Unit - IV

4 Trees

4.1 : Trees and Binary Trees Concept and Terminology

Q.1 Define the term - Tree.

Ans. : A tree is a finite set of one or more nodes such that -

i) Root node : It is a specially designated node.

ii) There are remaining n nodes which can be partitioned into disjoint sets T_1, T_2, T_3,... T_n where T_1, T_2, T_3,... T_n are called subtrees of the root.

- The concept of tree can be represented shown in Fig. Q.1.1.

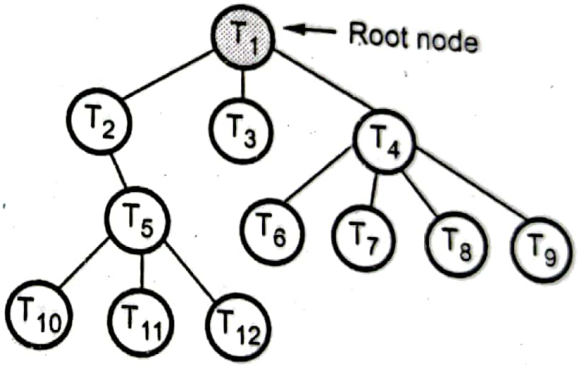


Fig. Q.1.1 Tree

Q.2 Explain the terms - i) Root ii) Sibling iii) Leaves.

Ans. : i) Root : Root is a unique node in the tree to which further subtrees are attached.

ii) Sibling : The nodes with common parent are called as siblings or brother.

iii) Leaves : These are the terminal nodes having no child nodes.

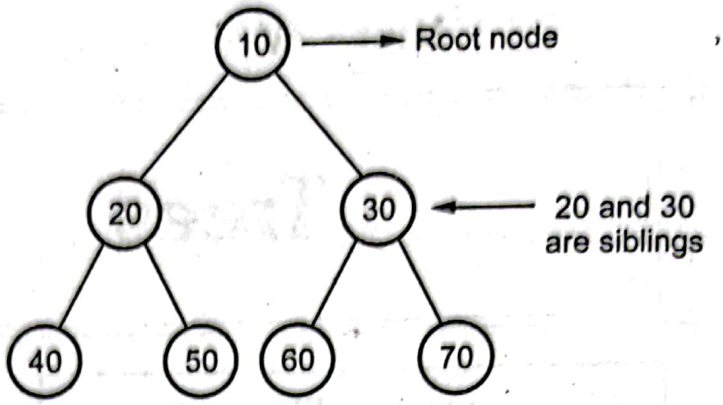


Fig. Q.2.1 Binary tree

Q.3 With examples define the following terms with respect to trees .
i) Skewed binary tree ii) Complete binary tree
iii) Ancestor and descendant leaf node. ☞**[SPPU : Dec.-10, Marks 6]**

Ans. : i) Skewed Binary Tree : The tree in which each node is attached as a left child of parent node then it is **left skewed tree.** The tree in which each node is attached as a right child of parent node then it is called **right skewed tree.**

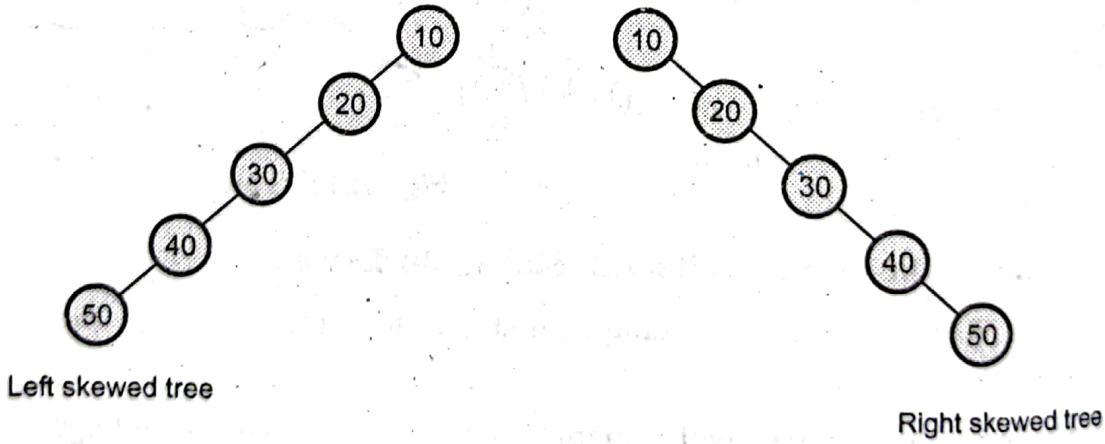


Fig. Q.3.1 Skewed trees

ii) Complete Binary Tree : A complete binary tree is full binary tree in which all leaves are at the same depth.

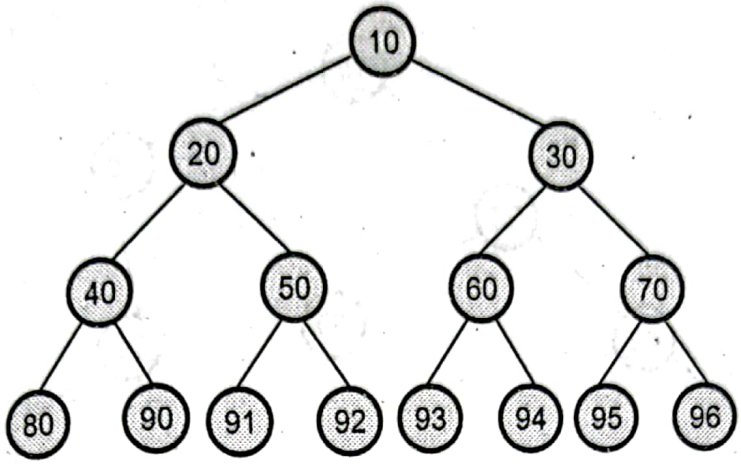


Fig. Q.3.2 Complete binary tree

iii) Ancestor and Descendant Leaf Node :

Predecessor : While displaying the tree, if some particular node occurs previous to some other node then that node is called **predecessor** or ancestor of the other node.

For example : While displaying the tree in Fig. Q.3.3 if we read node 20 first and then if we read node 40, then 20 is a predecessor of 40.

Successor : Successor or descendant is a node which occurs next to some node.

For example : While displaying tree in Fig. Q.3.3 if we read node 60 after reading node 20 then 60 is called successor of 20.

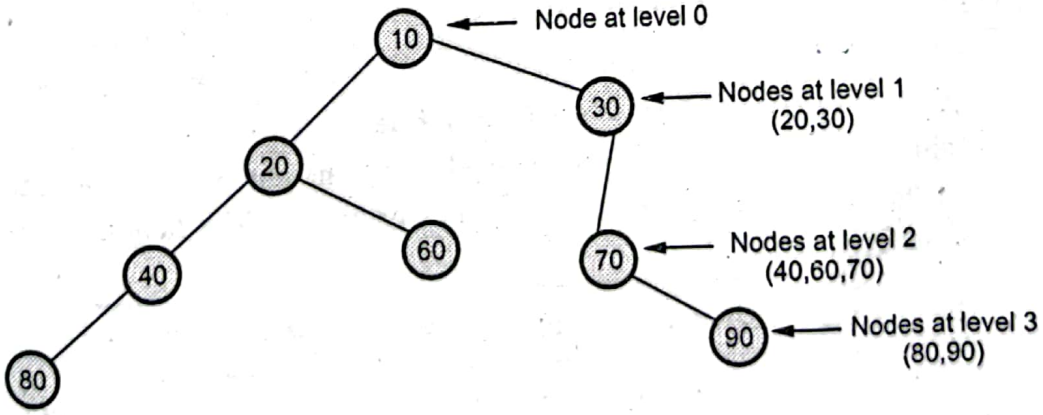


Fig. Q.3.3 Binary tree in which levels are shown

Q.4 Define the term - i) Root node ii) Leaf nodes iii) Degree of the node iv) Height of the tree.

