodes.

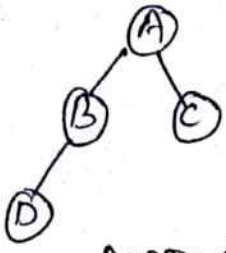
Almost Complete Binary Tree

In Almost Complete Binary Tree, at every node after completing left only, go to right and after completing previous level, then go to next level, then it is called as almost complete Binary Tree. (ACBT)

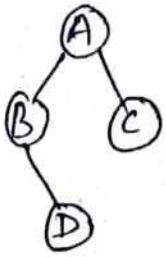
Scanned by CamScanner



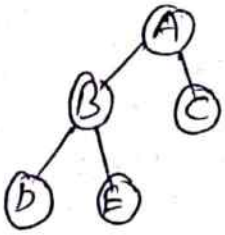
ACBT ✗



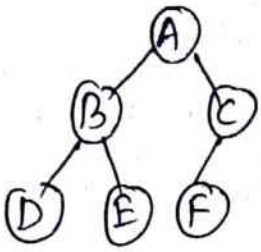
ACBT ✓



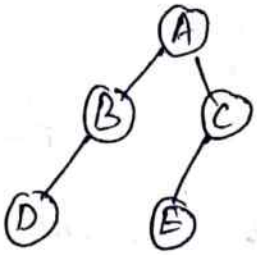
ACBT ✗



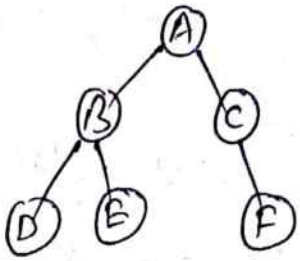
ACBT ✓



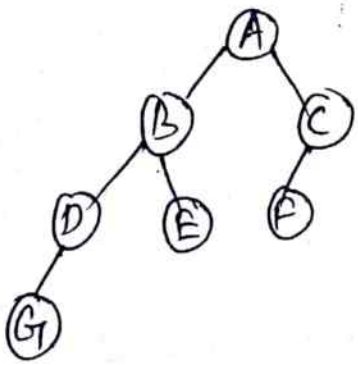
ACBT ✓



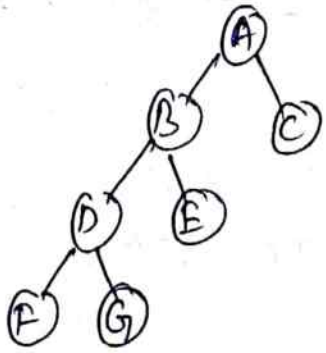
ACBT ✗



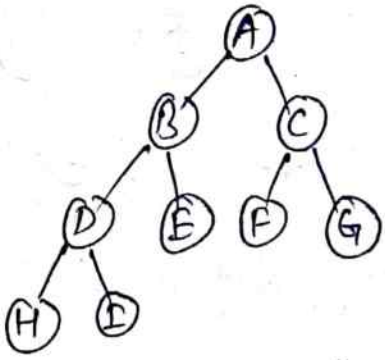
ACBT ✗



ACBT ✗



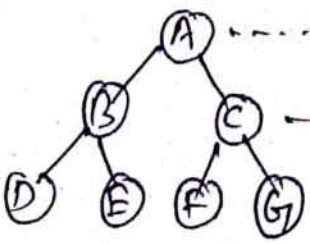
ACBT ✗



ACBT ✓

Complete Binary Tree

- Every node must have two children in all the levels except the last level. and each level must have exactly 2^L nodes where L is the level.



..... level 0 $= 2^0 = 1$ node

... level 1 $= 2^1 = 2$ nodes

... level 2 $= 2^2 = 4$ nodes

CBT ✓

Scanned by CamScanner

Left-skewed Binary Tree

Every node should have only left child.

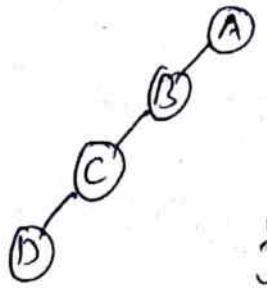


fig. left-skewed Binary tree

Right-skewed Binary Tree

Every node should have only right child.

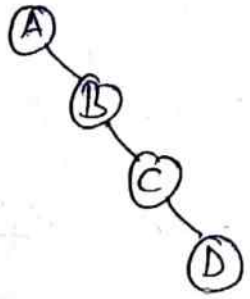


fig. Right-skewed Binary Tree

Q1 Write a C function to find height of a binary tree.

```
int height (struct node *root)
{
    if (root == NULL)
        return 0;
    if (root->left == NULL && root->right == NULL)
        return 0;
    else {
        int lheight = height (root->left);
        int rheight = height (root->right);
```

Scanned by CamScanner

```
    if (lheight > rheight)
            return (lheight+1);
    else
            return (rheight +1);
    }
}
```

Q2 Function to count number of nodes

```
int count (struct node * root)
{
  if (root == NULL)
          return 0;
  else
        return (1 + count (root->left) + count (root->right));
}
```

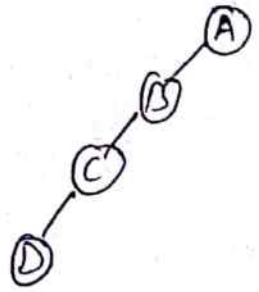
Q3 Function to Find the number of leaf nodes

```
int leafcount (struct node * root)
{  if (root == NULL)
        return 0;
   if (root->left == NULL && root->right == NULL)
        return 1;
   else
      return (leafcount (root->left) + leafcount (root->right));
}
```

Scanned by CamScanner

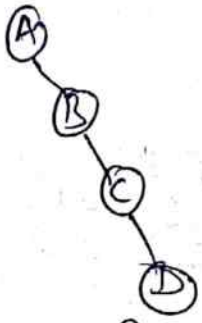
NOTE :→ For a Binary Tree, having n nodes and L levels minimum height possible is $\lceil \log_2(n+1)-1 \rceil$ and maximum height possible is n.

Proof: Maximum height is when binary tree is left or right skewed, i.e. one node at each level.



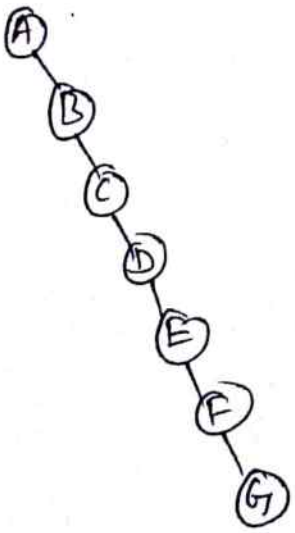
Left-skewed

(or)



Right-skewed

Let $n=7$

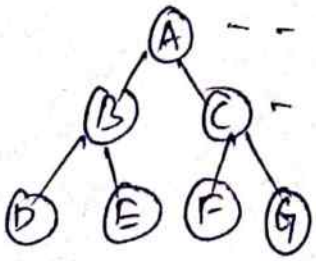


$\therefore$ Maximum level $= 7$

Maximum height $= 7$

$\therefore$ in terms of complexity, maximum height $= O(n)$.

Minimum height is when binary tree is complete.



level $0 = 2^0$

level $1 = 2^1$

level $2 = 2^2$

$\vdots$

level $L = 2^L$

Scanned by CamScanner

$\therefore$ total no. of nodes $= 2^0 + 2^1 + 2^2 + \cdots + 2^L \rightarrow (1)$

It is a G.P series.

G.P series is $a + ar^1 + ar^2 + \cdots$

$\therefore$ Sum $= \dfrac{a(r^n - 1)}{r-1}$ here, n is no. of terms

here, $a = 1$

$r = 2$

$n = L+1$

$\therefore$ equ (1) $\Rightarrow n = \dfrac{1(2^{L+1} - 1)}{2-1}$

$\Rightarrow n = 2^{L+1} - 1$

$\Rightarrow 2^{L+1} = n+1$

take log on both sides

$\log_2(2^{L+1}) = \log_2(n+1)$

$\Rightarrow L+1 = \log_2(n+1)$.

$\Rightarrow L = \log_2(n+1) - 1$

$\therefore$ Maximum level is the maximum height

$\therefore$ Maximum height $= \log_2(n+1) - 1$

Scanned by CamScanner

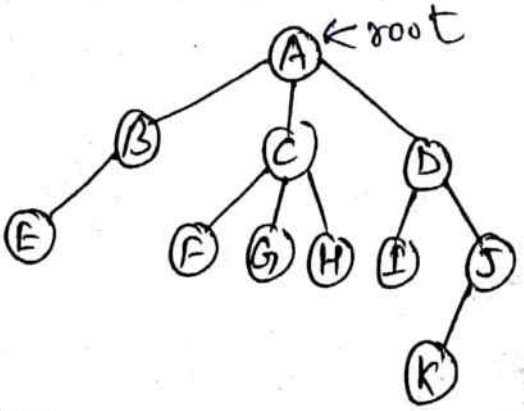
Algorithm to convert General Tree into Binary Tree

Rule(1): Root of Binary Tree = Root of General Tree

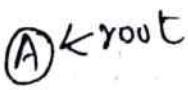
Rule (2): Left child of a node in the binary tree } = { Leftmost child of the node in the General tree.

