Introduction

Tree is a **non-linear data structure** used to represent hierarchical structure of data. A tree consists of a finite set of elements, called **nodes**, and a finite set of directed lines, called **branches**, that connect the nodes. If the tree is not empty, then the first node is called the **root.**

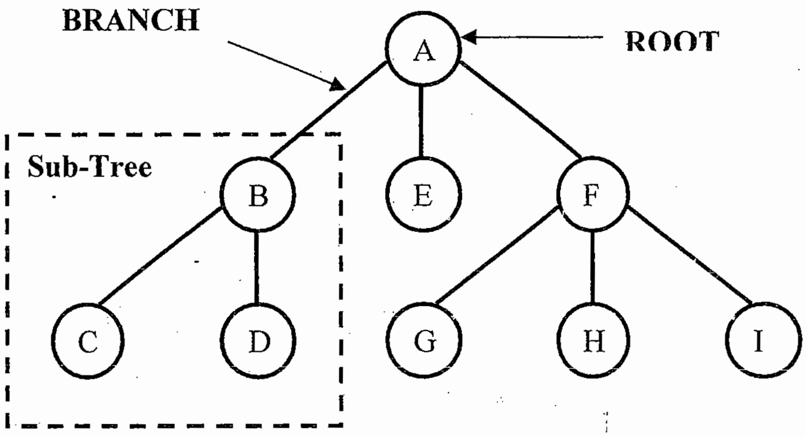
A tree may be divided into sub-trees. A sub-tree is any connected structure below the root. The first node in a sub-tree is known as the root of the sub-tree.

The concept of sub-tree leads to a **recursive definition** of a tree.

Definition

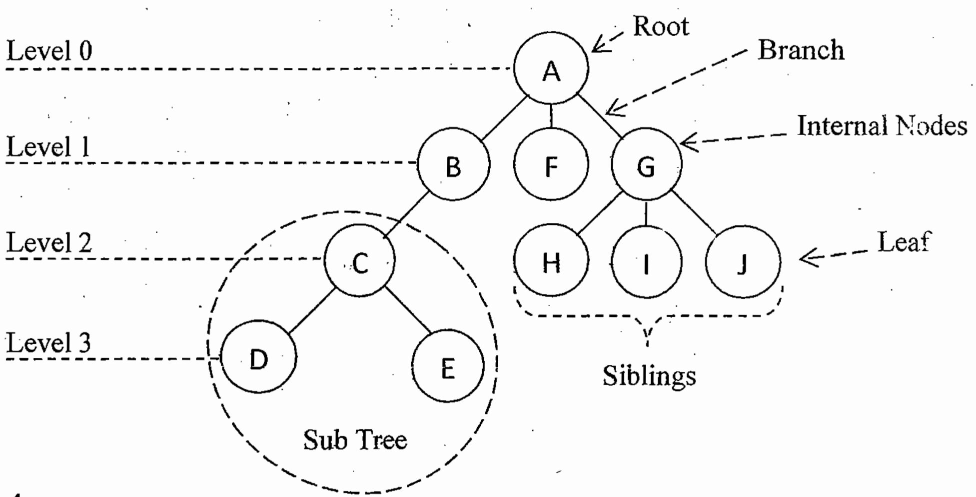
A tree is a finite set of one or more nodes such that

1) There is a specially designated node called the root.
2) The remaining nodes are partitioned into n≥0 disjoint sets $T_1, T_2, \ldots, T_n$, where each of these sets is a tree. $T_1, T_2, \ldots, T_n$ are called the sub trees of the root.



Terminologies

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT



Degree of a Node – The number of sub trees of a node is called as its degree.
Example: The degree of 'A' is 3, degree of 'B' is 1 and degree of 'D' is 0.

Degree of a Tree – The maximum degree of the nodes in the tree is called as degree of a tree.
Example: The degree of the above tree is 3.

Root Node – The first node at the top of the tree which does not have a parent is called as the root node.
Example: Node 'A' is the root node of the above tree.

Leaf Node – A node that has a degree of zero is called as a leaf node or a terminal node. It is node with no children.
Example: The nodes D, E, F, H, I and J are the leaf nodes of the above tree.

Internal Node – The nodes except the leaf nodes are called as the internal nodes.
Example: The nodes A, B, C and G are the internal nodes of the above tree.