Ans. :

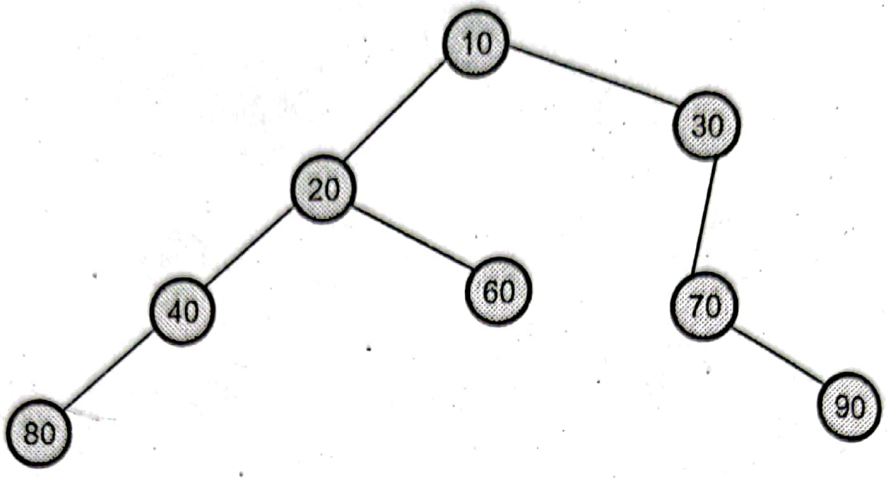


Fig. Q.4.1 Binary tree

i) **Root Node :** Root is a unique node in the tree to which further subtrees are attached. In above tree, node 10 is a root node.

ii) **Leaf Node :** The terminal nodes of the tree are called leaf nodes. In above tree, the 80, 60 and 90 are leaf nodes.

iii) **Degree of Node :** The total number of sub-trees attached to that node is called the degree of a node.

For example

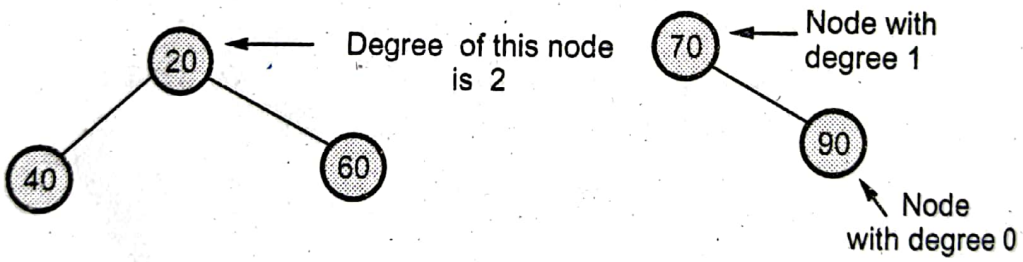


Fig. Q.4.2 Binary tree

iv) Height of Tree : The maximum level is the height of the tree. The height of the tree is called depth of tree. For example - The height of the tree in Fig. Q.4.1 is 3.

4.2 : Conversion of General Tree into Binary Tree

Q.5 What are the rules used for conversion of general tree into binary Tree ?

Ans. : i) The root of general tree becomes the root of the binary tree.

ii) Find the first child node of the node attach it as a left child to the current node in binary tree.

iii) The right sibling can be attached as a right child of that node.

Q.6 Convert the following generalized tree into a binay tree.

[SPPU : May-16, Marks 4]

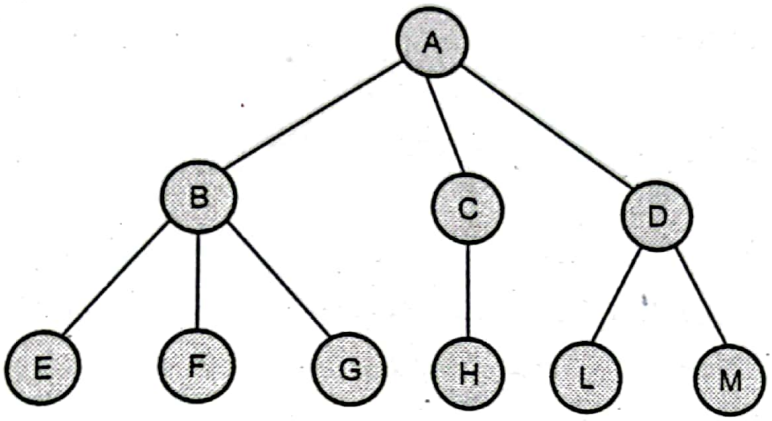


Fig. Q.6.1

Ans. : For converting general tree into binary tree following rules are applied -

i) Root of general tree is a root of binary tree.

ii) The first left child of general tree is a left child of binary tree.

iii) The right sibling can be attached as a right child of that node.

The converted binary tree is shown in Fig. Q.6.2

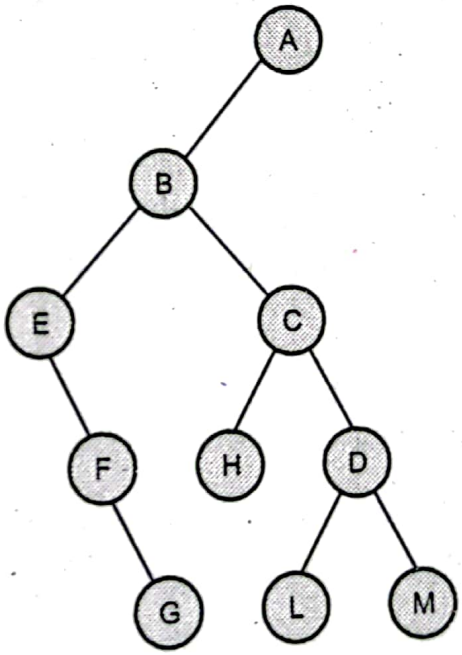


Fig. Q.6.2

4.3 : Binary Tree as an ADT

Q.7 Write an ADT for Binary Tree.

Ans. : The ADT for binary tree is

AbstractDataType Btree

{

Instances : Binary tree is a rooted tree having at the most two child nodes.

Operations :

1. **Create :** This operation is for creation of binary tree.
2. **Display :** This operation is used to display each node in the tree.
3. **Insert :** Using this operation the node can be inserted by attaching it as left or right child.
4. **Delete :** The node having left child or right child or both can be deleted.

}

Q.8 List down the steps to convert general tree to binary tree ? Convert the given general tree to binary tree -

☞[SPPU : Dec.-14, Marks 6]

Ans.: Refer Q.5.

The equivalent binary tree will be

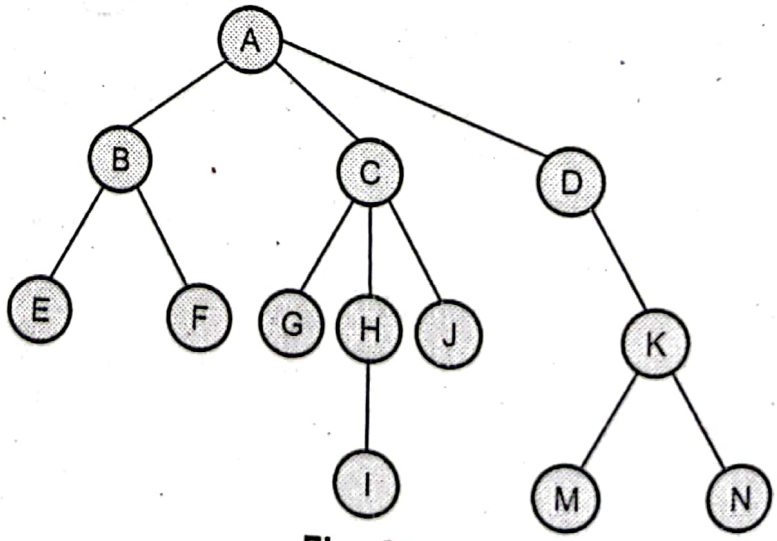


Fig. Q.8.1

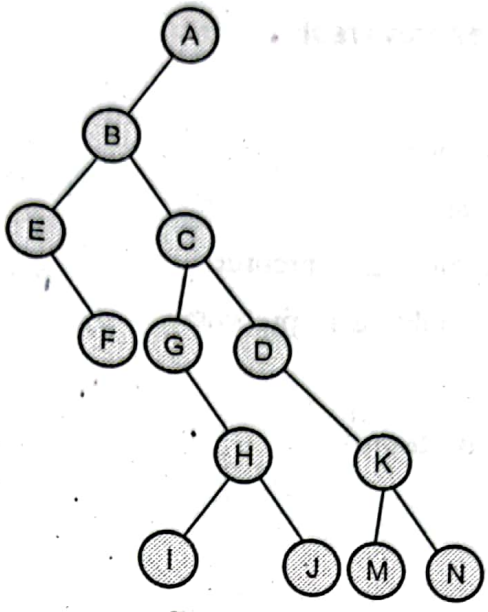


Fig. Q.8.2

4.4 : Binary Tree Traversals

Q.9 Write recursive algorithms for binary tree traversal.

Ans. : 1. Recursive inorder traversal

Algorithm :

1. If tree is not empty then
   a. traverse the left subtree in inorder
   b. visit the root node
   c. traverse the right subtree in inorder

C Function

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

2. Recursive preorder traversal

Algorithm :

1. If tree is not empty then
   a. visit the root node
   b. traverse the left subtree in preorder
   c. traverse the right subtree in preorder

C Function

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

3. Recursive postorder traversal

Algorithm :

1. If tree is not empty then
   a. traverse the left subtree in postorder
   b. traverse the right subtree in postorder
   c. visit the root node

C function

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

Q.10 Write functions for non recursive in order and preorder traversal.