Rule (3): Right child of a node in the binary tree } = { Right sibling of the node in the General tree.

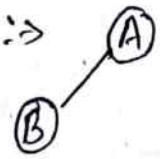
Q Convert Given General tree to Binary Tree



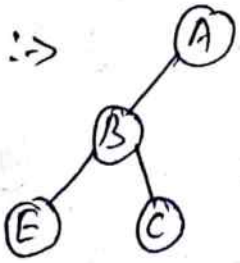
Soln step(1) :>



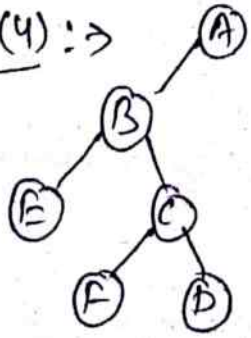
step (2) :>



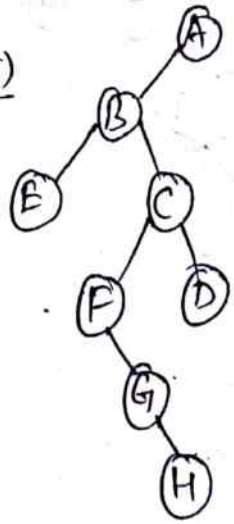
step(3) :>



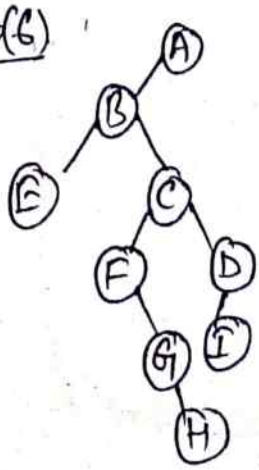
step(4) :>



step (5)



step(6)



Scanned by CamScanner

Step(7)

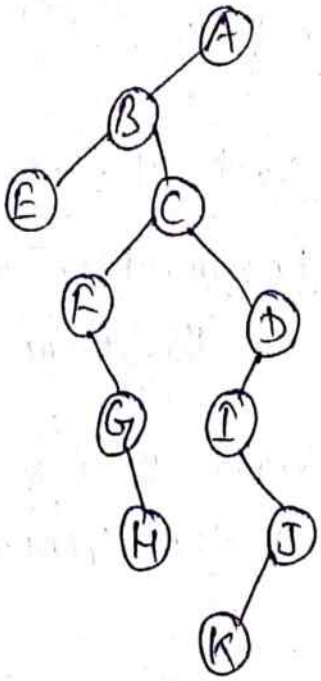
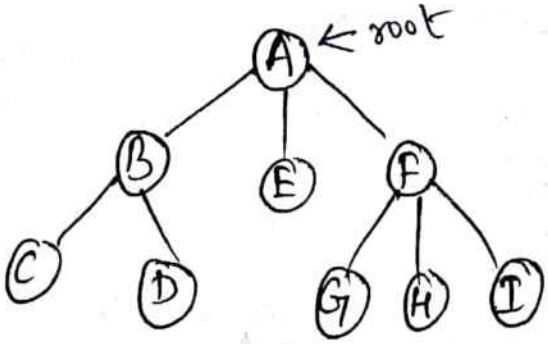


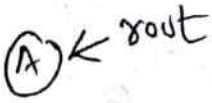
fig Binary Tree.

Q2

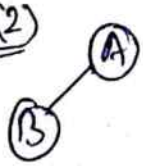


Soln

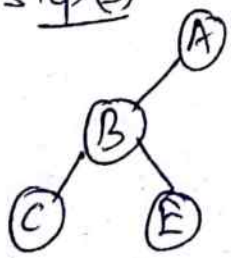
Step(1)



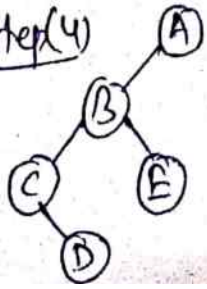
Step(2)



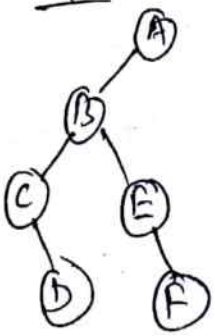
Step(3)



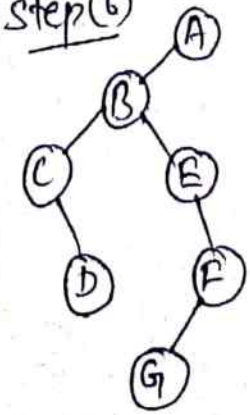
Step(4)



Step(5)



Step(6)



Step(7)

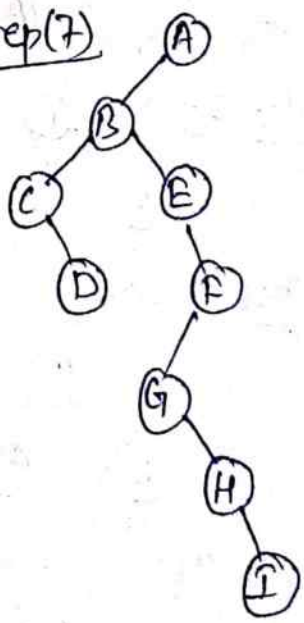


fig Binary Tree

Scanned by CamScanner

Q

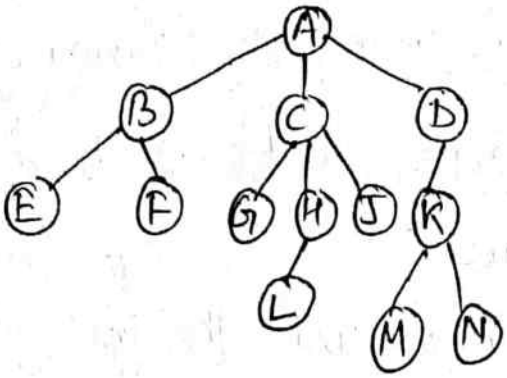
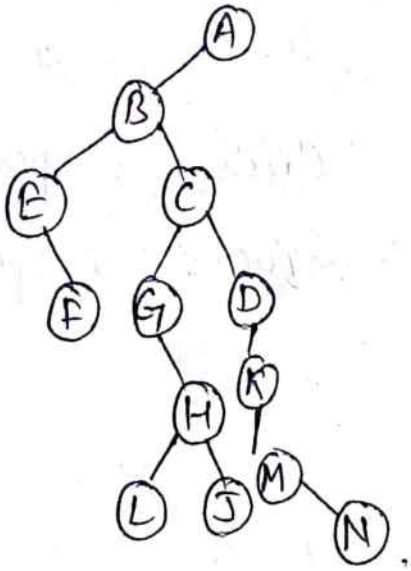


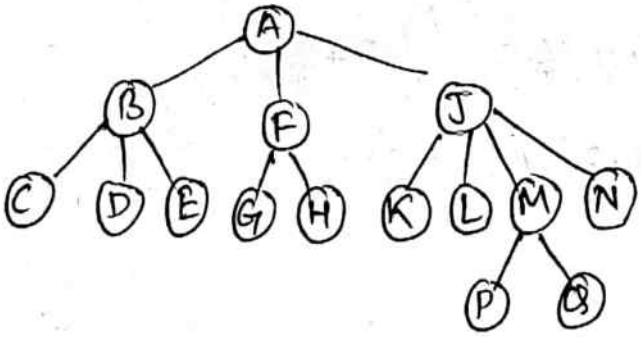
fig. General Tree

Soln

∴ Equivalent Binary Tree is



Q



Soln

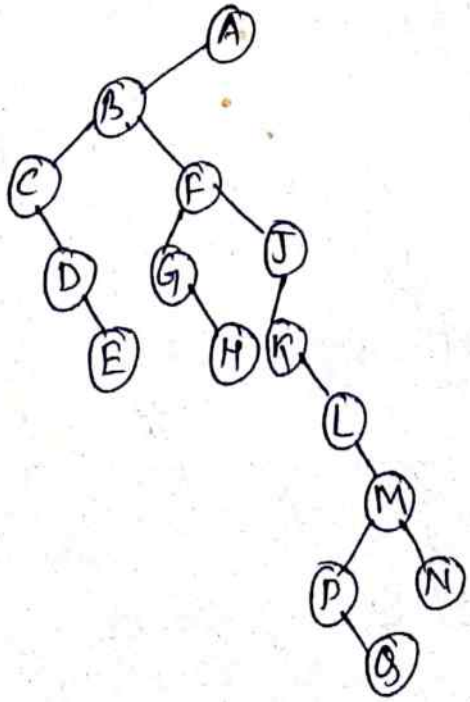


fig. Binary Tree

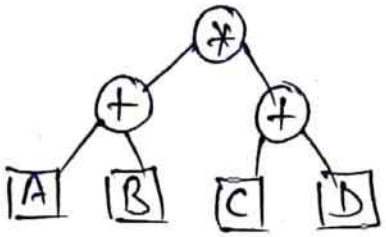
Scanned by CamScanner

Converting Algebraic Expression into Binary Tree

The arithmetic expression represented as binary trees are known as Expression trees.