Parent – A node having left subtree or right subtree or both is said to be a parent node for the left subtree and / or right subtree.
Example: The parent of D and E is C, The parent of H, I and J is G, the parent of B, F and G is A and the parent of C is B.

Child – The root of the subtree obtained from a parent node is called as a child.
Example: The children of C is D and E, The children of G is H, I and J, the children of A is B, F and G and the child of B is C.

Sibling – Two or more nodes having the same parent are called as siblings.
Example: The nodes D and E are siblings since they have the same parent C, The nodes H, I and J are siblings since they have the same parent G, and the nodes B, F and G are siblings since they have the same parent A.

Ancestors – The ancestors of a node are all the nodes along the path from the root to that node.
Example: Ancestors of D is C, B and A, Ancestors of H is G and A.

Descendants – The descendants of a node are all the nodes that are in its subtree.
Example: Descendants of B is C, D and E, Descendants of G is H, I and J

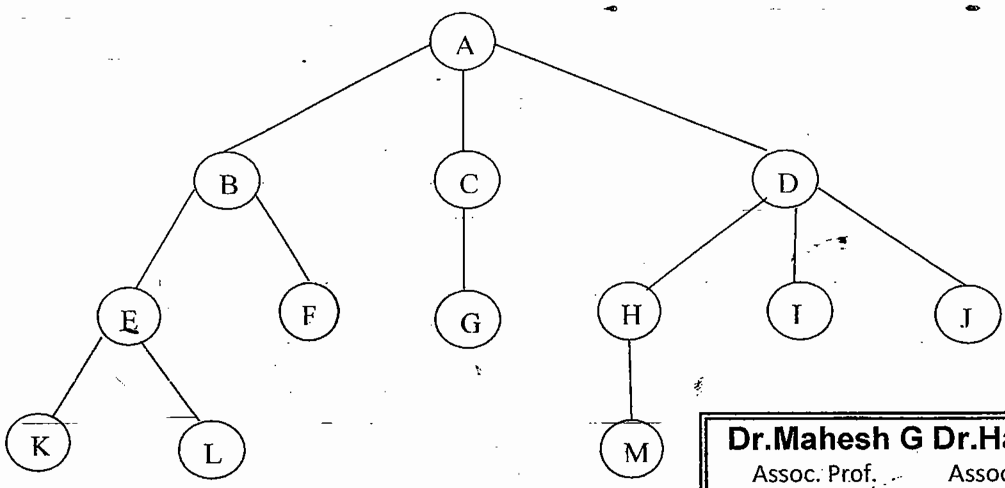
Level – The distance of a node from the root is called as the level of the node.
Example: The distance from the root node A to itself is 0, so the level of root node is 0. The level of other nodes is shown in the diagram.

Height or Depth – The height of a tree is defined as the maximum level of any leaf in the tree plus one.
Example: The maximum level of the above tree is three and hence the height of the above tree is four.

Dr.Mahesh G Assoc. Prof. BMSIT & M
Dr.Harish G Assoc. Prof. Dr. AIT

Representation of Trees

Consider the following tree,

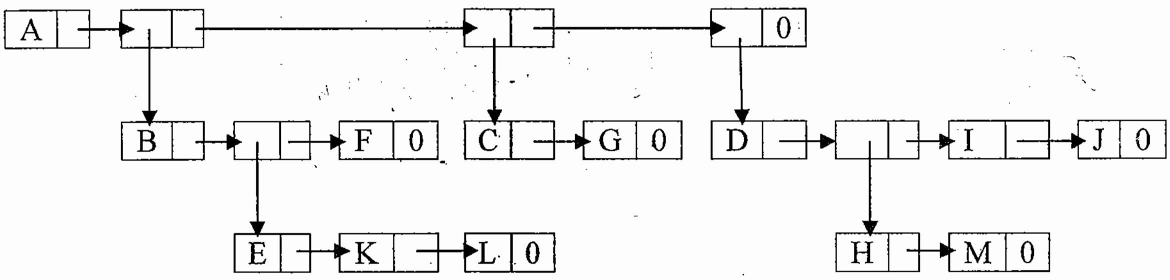


Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

There are three possible ways in which this tree can be represented.