☞[SPPU : May-11, 13, Marks 8]

Ans. : Non Recursive Inorder :

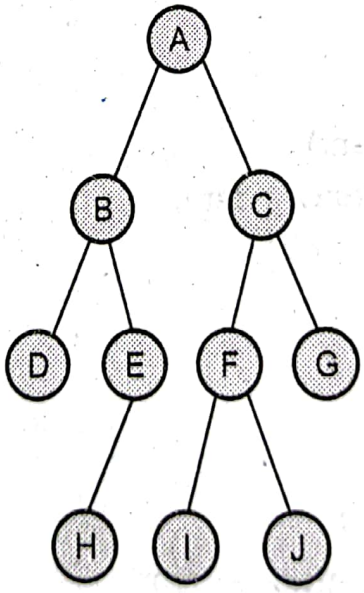
```
void TREE_CLASS::inorder(node *root)
{
node *current,*s[10];
int top=-1;
if(root==NULL)
{
   cout<<"\n Tree is empty\n";
   return;
}
current=root;
for(;;)
{
   while(current!=NULL)
   {
      push(current,&top,s);
      current=current->left;
   }
   if(!stempty(top))
   {
      pop(&top,s,&current);
      cout<<" "<<current->data;
      current=current->right;
   }
   else
      return;
}
```

Non Recursive Preorder :

```
void TREE_CLASS::preorder(node *root)
{
node *current,*s[10];
int top=-1;
if(root==NULL)
{
  cout<<"\n The Tree is empty\n";
```

```
    return;
}
current=root;
for(;;)
{
    while(current!=NULL)
    {
     cout<<" "<<current->data;
     push(current,&top,s);
     current=current->left;
    }
    if(!stempty(top))
    {
     pop(&top,s,&current);
     current=current->right;
    }
   else
    return;
}
}
```

Q.11 Traverse a given binary Tree in Inorder, Preorder and Postorder : ☞[SPPU : May-19, Marks 6]



Ans. : Preorder : A, B, D, E, H, C, F, I, J, G

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

Postorder : D, H, E, B, I, J, F, G, C, A

Q.12 Write an algorithm for preorder traversal of binary tree and give suitable example. ☞[SPPU : May-18, Dec.-18, Marks 6]

Ans. : Refer Q.9 and Q.11.

4.5 : Binary Search Trees

Q.13 Explain the concept of binary search tree.

Ans. : The binary search tree is based on the binary search algorithm. While creating the binary search tree the data is systematically arranged. That means values at left subtree < root node value < Right subtree values.

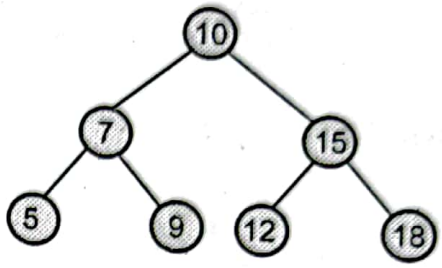


Fig. Q.13.1 Binary search tree

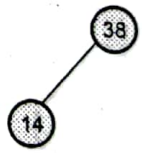
Q.14 Define binary search tree. Draw the BST for given nodes : 38,14,56,23,82,8,45,70,18,15. ☞[SPPU : Dec.-14, May-17, Marks 4]

Ans. : Binary search tree - Refer Q.13.

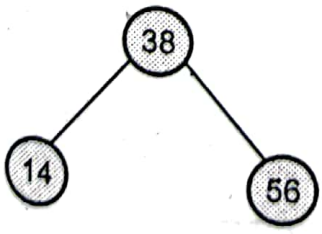
Example :

Step 1 : Insert 38

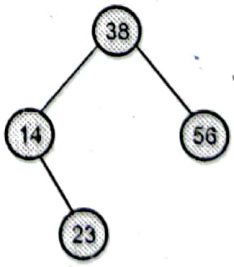
Step 2 : Insert 14



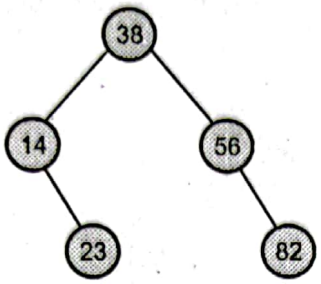
Step 3 : Insert 56



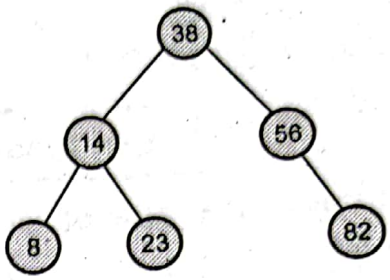
Step 4 : Insert 23



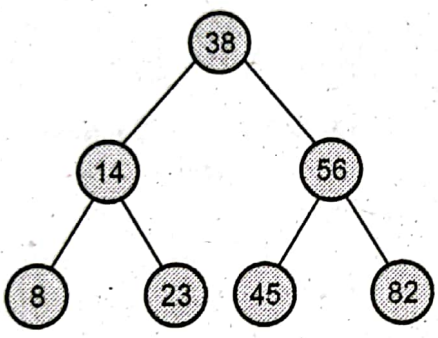
Step 5 : Insert 82



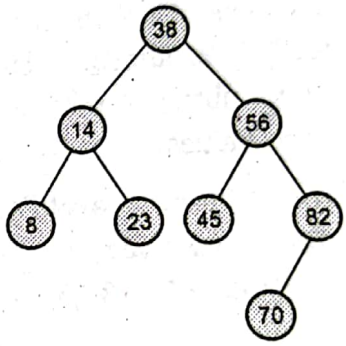
Step 6 : Insert 8



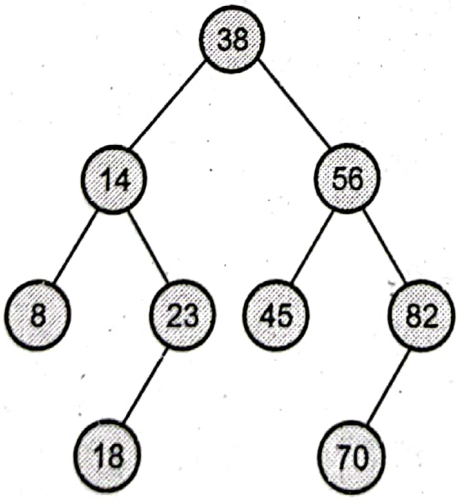
Step 7 : Insert 45



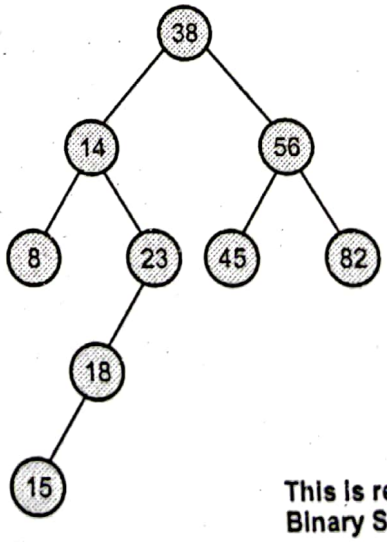
Step 8 : Insert 70



Step 9 : Insert 18



Step 10 : Insert 15



This is required Binary Search Tree

4.6 : Binary Search Tree as ADT

Q.15 Write a function for creating binary search tree.

☞[SPPU : Dec.-12, Marks 4]

Ans. :

```
void bintree::create()
{
   New=new node;
   New->left=NULL;
   New->right=NULL;
   cout<<"\n Enter The Element ";
   cin>>New->data;
   if(root==NULL)
      root=New;
   else
      insert(root,New);
}
```

Q.16 Write an algorithm to delete a node from binary search tree.