In this, the root is operator and the left and right children are operands.

Example:> (A+B) * (C+D)



Circle for operator
Square for operand.

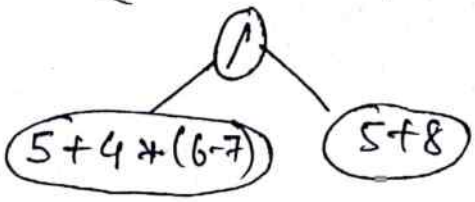
fig. Expression tree

Q Draw the binary tree to represent the following expression: (5+4*(6-7))/(5+8)

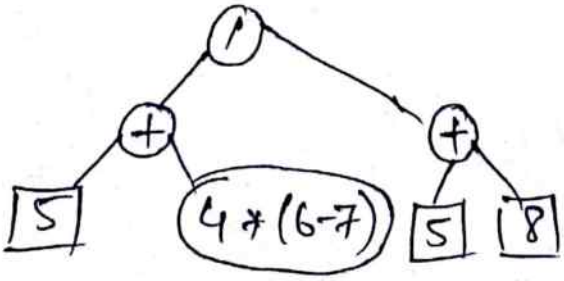
Soln

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

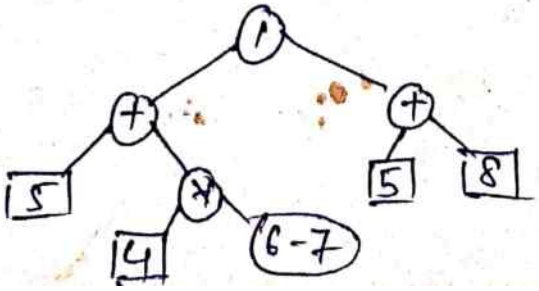
step(1)



step(2)



step(3)



Scanned by CamScanner

Step (4)

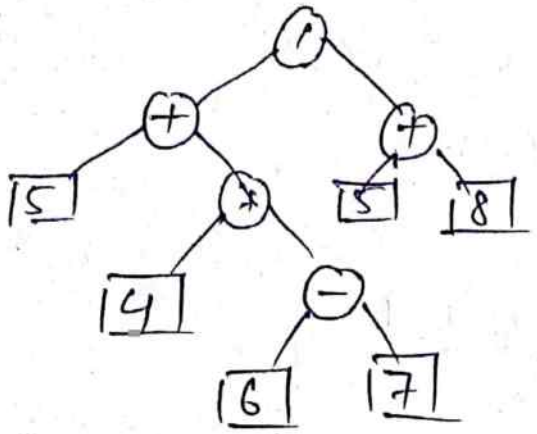
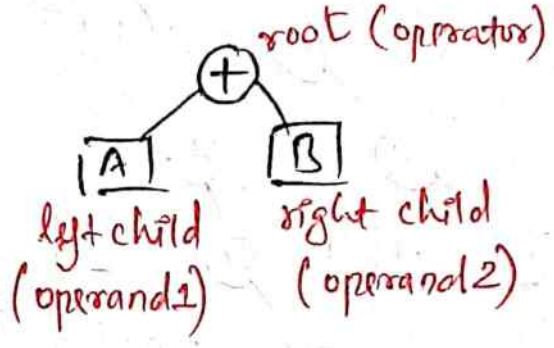


fig. Expression Tree.

NOTE (1) $A+B \Rightarrow$



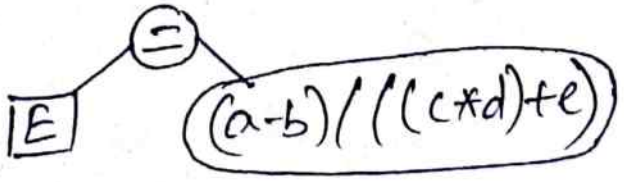
(2) A preorder traversal of expression tree gives prefix equivalent of expression.

(3) A postorder traversal of expression tree gives postfix equivalent of expression.

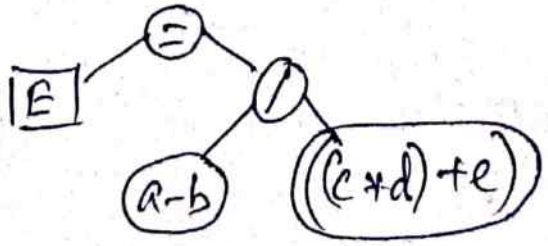
Que Construct a binary tree for the algebraic expression

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

Soln Step(1)

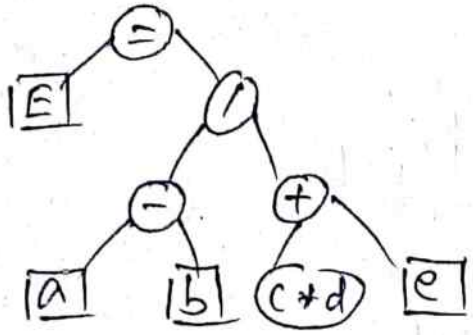


Step(2)



Scanned by CamScanner

step(3)



step(4)

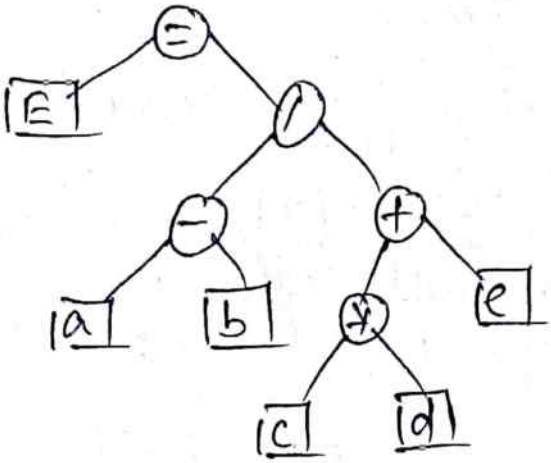
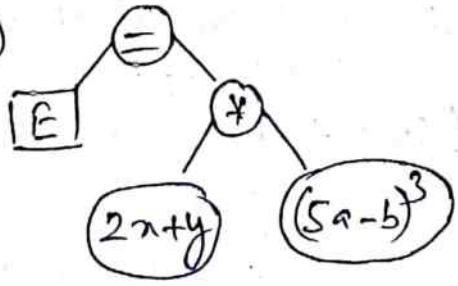


fig. Expression tree.

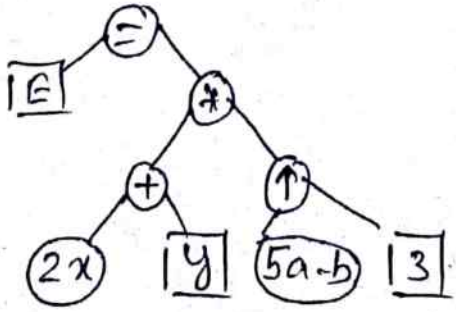
Q Construct a binary tree for the algebraic expression.

$$E = (2x+y)(5a-b)^3$$

Soln: step(1)

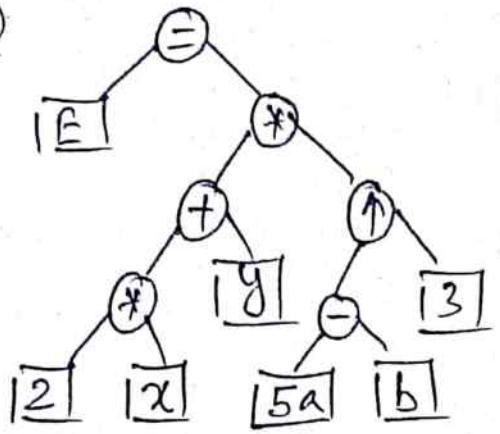


Step(2)



Scanned by CamScanner

Step(3)



Step(4)

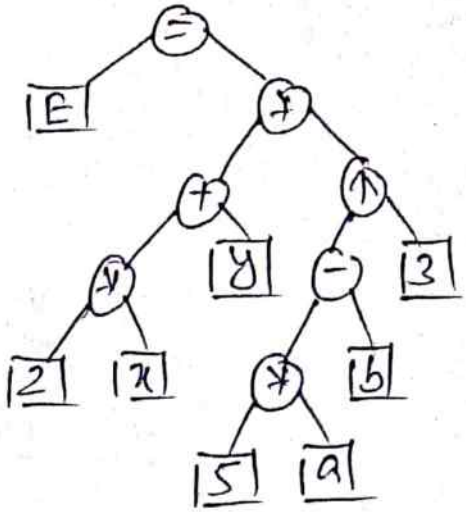


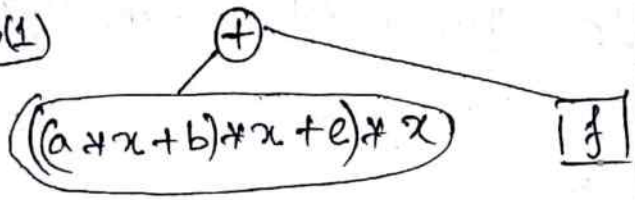
fig. Binary Tree
(Expression Tree)

Que For the following expression :

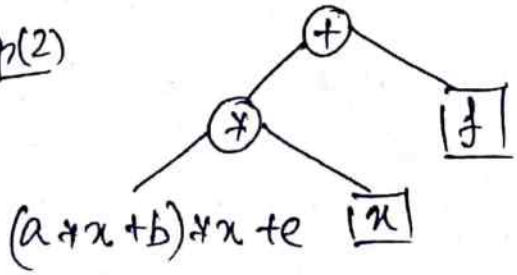
$$((a*x+b)*x+e)*x+f$$

Construct the expression tree. Find the equivalent postfix notation.

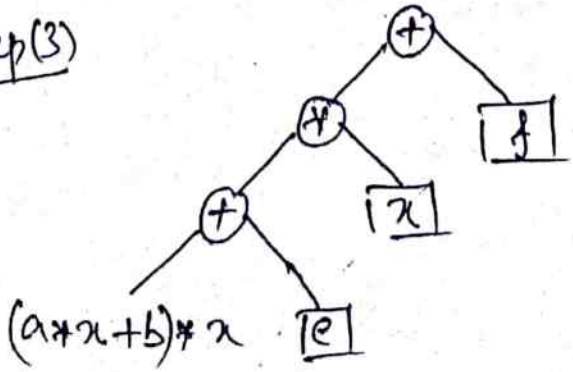
Soln Step(1)



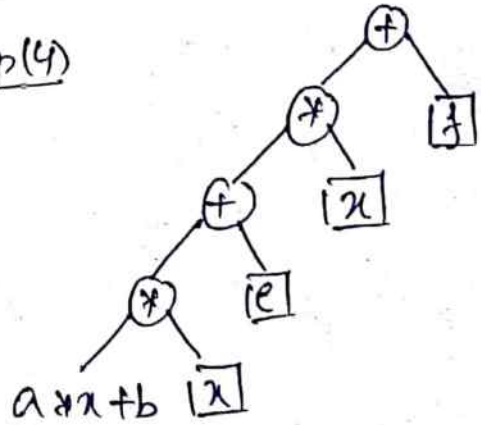
Step(2)



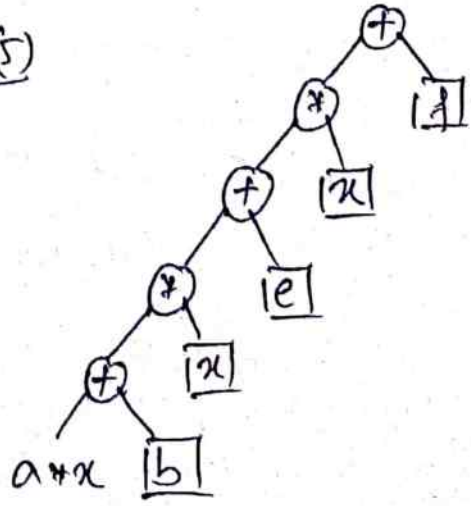
Step(3)



Step(4)



Step(5)



Scanned by CamScanner

Step (6)

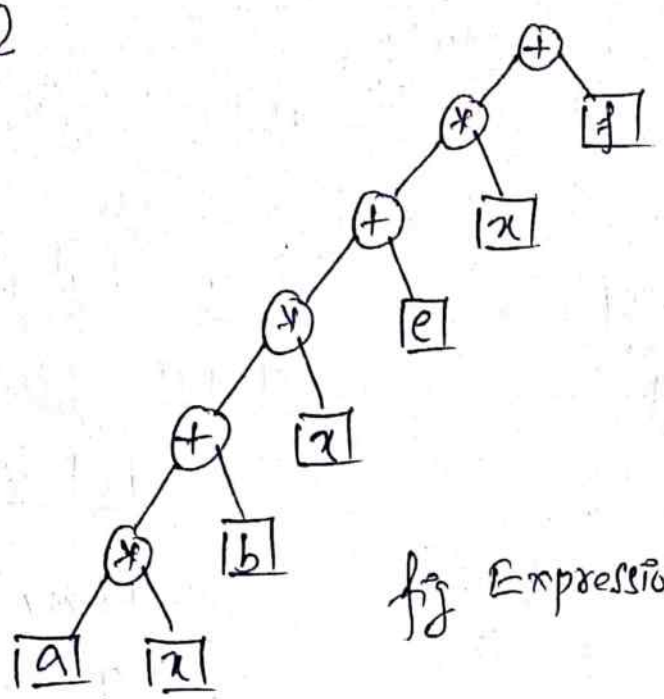


fig Expression Tree

∴ Postfix notation is $ax*b+x*e+x*f+$

Que How many different trees are possible with 10 nodes?

Soln Let $n=3$

∴ if there are n nodes, there exist 2^n-n different binary trees.

∴ 5 different trees are possible.

∴ For $n=10$ nodes

∴ $2^{10}-10$ different trees are possible

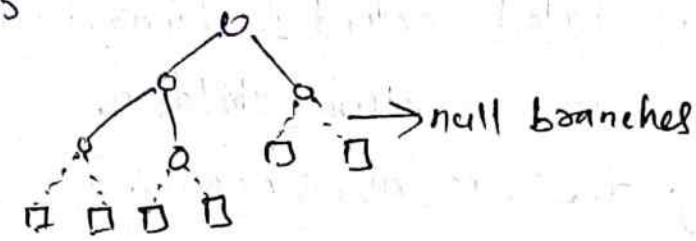
$= 1024-10$

$= 1014$

Scanned by CamScanner

Q A binary tree with 20 nodes has ___ NULL branches.

Soln Let $n=5$



It has 6 null branches.

$\therefore$ A binary tree with n nodes has exactly $n+1$ null branches

$\therefore$ A binary tree with 20 nodes has 21 NULL branches.

Scanned by CamScanner

Extended Binary Trees

- A binary tree is called extended Binary tree if every node of tree has zero or two children.
- Extended Binary tree is also known as 2-tree.
- A binary tree can be converted to an extended binary tree by adding new nodes to its leaf nodes and to the nodes that have only one child.
- Each node with 2 children are called internal nodes. and nodes with 0 children are called external nodes.

O → internal nodes □ → external nodes

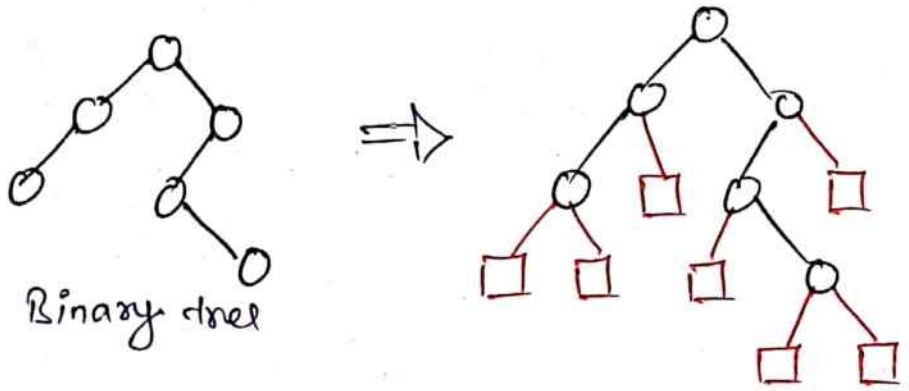


fig. Binary tree fig. Extended Binary tree

$$N_E = N_I + 1$$

Number of external nodes is 1 more than number of internal nodes.

Internal Path length $= 0 + 1 + 1 + 2 + 2 + 3$

$L_I = 9$

External Path length $(L_E) = 3 + 3 + 2 + 3 + 4 + 4 + 2$

$= 21$

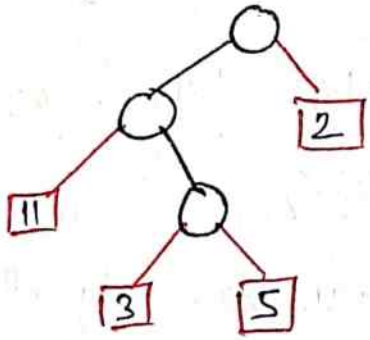
$$\therefore L_E = L_I + 2N_I$$

Scanned by CamScanner

The (external) weighted path length P of a tree T is defined to be the sum of weighted path length

$$P = w_1 L_1 + w_2 L_2 + \cdots + w_n L_n$$

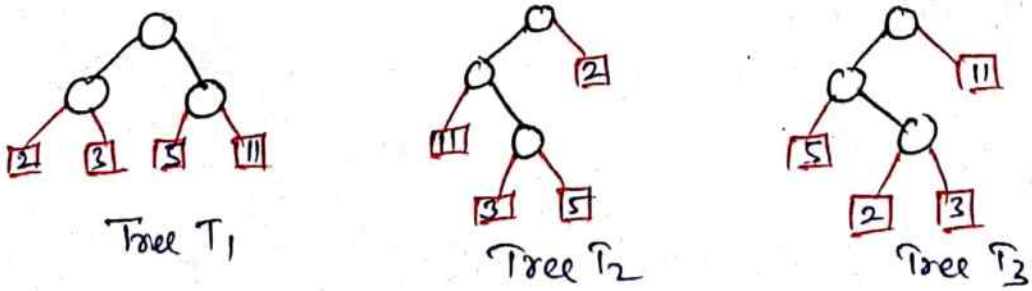
where w_i and L_i denotes the weight and path of external nodes N_i.



$$P = 11 \times 2 + 3 \times 3 + 5 \times 3 + 2 \times 1$$
$$= 22 + 9 + 15 + 2$$
$$= 48$$

Suppose a list of n weights are given: $w_1, w_2, \ldots w_n$
Among all the 2-trees with n external nodes and with the given n weights, find a tree T with a minimum weighted path length.