☞[SPPU : May-10,11, Marks 6, June-22, Marks 8]

Ans. :

```
void bintree::del(node *root,int key)
{
node *temp_succ;
 if(root==NULL)
  cout<<"Tree is not Created!";
 else
 {
  temp=root;
  search(&temp,key,&parent);
//temp node is to be deleted
/*deleting a node with two children*/
if(temp->left!=NULL && temp->right!=NULL)
 {
  parent=temp;
  temp_succ=temp->right;
  while(temp_succ->left!=NULL)
  {
   parent=temp_succ;
```

```
  temp_succ=temp_succ->left;
 }
 temp->data=temp_succ->data;
        //copying the immediate successor
 temp->right=NULL;
 cout<<" Now Deleted it!";
 return;
}
/*deleting a node having only one child*/
/*The node to be deleted has left child*/
if(temp->left!=NULL && temp->right==NULL)
{
 if(parent->left==temp)
   parent->left=temp->left;
 else
   parent->right=temp->left;
 temp=NULL;
 delete temp;
 cout<<" Now Deleted it!";
 return;
}
/*The node to be deleted has right child*/
if(temp->left==NULL && temp->right!=NULL)
{
 if(parent->left==temp)
   parent->left=temp->right;
 else
   parent->right=temp->right;
 temp=NULL;
 delete temp;
   cout<<" Now Deleted it!";
 return;
}
/*deleting a node which is having no child*/
if(temp->left==NULL && temp->right==NULL)
```

```
    {
      if(parent->left==temp)
       parent->left=NULL;
      else
       parent->right=NULL;
     cout<<" Now Deleted it!";
     return;
    }
   }
   }
```

Q.17 What is binary search tree ? Write pseudo code for deletion and insertion of a node in binary tree. ☞**[SPPU : May-13, Marks 8]**

Ans. : Binary Search Tree - Refer Q.13.

Insertion of a node -

```
void bintree::insert(node  *root,node *New)
{
   if(New->data<root->data)
    {
       if(root->left==NULL)
        root->left=New;
       else
        insert(root->left,New);
    }
   if(New->data>root->data)
   {
   if(root->right==NULL)
          root->right=New;
   else
          insert(root->right,New);
   }
}
```

Deletion of a node - Refer Q.16.

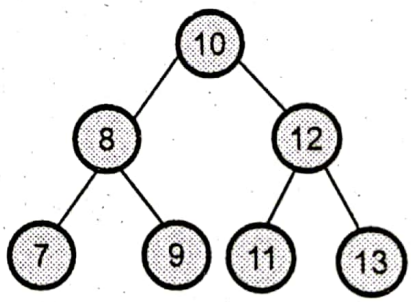
Q.18 Write a recursive algorithm to find the mirror image of binary tree and explain it with suitable example.

☞[SPPU : May-11, Marks 6]

Ans. : Recursive function to find mirror image

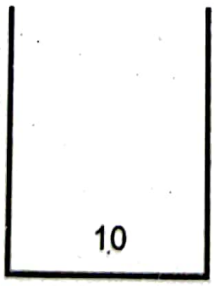
```
void tree : : mirror (btree*root)
{
btree * temp:
if (root ! = NULL)
{
    mirror(root → left);
    mirror(root → right);
    temp = root → left;
    root → left = root → right;
    root → right = temp ;
}
}
```

Stack contents - Assume the binary search tree as -



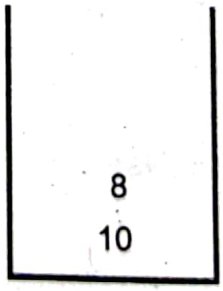
Binary search tree

Step 1 :



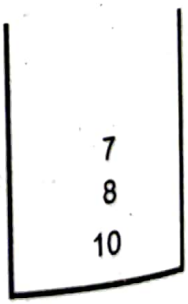
Stack

Step 2 :



Stack

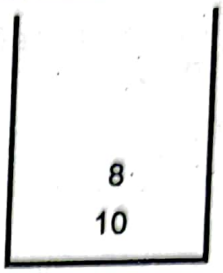
Step 3 :



Stack

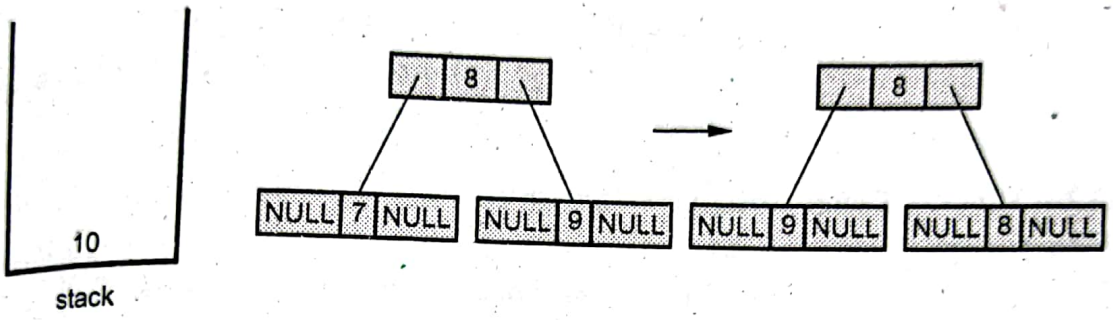
Step 4 :

The stack will be popped, node 7 has no left and right child.

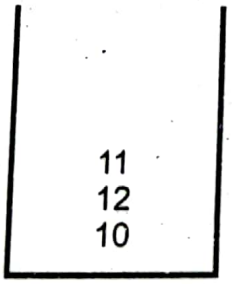


Stack

Step 5 : The stack will be popped. Then the left and right child of the node 8 will be swapped.

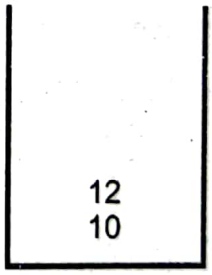


Step 6 :



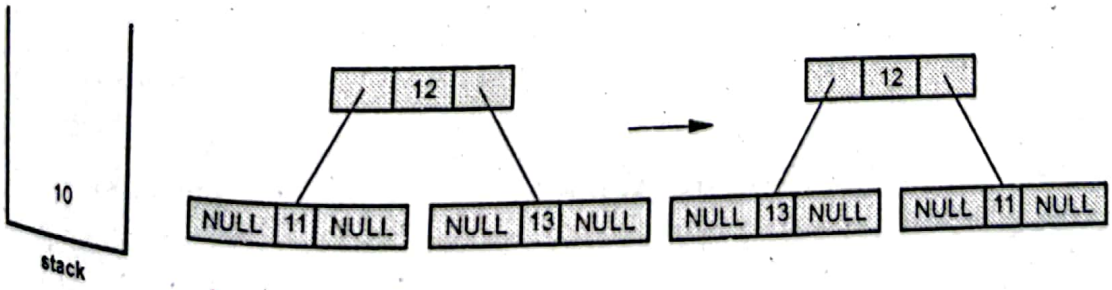
Stack

Step 7 : The stack will be popped, node 11 has no left and right child

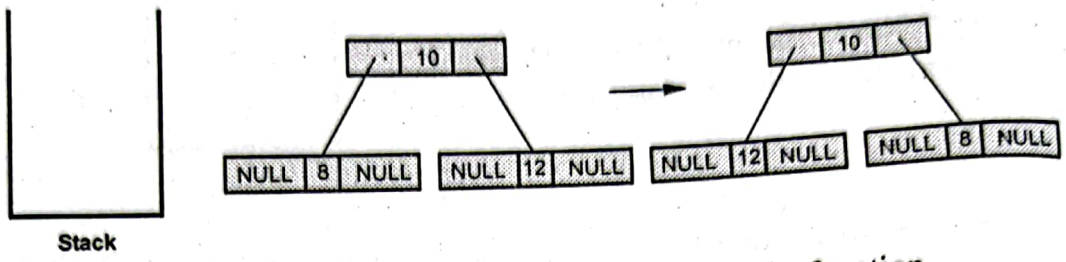


Stack

Step 8 : The stack will be popped. Then left and right child of the node 12 will be swapped.



Step 9 : The stack will be popped. Then the left and right child of node 10 will be popped.



As now stack is empty, the control will return to main function.

Q.19 Write an algorithm to delete a node from binary search tree.

[SPPU : May-10,11, Marks 6]

Ans. : Refer Q.16.

Q.20 Write efficient functions that takes only a pointer to the root of binary tree T, and compute :
1) The number of nodes in T 2) The number of leaves in T.
What is running time of your algorithm ?

Ans. : 1) The number of nodes in T

There are two function written -

count_nodes function has a logic for visiting each node of the tree and counting them.

display function displays the actual count value

```
/*
------------------------------------------------------------------------
Function which counts the total number of nodes in the tree
------------------------------------------------------------------------
*/
void tree::count_nodes(btree *temp,int *count)
{
 if(root==NULL)
   cout<"TREE IS NOT CREATED";
if(temp!=NULL)
{
```

```
 count_nodes(temp->left,count);
 *count++;
 count_nodes(temp->right,count);
 }
}
/*
-----------------------------------------------------------------
The calling function - It calls the function for counting total number of
nodes
-----------------------------------------------------------------
*/

void tree::display()
{
 int *count=0;
 count_nodes(root,count);
 cout<<"Total nodes= "<<*count;
}
```

2) The number of leaves in T

There are two function written –

count_leaves function has a logic for visiting each leaf node of the tree and counting them.

display function displays the actual count value for the leaves

```
/*
-----------------------------------------------------------------
The count_leaves Function
-----------------------------------------------------------------
*/
void tree::count_leaves(btree *temp,int *count)
{
 if(root==NULL)
   cout<<"TREE IS NOT CREATED";
if(temp!=NULL)
{
 if((temp->left==NULL)&&(temp->right==NULL))
```

```
    *count=*count+1;
 else
 {
  count_leaves(temp->left,count);
  count_leaves(temp->right,count);
 }
}
}
/*
-----------------------------------------------------------------
 The display function
-----------------------------------------------------------------
*/

void tree::display()
{
  int *count;
  *count=0;
  count_leaves(root,count);
  cout<<"Total nodes= "<<*count;
}
```

Q.21 Write a pseudo C routine to find the depth of the binary tree.