Huffman Algorithm is used to solve this problem.



$P_1 := 2 \times 2 + 3 \times 2 + 5 \times 2 + 11 \times 2 = 42$
$P_2 := 2 \times 1 + 3 \times 3 + 5 \times 3 + 11 \times 2 = 48$
$P_3 := 2 \times 3 + 3 \times 3 + 5 \times 2 + 11 \times 1 = 36$ (min)

Scanned by CamScanner

Huffman Coding

- Huffman coding is a loseless data compression technique for encoding and decoding of messages.
- Huffman coding is a variable length coding.
- Huffman tree is an application of extended binary tree.
- Huffman tree is constructed based on the frequency of symbols of a given message.
- The main aim of using Huffman algorithm is to minimize the length of the encoded message by assigning a shorter bit to the frequently occurring symbols in the message.

i.e. Most frequent characters have smallest codes and longer codes for least frequent characters.

- Two major part in Huffman coding:-

(1) Build the huffman tree from input characters.

(2) Traverse the huffman tree and assign codes to each characters.

Scanned by CamScanner

Why Huffman Coding ?

Message: ABBCDBCCDAABDEEEBEAB

character	ASCII	Bits	
A	65	01000001	= 8 bits
B	66	01000010	= 8 bits
C	67	01000011	= 8 bits
D	68	01000100	= 8 bits
E	69	01000101	= 8 bits

Fixed length

∴ To transfer the above message, total bits required

$= 20 \times 8 = 160$ bits

Here, we are using fixed length coding to transfer the message.

Now suppose we are using some compression technique to reduce the number of bits required to transfer the message.

For this, we are using the concept as follows:

For 2 bits

0 0
0 1
1 0
1 1

Four different combinations are possible

i.e 4 different characters can be represented.

For 3 bits

0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

8 different combinations are possible.

Scanned by CamScanner

But in our case, we are having only 5 different characters A, B, C, D, E

∴ Instead of using 8 bits for each characters, we can use 3 bits for each character after our compression technique.

∴

$$\left.\begin{array}{l} 0\ 0\ 0 \rightarrow A \\ 0\ 0\ 1 \rightarrow B \\ 0\ 1\ 0 \rightarrow C \\ 0\ 1\ 1 \rightarrow D \\ 1\ 0\ 0 \rightarrow E \end{array}\right\} \text{still we using Fixed length coding.}$$

∴ after applying this compression technique,

total bits required $= 20 \times 3$

$= 60$ bits

But here we have a problem.

Question is how the receiver will know that we have used

0 0 0 for A
0 0 1 for B
0 1 0 for C
0 1 1 for D
1 0 0 for E

∴ we need to send this message / table along with our original message.

So to transfer this table, total bits required will be

$$= \underset{\text{no. of char}}{5} \times \underset{\substack{\text{8 bits per char} \\ \text{(ASCII)}}}{8} + \underset{\text{no. of char}}{5} \times \underset{\text{3 bits : 000 for A and so on.}}{3}$$

$= 55$ bits

Scanned by CamScanner

∴ Total bits Required to send the required message.

$= 60 + 55$

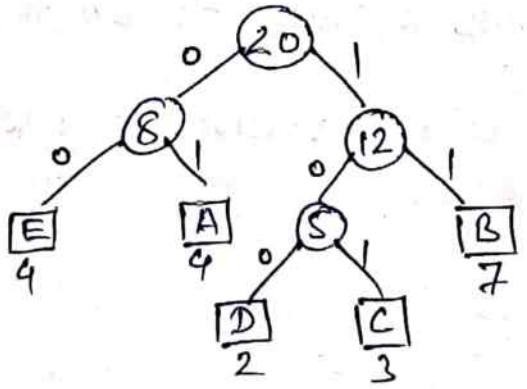
$= 115$ bits

Huffman coding is a variable length coding.
i.e different bits length are used for characters.

How Huffman Coding works?

Message: A B B C D B C C D A A B B E E E B E A B

Character	Frequency /count
A	4
B	7
C	3
D	2
E	4



A → 0 1
B → 1 1
C → 1 0 1
D → 1 0 0
E → 0 0

} Huffman codes

fig Huffman Tree

∴ Total code length $= 4 \times 2 + 7 \times 2 + 3 \times 3 + 2 \times 3 + 4 \times 2$

$= 8 + 14 + 9 + 6 + 8$

$= 45$ bits

Average code length $= \frac{45}{20}$

$= \frac{9}{4} = 2.2$ bits / char

Scanned by CamScanner

To send the message, total bits required = 45 bits

Also, to send A → 01, B → 11, C → 101, D → 100, E → 00, total bits required

$$= \underset{\text{no. of char}}{5} \times \underset{\text{ASCII for char}}{8} + (2+2+3+3+2)$$

$$= 40 + 12$$

$$= 52 \text{ bits}$$

∴ To transfer the message, total bits required

$$= 45 + 52$$

$$= 97 \text{ bits}$$

NOTE :→ Huffman codes are Pre-fix free codes.

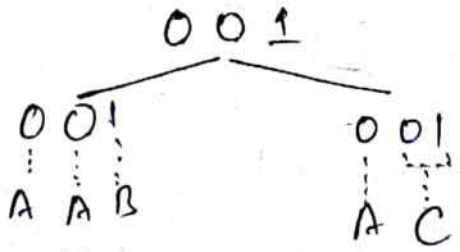
Prefix code :→ No code is prefix of another code.

Example

A — 0
B — 1
C — 01

Suppose message transmitted is 001

Let see how receiver decodes this message.



here we have decoded message 001 in two forms.

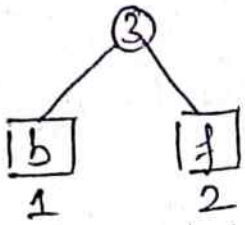
This ambiguity occurs because prefix of 01 i.e 0 is for A.

But in Huffman codes, No code is prefix of another code.

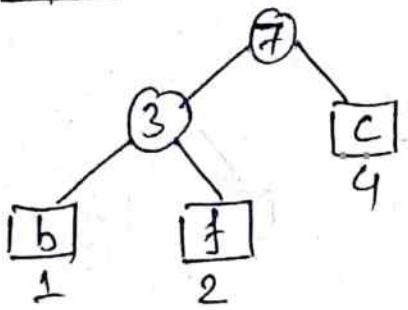
Scanned by CamScanner

Q1 Message, M: a=60, b=1, c=4, d=8, e=25, f=2

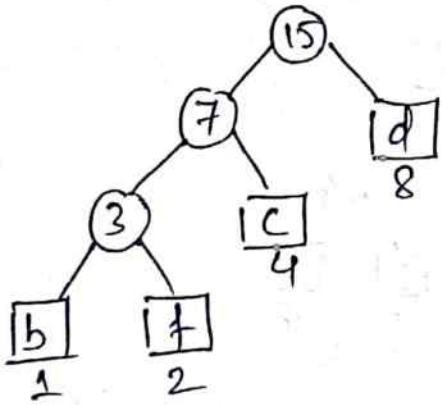
step(1)



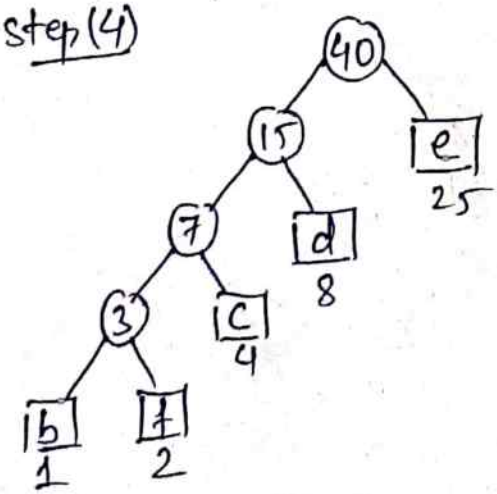
step(2)



step (3)



step(4)



step (5)

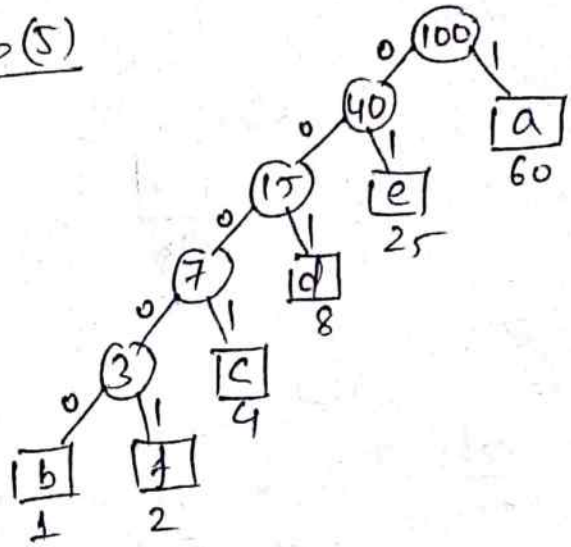


fig. Huffman Tree

∴ Huffman codes are

a = 1

b = 00000

c = 0001

d = 001

e = 01

f = 00001

∴ total code length

$= 60\times1 + 1\times5 + 4\times4 + 8\times3 + 25\times2 + 2\times5$

$= 165$ bits

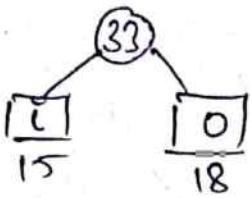
∴ Average $= \frac{165}{100} = 1.65$ bits/char

Scanned by CamScanner

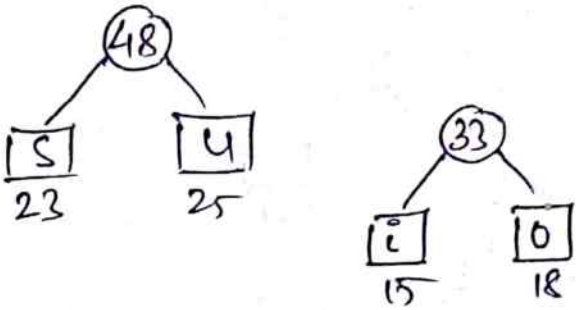
Q2 Message, M: a=45, e=30, i=15, o=18, u=25, s=23, t=35

Soln

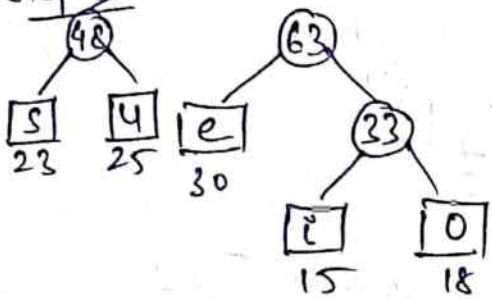
step(1)



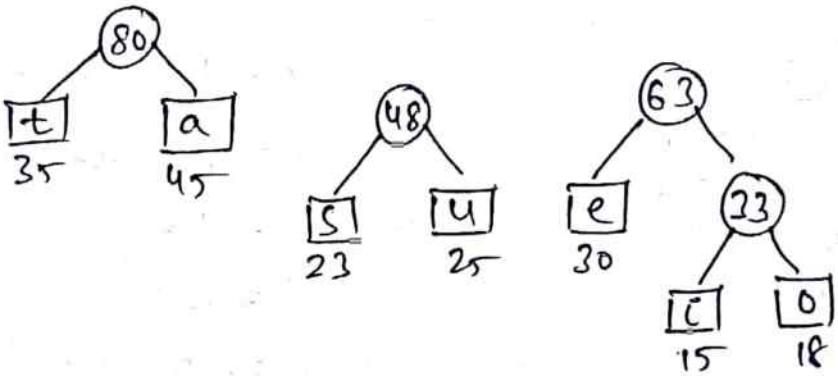
step (2)



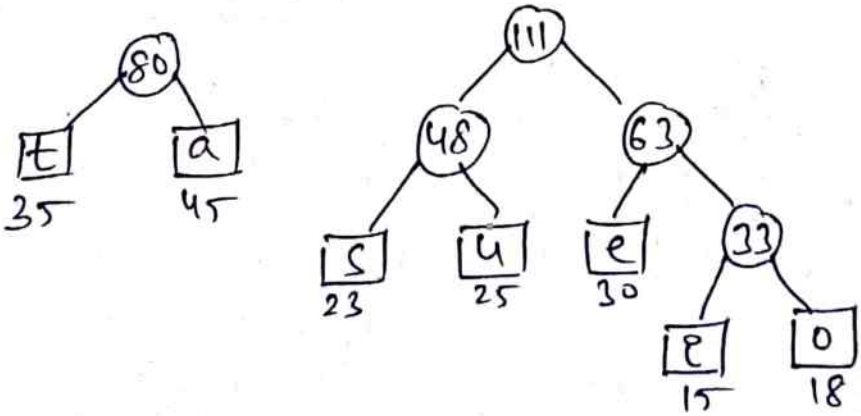
step (3)



step (4)



step(5)



Scanned by CamScanner

Step(6)

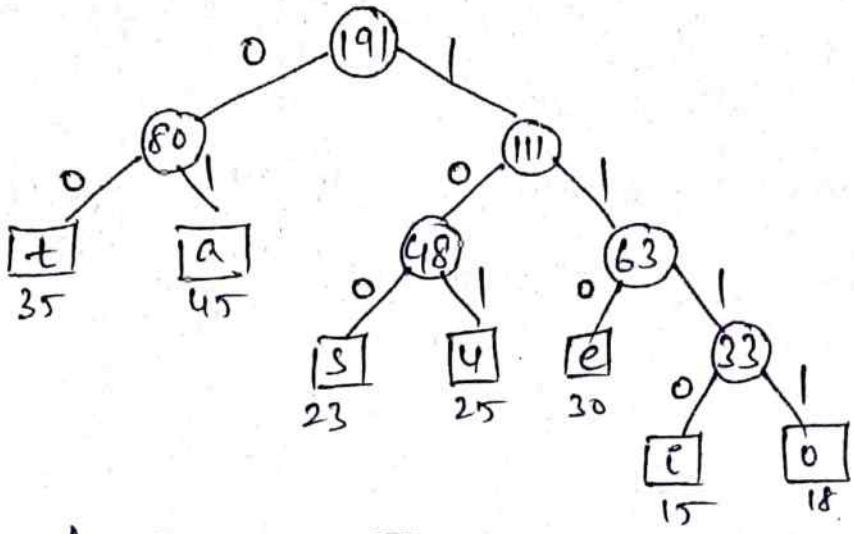


fig. Huffman Tree

Huffman codes are

a = 01

e = 110

i = 1110

o = 1111

u = 101

s = 100

t = 00

total code length = 45×2 + 30×3 + 15×4 + 18×4 + 25×3 + 23×3 + 35×2

= 526 bits

Average code length = $\frac{526}{191}$ = 2.75 bits/char

Scanned by CamScanner

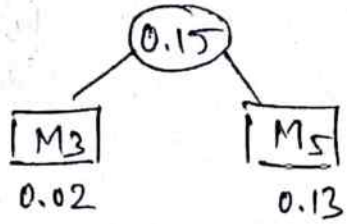
Q3 Message M ⇒ $M_1 = 0.45$, $M_2 = 0.18$, $M_3 = 0.02$, $M_4 = 0.22$, $M_5 = 0.13$

Find Huffman codes and decode the message

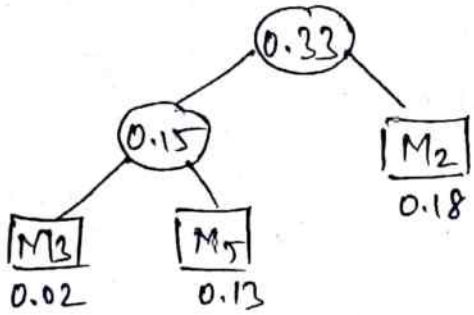
1101101100111011001011011

Soln

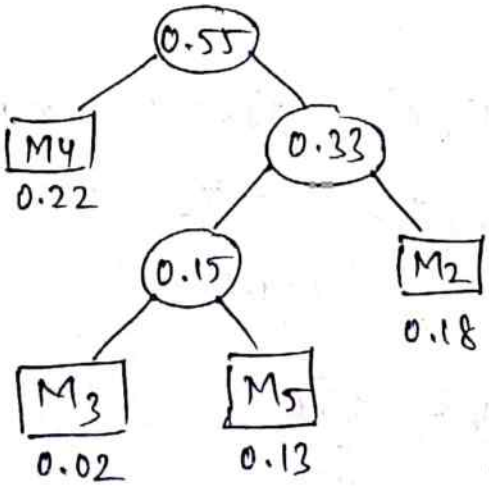
Step (1)



Step (2)



Step (3)



Step (4)

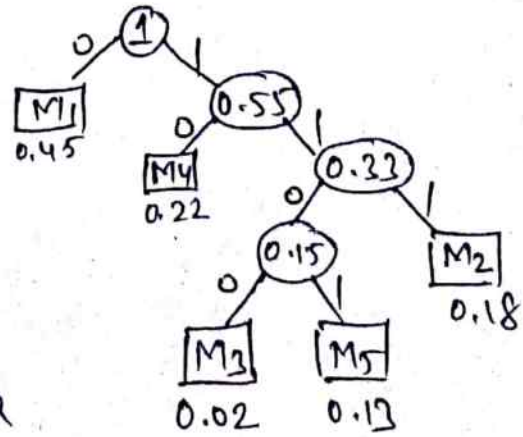


fig. Huffman Tree

Scanned by CamScanner

Huffman Codes are

$M_1 = 0$

$M_2 = 111$

$M_3 = 1100$

$M_4 = 10$

$M_5 = 1101$

Total code length $= 0.45 \times 1 + 0.18 \times 3 + 0.13 \times 4 + 0.22 \times 2 + 0.13 \times 4$