☞[SPPU : Dec.-06, Marks 4]

Ans. :

```
int tree::Height(btree *temp)
{
 int d1,d2;
 if(temp==NULL)
   return 0;
 if(temp->left==NULL && temp->right==NULL)
   return 0;
 d1=Height(temp->left);//computing height of leftsubtree
 d2=Height(temp->right);//computing height of rightsubtree
  if(d1>d2)
```

```
  return d1+1;
 else
  return d2+1;
//maximum depth=height of the tree
}
/*
-----------------------------------------------------------------------
   The display function
-----------------------------------------------------------------------
*/

void tree::display()
{
 cout<<"The height of the tree is "<<Height(root);
}
/*
```

Q.22 Create a binary tree from the following sequence.
Postorder : H I D E B F G C A
Inorder : H D I B E A F C G

☞[SPPU : May-17, Marks 6]

Ans. : Step 1 : The last node in postorder sequence is the root node. In above example "A" is the root node. Now observe inorder sequence locate the "A". Left sequence of "A" indicates the left subtree and right sequence of "A" indicates the right sub-tree.

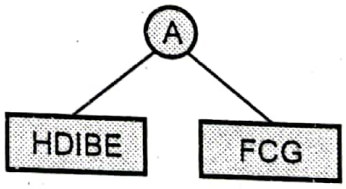


Fig. Q.22.1

i.e. as shown in Fig. Q.22.1.

Step 2 : Now, with these alphabets H, D, I, B, E observe the postorder and sequences.

Postorder H I D E B

Inorder H D I B E

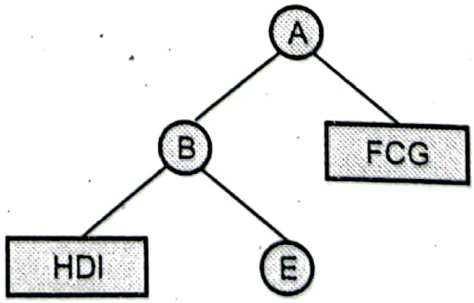


Fig. Q.22.2

Here B is parent node, therefore pictorically tree will be,

Step 3 : With the alphabets H, D and I observe both the sequences.

Postorder H I D

Inorder H D I

D is the parent node, H is leftmost node and I is the right child of D node. So tree will be as shown in Fig. Q.22.3.

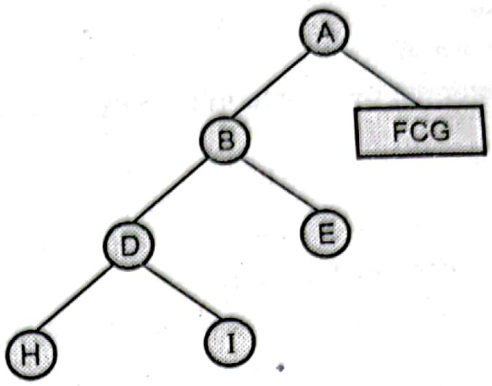


Fig. Q.22.3

Step 4 : Now we will slove for right sub-tree of root "A". With the alphabets F, C, G observe both the sequences

Postorder F G C

Inorder F C G

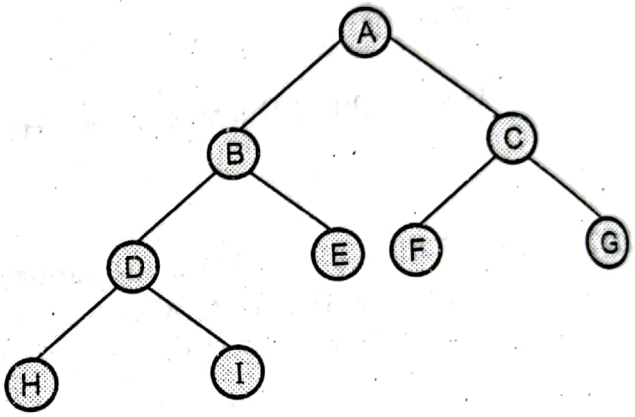


Fig. Q.22.4

'C' is the parent node, F is the left child and G is the right child. So finally the tree will be as shown in Fig. Q.22.4.

Q.23 Construct binary tree using tree traversals given below :
Preorder traversal : P A Q B R S D E F
Inorder traversal : A P B Q D S E R F

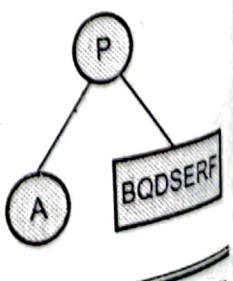
☞ **[SPPU : May-16, Marks 4]**

Ans. : The first node in Preorder traversal is root node.

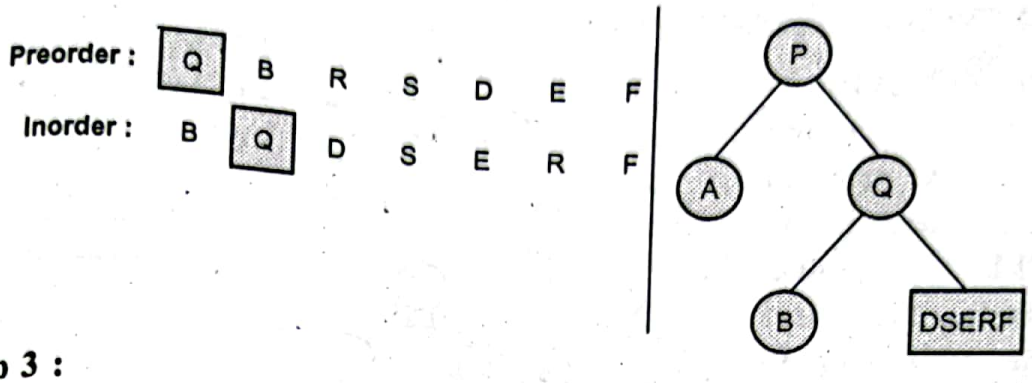
Step 1 :

Preorder : P A Q B R S D E F

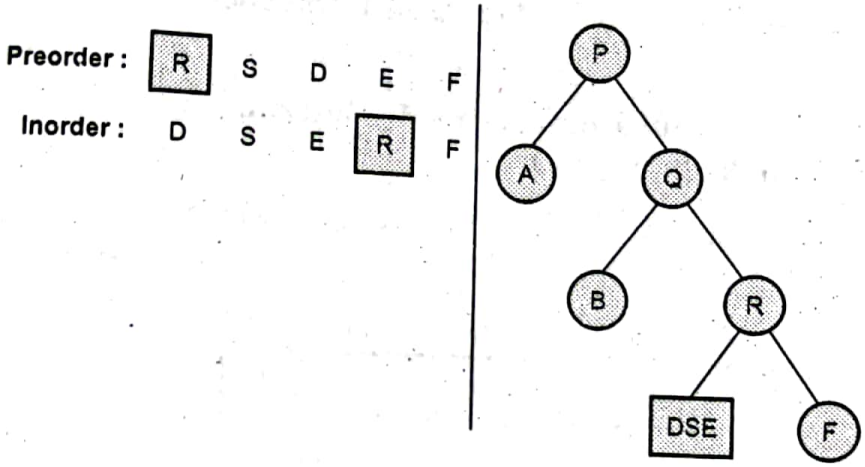
Inorder : A P B Q D S E R F



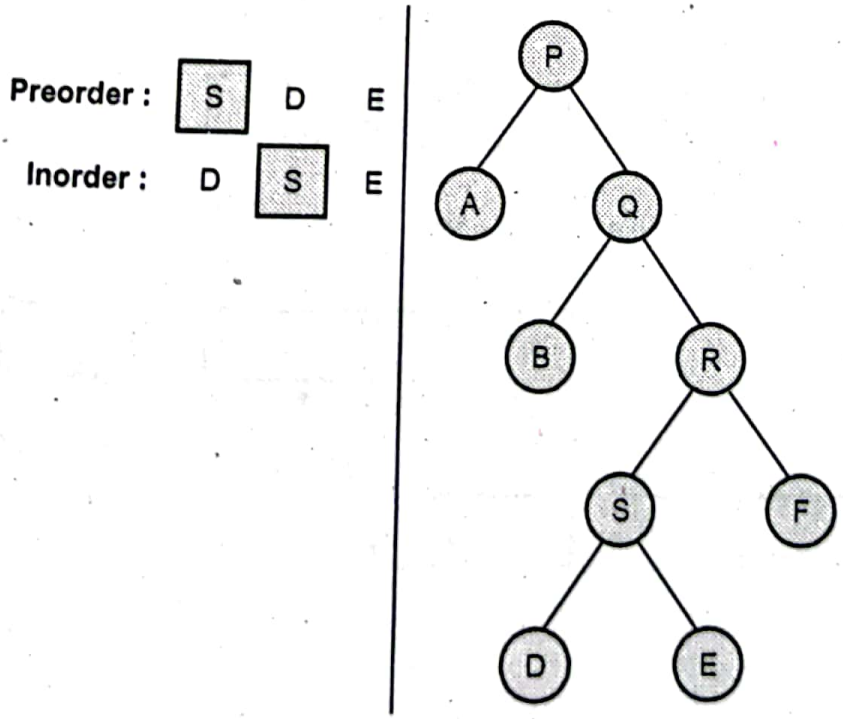
Step 2 :



Step 3 :



Step 4 :



Binary tree

4.7 : Threaded Binary Tree

Q.24 What is threaded binary tree ?

Ans. : Threaded binary tree is a kind of binary tree in which there is no NULL value. In-fact the leaf nodes will point to their predecessors and successors. For example

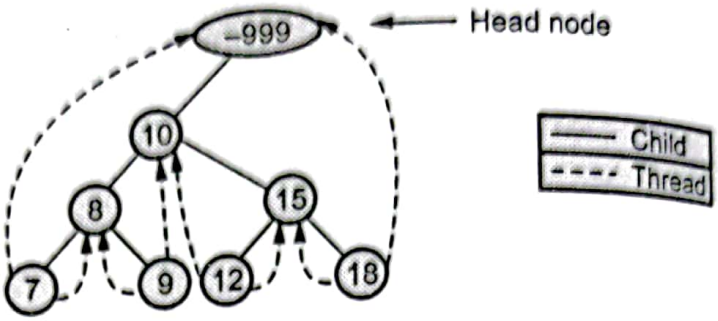
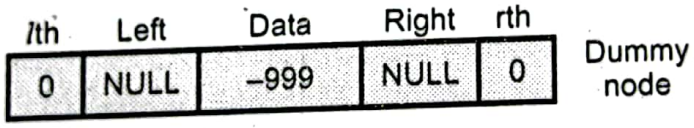


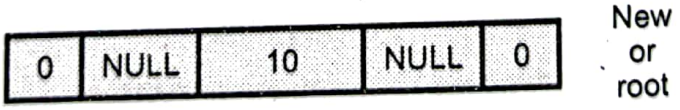
Fig. Q.24.1 Threaded binary tree

Q.25 Create a threaded binary tree by inserting the following values one by one - 10,8,6,12,9,11,14

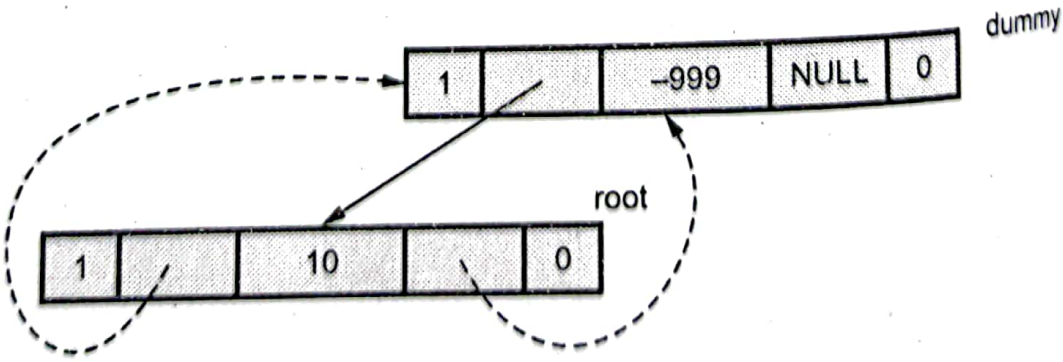
Ans. : Initially we will create a dummy node which will act as a header of the tree.



Now let us take first value i.e. 10. The node with value 10 will be the root node we will attach this root node as left child of dummy node.



The NULL links of root's left and right child will be pointed to dummy.



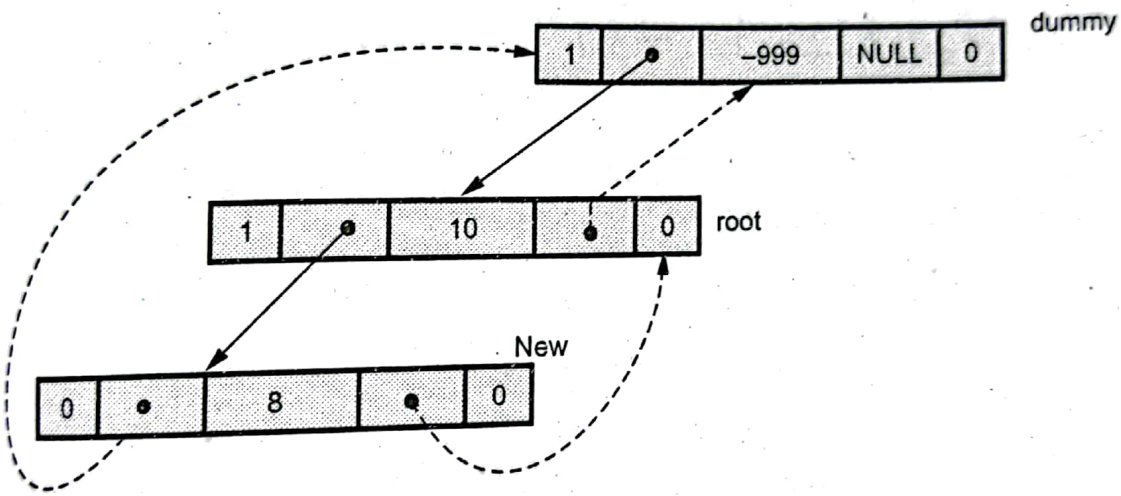
Now next comes 8. The 8 will be compared with 10, as the value is less than 10, we will attach 8 as left child of 10 and we will set root's lth field to 1, indicating that the node 10 is having left child. See then how the links are arranged.

New → left = root → left

New → right = root;

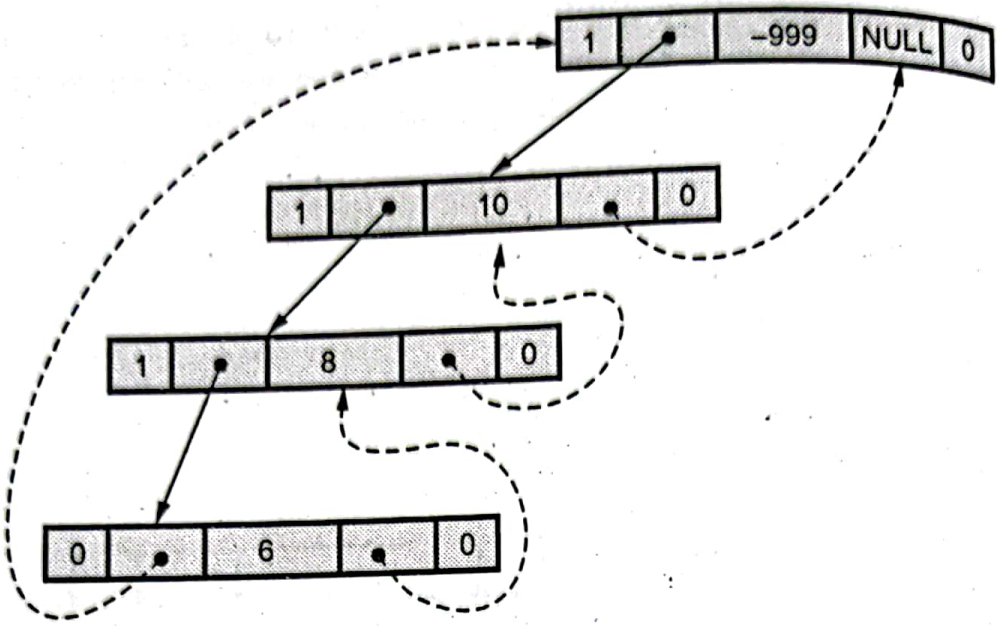
root → left = New;

root → lth = 1;