$= 2.03$ bits

Average code length $= \frac{2.03}{1} = 2.03$ bits/char

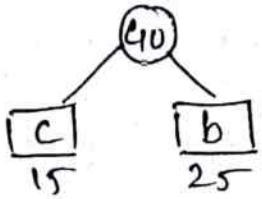
$\therefore$ Decode: $\underbrace{1101}_{M_5}\underbrace{10}_{M_4}\underbrace{1100}_{M_3}\underbrace{111}_{M_2}\underbrace{0}_{M_1}\underbrace{1100}_{M_3}\underbrace{10}_{M_4}\underbrace{1101}_{M_5}$

Q4 Find an optimal Huffman code for the following set of frequencies:

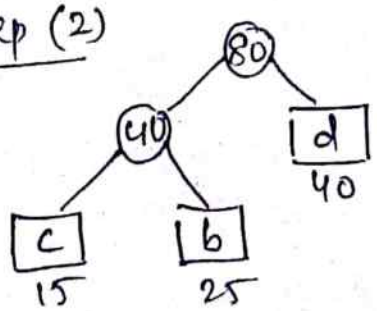
$a = 50,\ b = 25,\ c = 15,\ d = 40,\ e = 75$

Soln

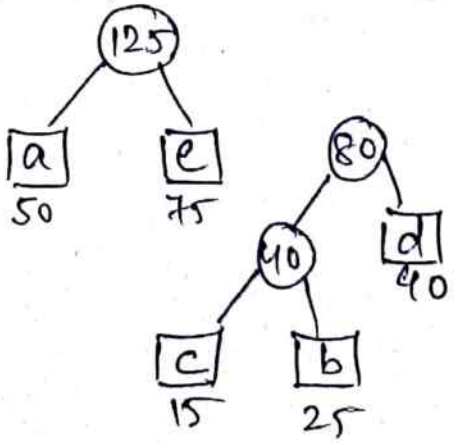
step (1)



step (2)



step(3)



Scanned by CamScanner

Step(4)

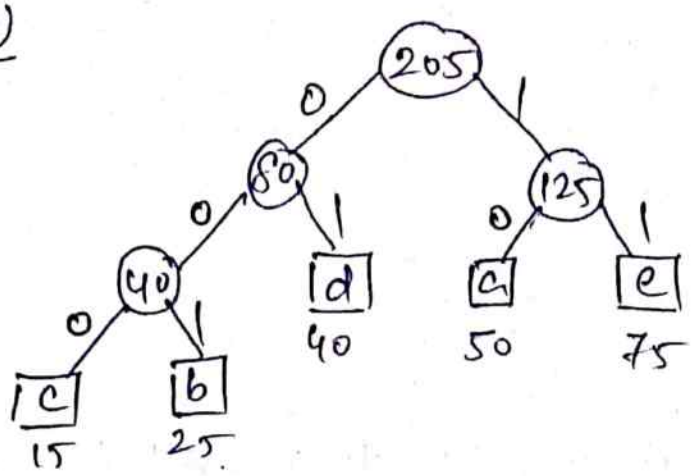


fig. Huffman Tree

∴ Huffman codes are a = 10
b = 001
c = 000
d = 01
e = 11

∴ Total code length = 50×2 + 25×3 + 15×3 + 40×2 + 75×2

= 450 bits

∴ Average bits required $= \frac{450}{205}$

= 2.195 bits/char

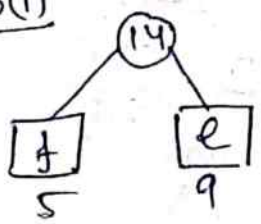
Scanned by CamScanner

Q5 Find an optimal Huffman code for the following frequencies:

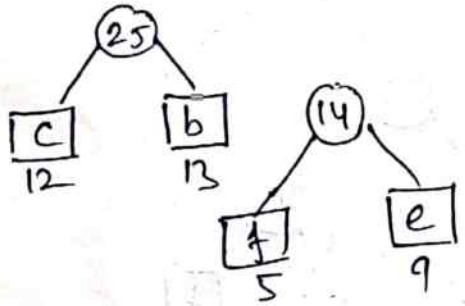
$a=45$, $b=13$, $c=12$, $d=16$, $e=9$, $f=5$

Soln

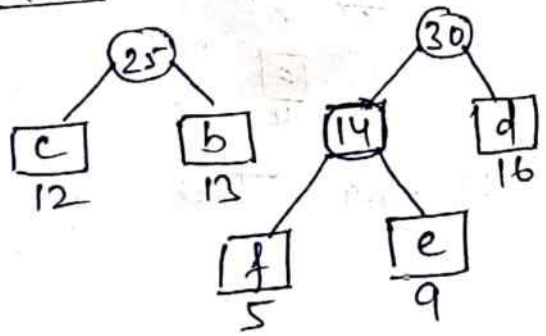
step(1)



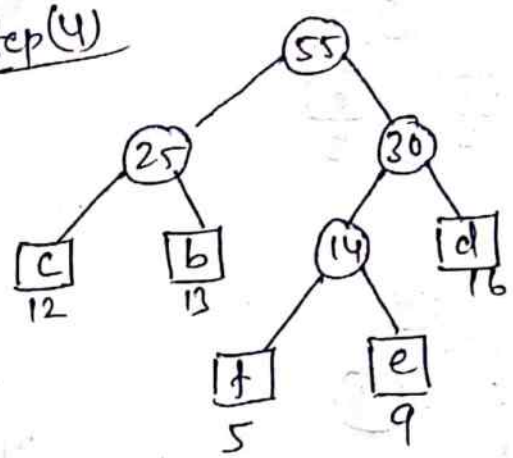
step(2)



step (3)



Step(4)



step(5)

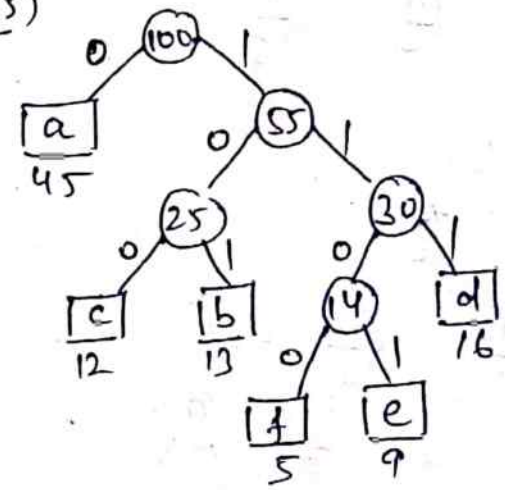


fig. Huffman Tree

∴ Huffman codes are
$a = 0$
$b = 101$
$c = 100$
$d = 111$
$e = 1101$
$f = 1100$

Total code length $= 45\times1 + 13\times3 + 12\times3 + 16\times3 + 9\times4 + 5\times4$
$= 224$ bits

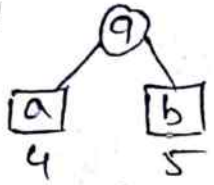
∴ Average code length $= \frac{224}{100} = 2.24$ bits/char

Scanned by CamScanner

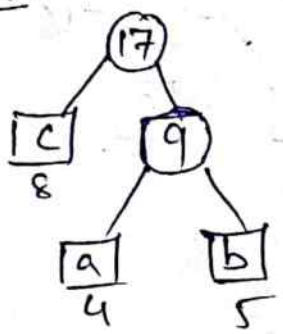
Q6 Suppose the six weights 4, 5, 8, 15, 16, 25. Find the Huffman tree and a minimum weighted path length P.