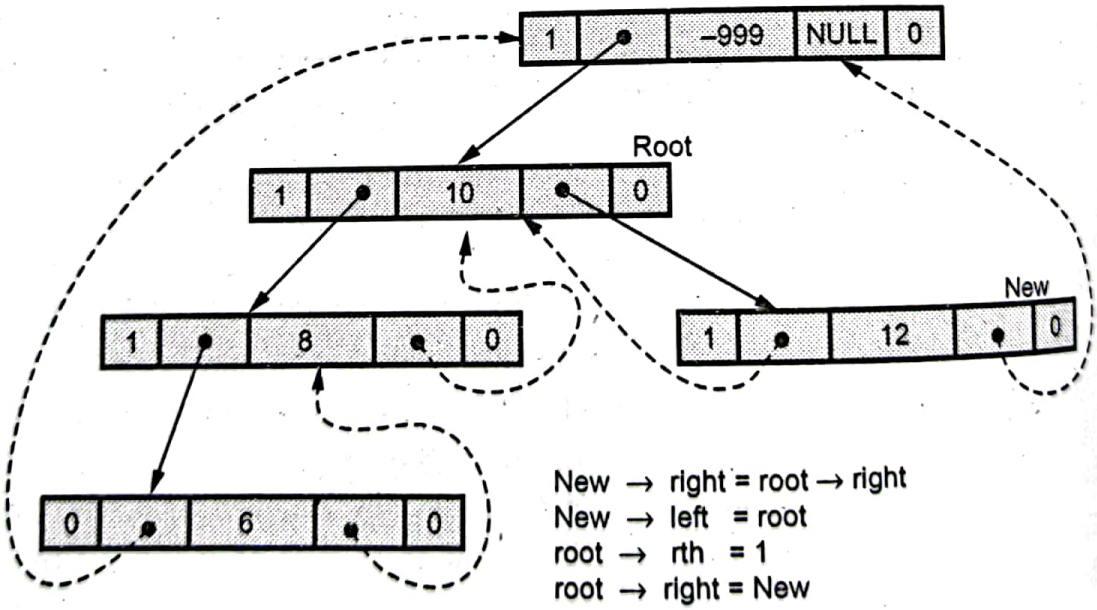


Note that the left link of node 8 points to its inorder predecessor and right link of node 8 points to its inorder successor.

Next comes 6. The value 6 will be first compared with root i.e. with 10. As 6 is less than 10, we will move on left sub-branch. The 6 will then be compared with 8, so we have to move on left sub-branched of 8. But lth of node 8 is 0, indicating that there is no left child to 8, so we will attach 6 as left child to 8.



Then comes 12. We will compare 12 with 10. As 12 is greater than 10, we will attach the node 12 as right child of 10.



Note that the left field of node 12 points to its inorder predecessor and right field of node 12 points to its inorder sussessor. Thus by attaching 9, 11, 14 as appropriate children, the threaded binary tree will look like this.

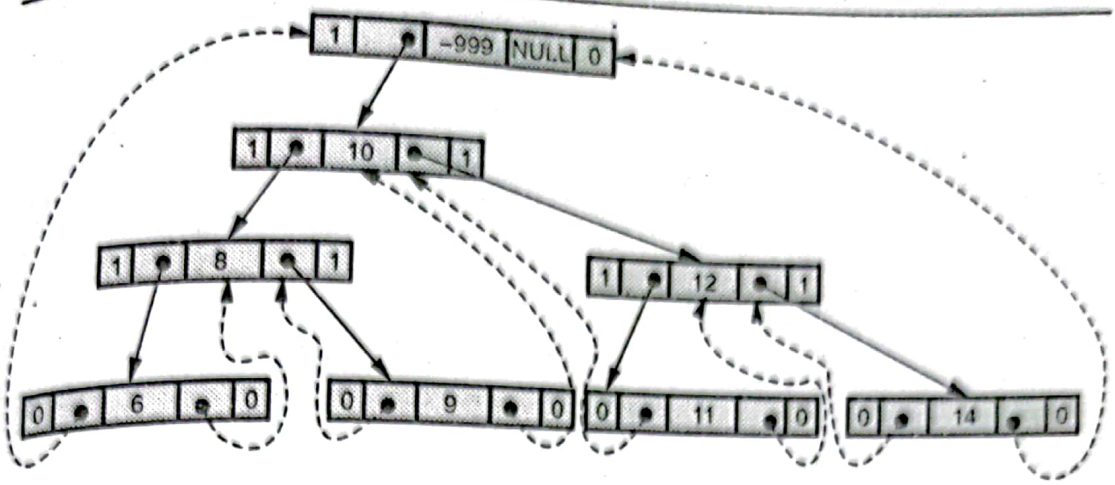


Fig. Q.25.1

Such a threaded tree is called inorder treaded binary tree.

Q.26 Give advantages and disadvantages of threaded binary tree.

Ans. : Advantages of threaded binary tree :

1. In threaded binary tree there is no NULL pointer present. Hence memory wastage in occupying NULL links is avoided.
2. The threads are pointing to successor and predecessor nodes. This makes us to obtain predecessor and successor node of any node quickly.
3. There is no need of stack while traversing the tree, because using thread links we can reach to previously visited nodes.

Disadvantages of threaded binary tree :

1. Implementing threads for every possible node is complicated.

Q.27 Draw the threaded binary tree equivalent for the tree represented by the following array assuming root node of tree is stored at index 0 in the array. ☞ [SPPU : May-16, Marks 4]

Index	Data
0	10
1	–
2	20
3	–
4	–
5	30

6	–
7	–
8	–
9	–
10	–
11	–
12	40
13	–
14	–
15	–
16	–
17	–
18	–
19	–
20	–
21	–
22	–
23	–
24	–
25	50
26	–
27	–
28	–
29	–
30	–
31	–

Ans. : When n = 0, the root node will be placed at 0^{th} location. The nodes are stored in array using following formula

Parent (n) = floor (n – 1) / 2

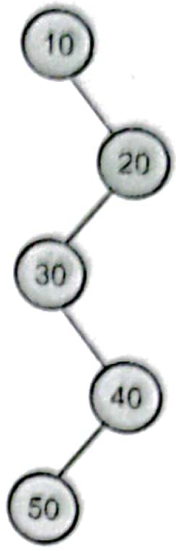
Leftchild (n) = 2n + 1

Rightchild (n) = 2n + 2

When n = 0

Rightchild (0) = 2 × 0 + 2 = 2. That means 20 is attached as right child of 10.

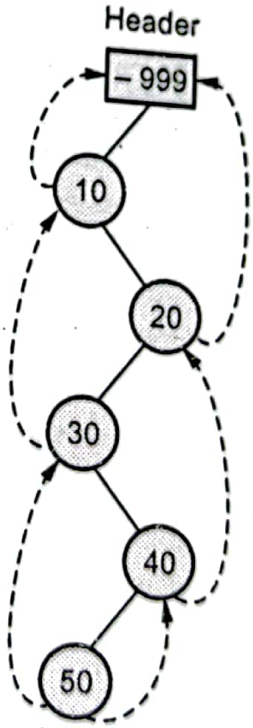
Let n = 2, Leftchild (2) = 2n + 1 = 2 × 2 + 1 = 5. At 5th index the data is 30. That means 30 is attached as leftchild of 20. In this way we can get the construction of binary tree as -



Binary tree

Fig. Q.27.1

The predecessor is pointed by left thread and successor of node is pointed by right thread. Thus the converted threaded binary tree will be -



Threaded binary tree

Fig. Q.27.2

Q.28 Write an algorithm for the inorder traversal of a threaded binary tree.

☞ [SPPU : Dec.-17, 6 Marks]

Ans. :

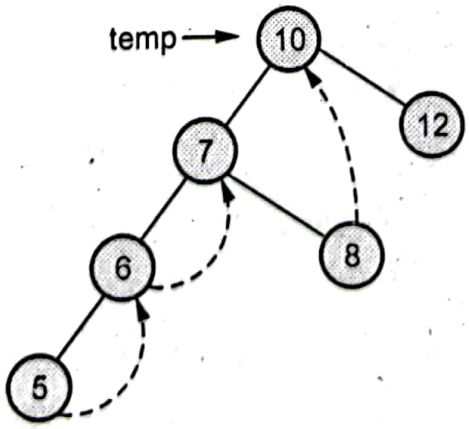
```
void inorder(node *temp,node *dummy)
{
  while(temp!=dummy)
  {
   while(temp->lth==1)
    temp=temp->left;
  cout<<" "<<temp->data;
  while(temp->rth==0)
  {
  temp=temp->right;
   if(temp==dummy)
    return;
   cout<<" "<<temp->data;
  }
  temp=temp->right;
 }
}
```

Q.29 What is TBT ? What is advancement in TBT over BT ? Draw any suitable in-ordered TBT and traverse it in pre-order traversal

☞ **[SPPU : Dec.-18, June-22, Marks 8]**

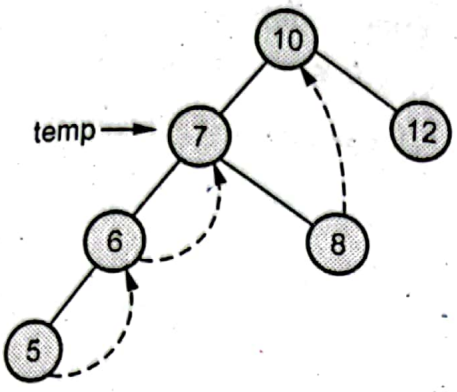
Ans. : Refer Q.24 and Q.25.