Soln: Let $a=4$, $b=5$, $c=8$, $d=15$, $e=16$, $f=25$

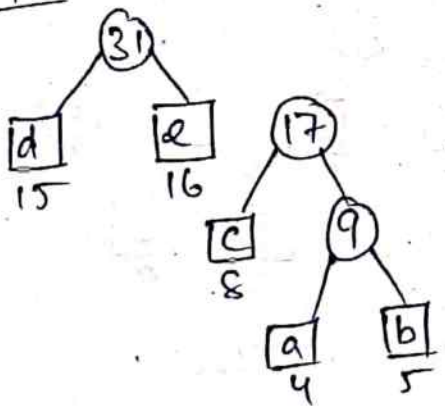
step(1)



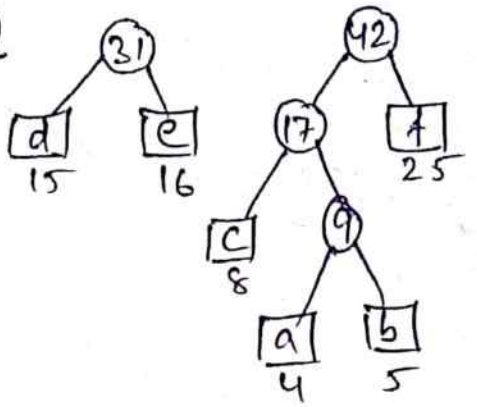
step(2)



step(3)



step(4)



step(5)

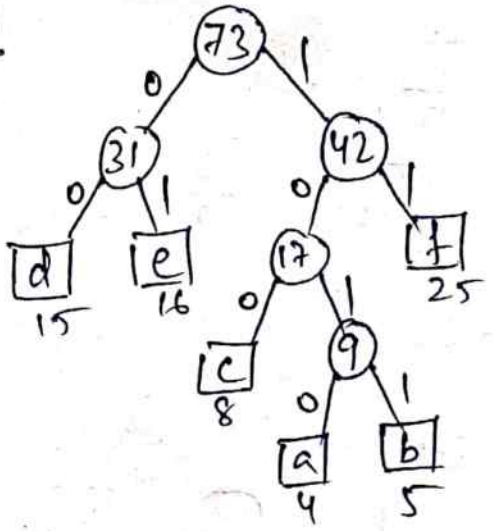


fig Huffman Tree

∴ Huffman codes are:
$a = 1010$
$b = 1011$
$c = 100$
$d = 00$
$e = 01$
$f = 11$

∴ Path length $P = 4\times4 + 5\times4 + 8\times3 + 15\times2 + 16\times2 + 25\times2$
$= 172$ bits

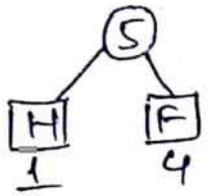
∴ Averge bits required $= \frac{172}{73} = 2.356$ bits / char

Scanned by CamScanner

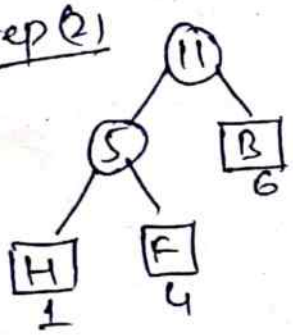
Q7 Message, M: A=15, B=6, C=7, D=12, E=25, F=4
G=6, H=1, I=15.

Decode : 1110100010111011

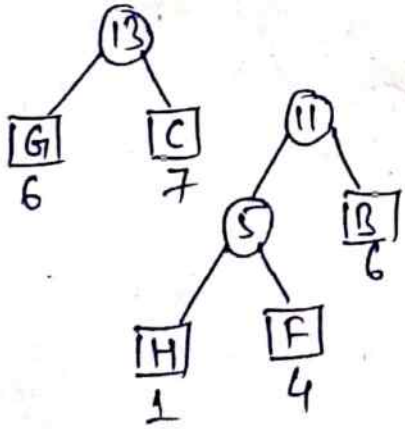
Soln step(1)



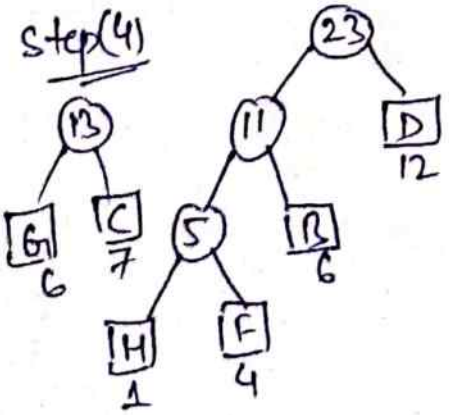
step(2)



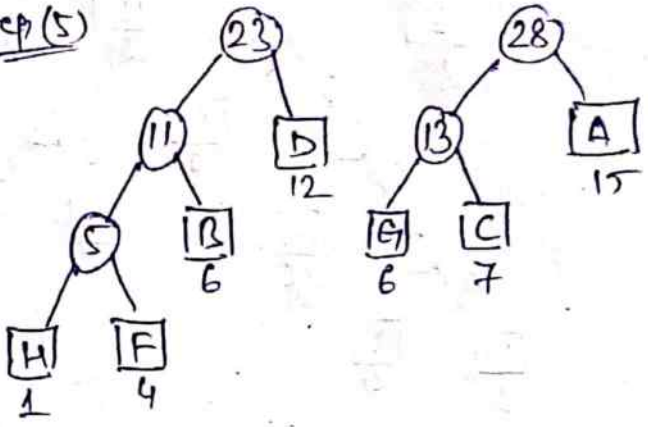
step(3)



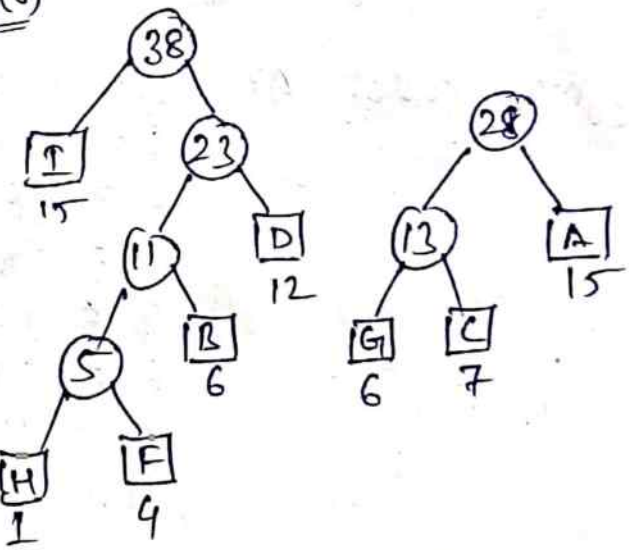
step(4)



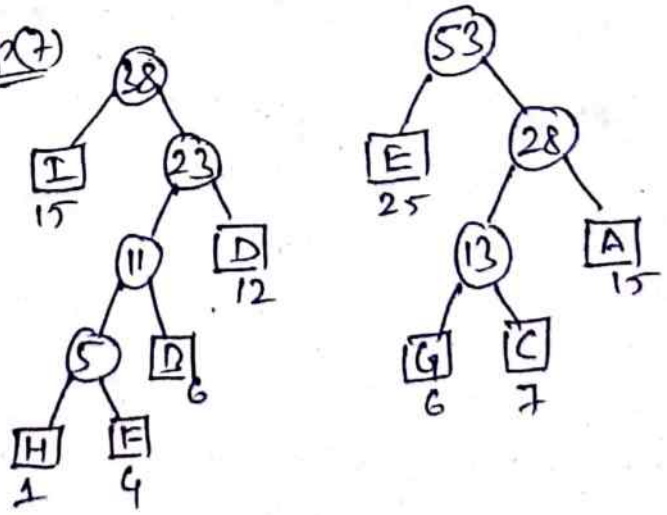
step(5)



step(6)



step(7)



Scanned by CamScanner

Step(8)

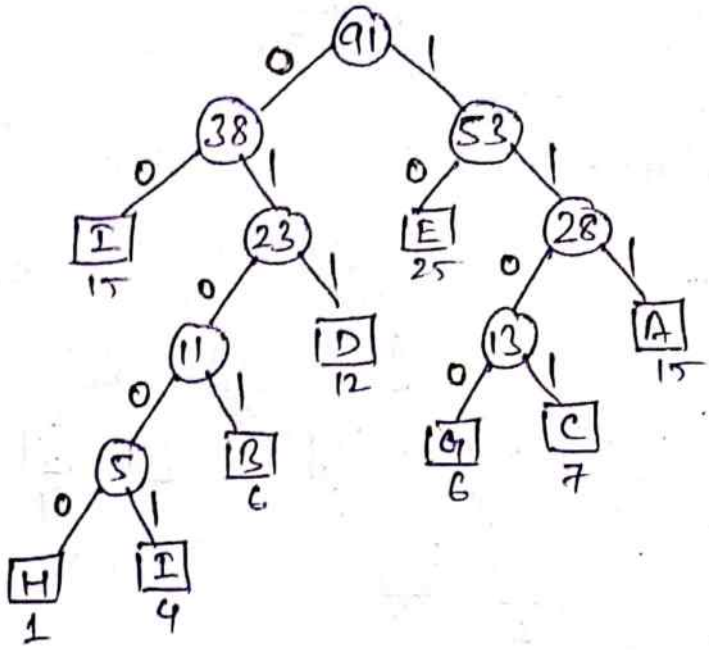


fig. Huffman Tree

Huffman codes are A = 111
B = 0101
C = 1101
D = 011
E = 10
F = 01001
G = 1100
H = 01000
I = 00

∴ Decode: $\underbrace{111}_{A}\ \underbrace{01000}_{H}\ \underbrace{10}_{E}\ \underbrace{111}_{A}\ \underbrace{011}_{D}$

Scanned by CamScanner

Algorithm for Huffman Coding.

no. of characters

Huffman (C)

1. $n = |C|$
2. $Q = C$ /* Q is a priority Queue */
3. for $i = 1$ to $n-1$
4. allocate a new node z
5. z.left = x = Extract-min (Q)
6. z.right = y = Extract-min (Q)
7. z.freq = x.freq + y.freq
8. Insert (Q, z)
9. return Extract-min (Q)

Scanned by CamScanner