In preorder traversal we always visit the parent node, then left node and lastly the right node. In this manner we traverse the entire tree. Following example illustrates the preorder traversal of threaded binary tree.



10

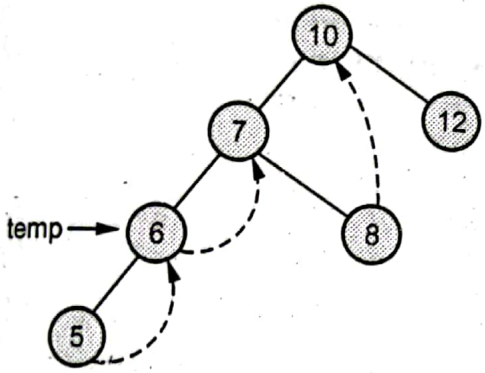
Print "temp -> data"



10 7

Print "temp -> data" if left subtree exists.

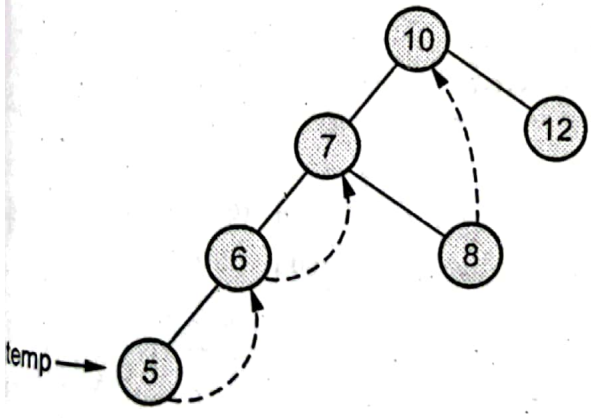
Make temp -> left as temp.



10 7 6

Print "temp -> data" if left subtree exists.

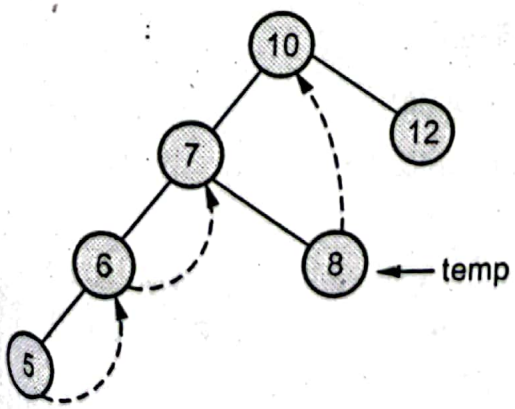
Make temp -> left as temp.



10 7 6 5

Print " temp -> data " If left and right subtree don't exists, and right thread exist, traverse up the right threads.

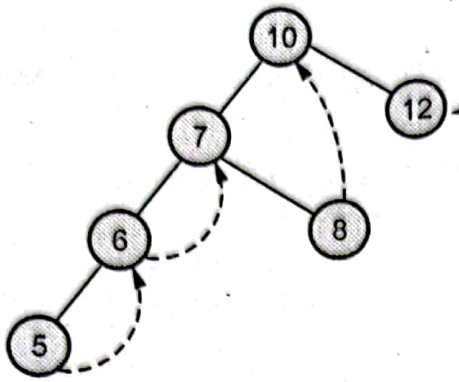
If you arrive at last node, then stop. Else make temp -> right as temp.



10 7 6 5 8

Print " temp -> data " If left and right subtree don't exists, and right thread exists, traverse up the right threads.

If you arrive at last node, then stop. Else make temp -> right as temp.



Print "temp -> data"

∴ 10 7 6 5 8 12

This is preorder travers.

4.8 : Applications of Trees

Q.30 what are applications of binary tree ?

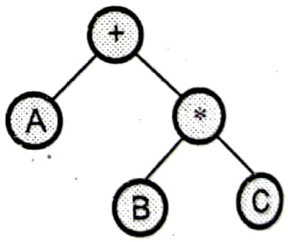
Ans.: Various applications of binary tree are

1. Decision Trees
2. Game Trees
3. Expression Trees

4.9 : Expression Tree

Q.31 What is expression tree ?

Ans.: An expression tree is a binary tree in which the operands are attached as leaf nodes and operators become the internal nodes.



For example

From expression tree :

Inorder traversal : A+B*C(Left-Data-Right)

Preorder traversal : +A*BC(Data-Left-Right)

Postorder traversal : ABC*+ (Left-Right-Data)

If we traverse the above tree in inorder, preorder or postorder then we get infix, prefix or postfix expressions respectively.

Q.32 Parenthesis are not given in an expression in prefix or postfix. Justify. Draw the expression tree and find the infix and postfix expressions for the following prefix expression.

***–AB + *CD/EF**

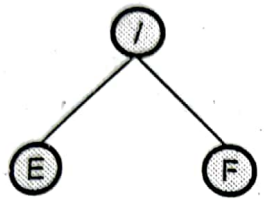
☞[SPPU : Dec.-11, Marks 8]

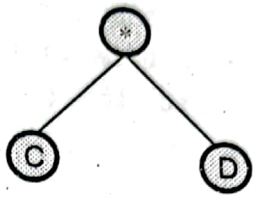
Ans. : Basically parenthesis indicate the operations which need to be carried out first i.e. according to the BOOMAS rule. So in case of postfix or prefix expression they are actually conversions of the original standard equation, where the brackets have already been taken into consideration and the formed prefix/postfix expression of a given mathematical statement.

Given expression

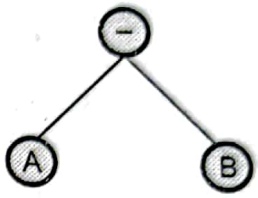
* – AB + * CD/EF

I)

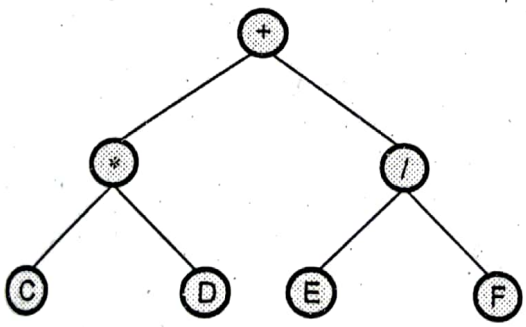




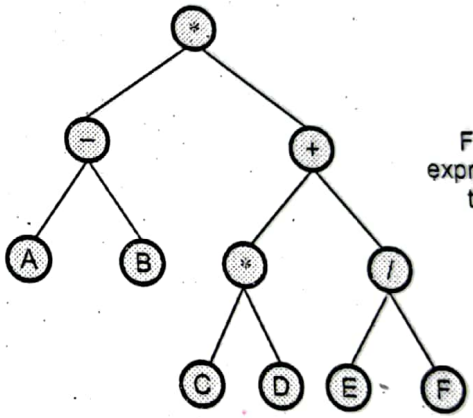
II)



III)



IV)



Final expression tree

Infix expression = (A – B) * [(C * D) + (E/F)]

Postfix expression = AB – CD * + EF / *

Q.33 Write an algorithm for evalution of an expression in expression Tree.

☞[SPPU : Dec.-17, Marks 4]

Ans. : Algorithm : 1) Traverse the expression tree in inorder manner and get the expression in an array.

2) Read the expression from left to right.

3) If the input symbol read is '(' then push it onto the stack.

4) If the input symbol read is an operand then place it in postfix expression.

5) If the input symbol read is an operator then,

   a) Check if the precedence of the operator which is in the stack has greater precedence than the precedence of the operator read, if so then remove that symbol from stack and place it in the postfix expression. Repeat step 5(a) till you get the operator in the stack has greater precedence than the operator being read.

   b) Otherwise push the operator being read onto the stack.

6) If the input symbol read is a closing parenthesis ')' then pop all the operators from the stack, place them in postfix expression till the opening parenthesis is not popped. The '(' should not be place in the postfix expression.

7) Finally print the postfix expression.

END...