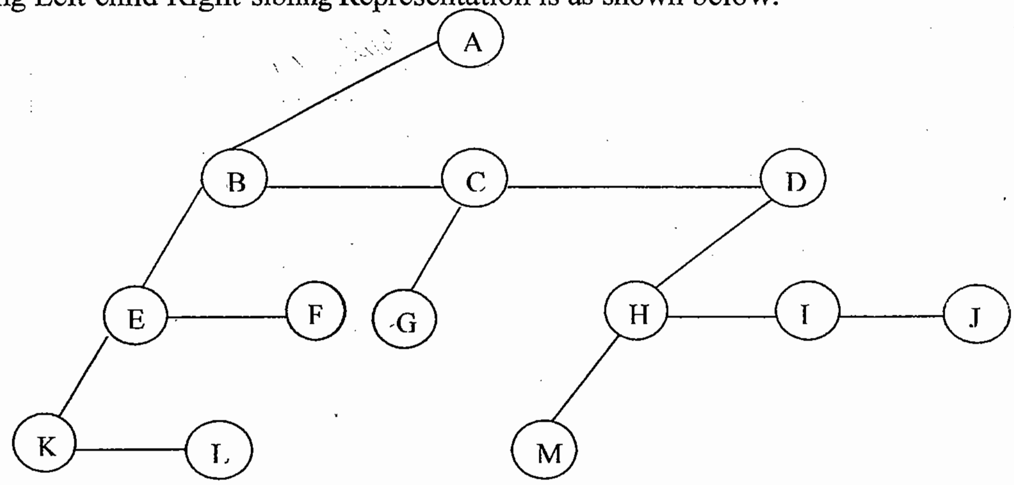
List Representation

In this representation, the information in the root node comes first, which is immediately followed by a list of sub trees of that node. This is recursively repeated for each of the sub tree. The above tree can be written as a list (A (B (E (K L) F) C (G) D (H (M) I J))) The following is the resulting memory representation of the above tree.



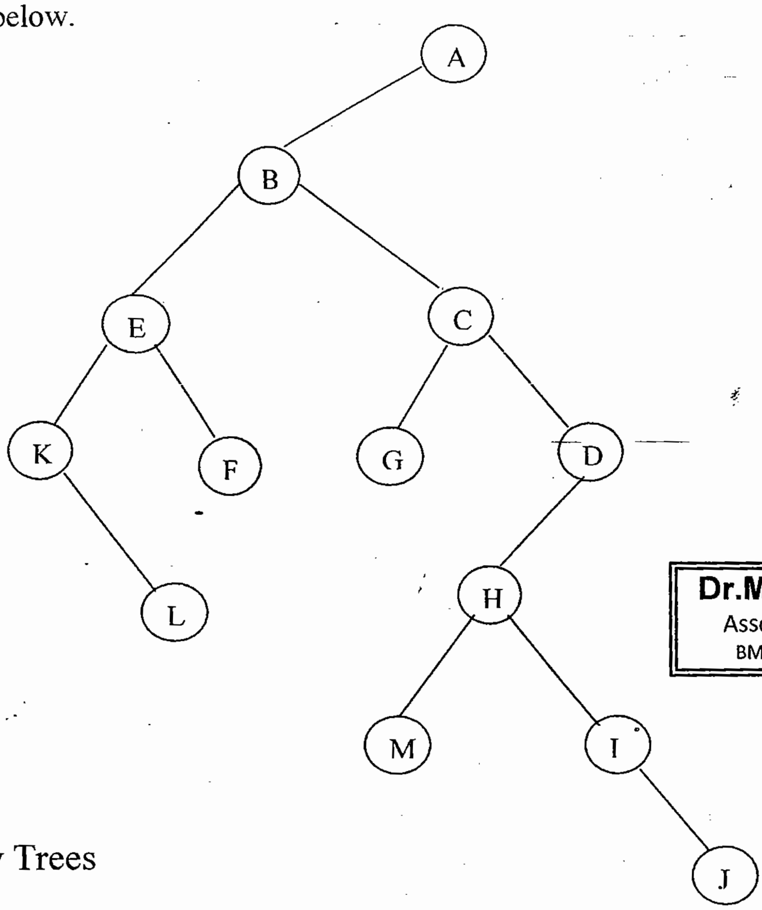
Left-child Right-sibling Representation

In this representation, the left pointer of a node in the tree will be the left child and the remaining children of the node will be inserted horizontally to the left child. The above tree written using Left-child Right-sibling Representation is as shown below.



Degree Two Tree Representation (Also called as Binary Tree Representation)

This representation is obtained by rotating the right-siblings in a Left-child Right-sibling Representation by 45 degrees. The degree two tree representation of the above tree is as shown below.



Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Binary Trees

Definition

A binary tree is a tree which has finite set of nodes that is either empty or consists of a root and two disjoint binary trees called the left sub tree and right sub tree. It is named as binary tree because each node can have atmost 2 sub trees i.e. every node can have 0, 1 or 2 children.

Note:

1. **Root** – If the tree is not empty, the first node is called the root node.
2. **Left sub tree** – It is a tree which is connected to the left of the root. Since this tree comes towards left of root, it is called left sub tree.
3. **Right sub tree** – It is a tree which is connected to the right of the root. Since this tree comes towards right of root, it is called right sub tree.

The Abstract Datatype

The Abstract Data Type of Binary Tree (Bin Tree) is

Objects: A finite set of nodes either empty or consisting of a root node, left binary tree and right binary tree.

Functions:

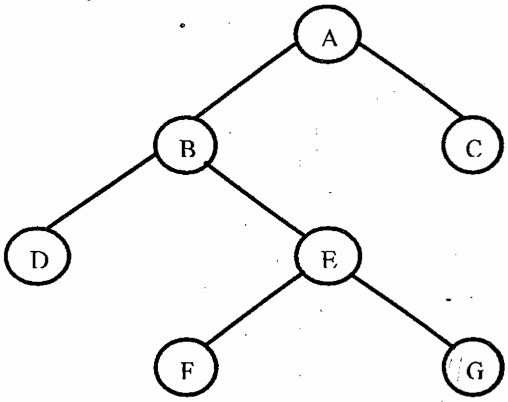
For all,

root, root1 and root2 Є BinTree
item Є element

BinTree Create( )	::=	Creates an empty binary tree
Boolean IsEmpty(root)	::=	**If**(root==NULL) return TRUE **else return** FALSE
BinTree MakeBT(item, root1, root2)	::=	**return** a binary tree whose node contains the data iter with root1 as left sub tree and root2 as right sub tree.
BinTree Lchild(root)	::=	**If**(IsEmpty(root)) **return** error **else** **return** left sub tree of root.
BinTree Rchild(root)	::=	**If**(IsEmpty(root)) **return** error **else** **return** right sub tree of root.
element Data(root)	::=	**If**(IsEmpty(root)) **return** error **else** **return** the data specified in the root node.

Special types of Binary Tree
The following are some of the special types of binary trees

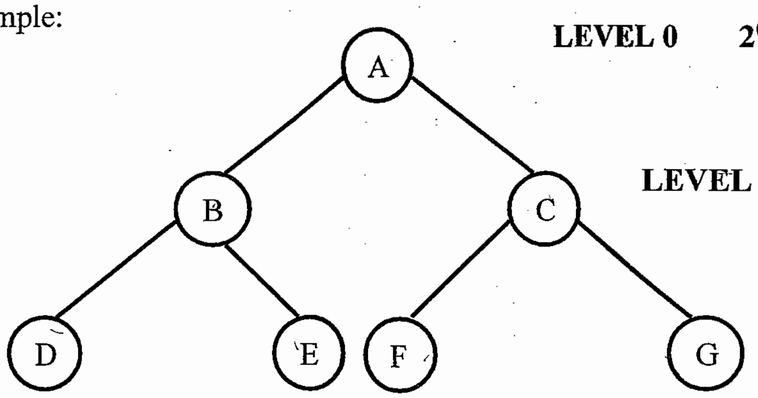
Strictly Binary Tree – It is a binary tree in which the out-degree of every node in a tree is either 0 or 2 i.e. every node must have either 2 children or no child.
Example:



out

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Full or Complete Binary Tree – It is a binary tree that contains maximum possible number of nodes at all levels i.e the number of nodes at any level 'i' is 2^i.
Example:



LEVEL 0 $2^0 = 1$

LEVEL 1 $2^1 = 2$

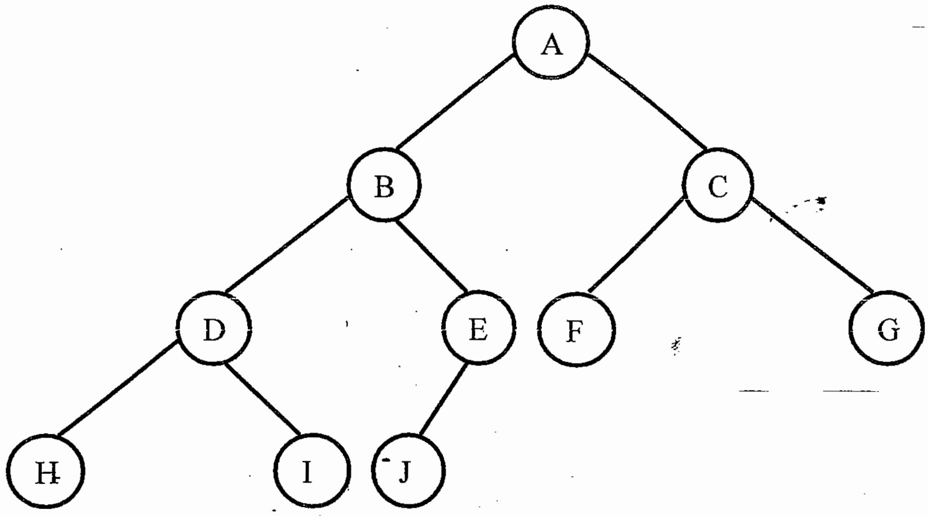
LEVEL 2 $2^2 = 4$

TOTAL NUMBER OF NODES FOR LEVEL 2 = $2^{2+1} - 1 = 8 - 1 = 7$

Almost Complete Binary Tree – It is a binary tree in which

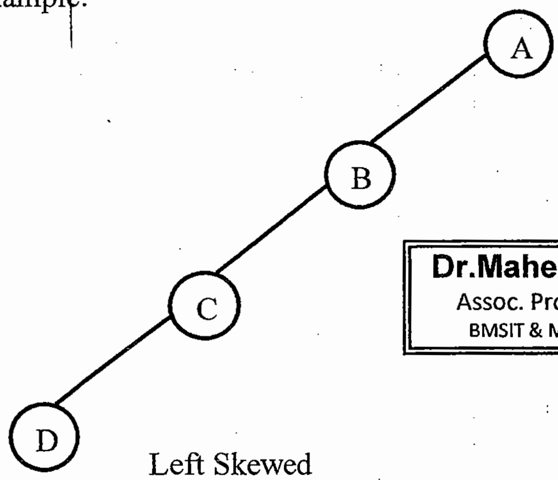
1. It has maximum possible number of nodes at each level except the last level AND
2. All nodes at the last level should be present only from left to right, if the number of nodes at this level is not the maximum possible.

Example:



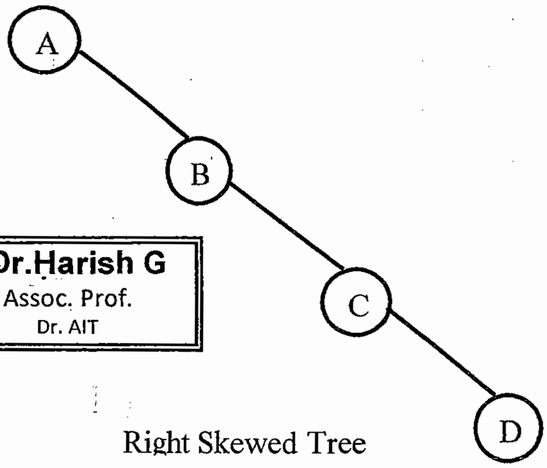
Skewed Tree – A skewed tree is a tree consisting of only left subtree or only right sub tree. A tree with only left subtrees is called as left skewed binary tree and a tree with only right subtrees is called right skewed binary tree.

Example:



Left Skewed

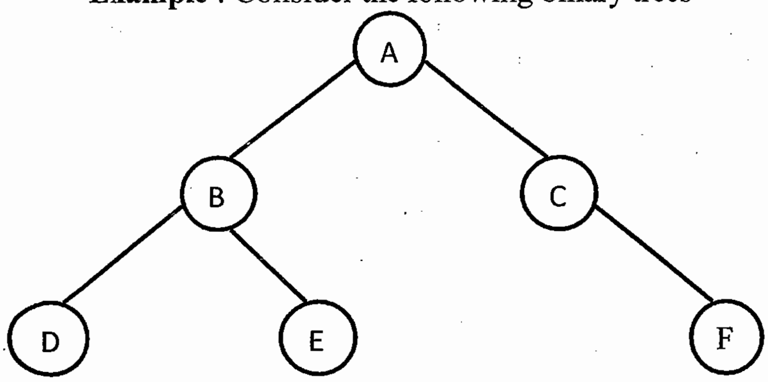
Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

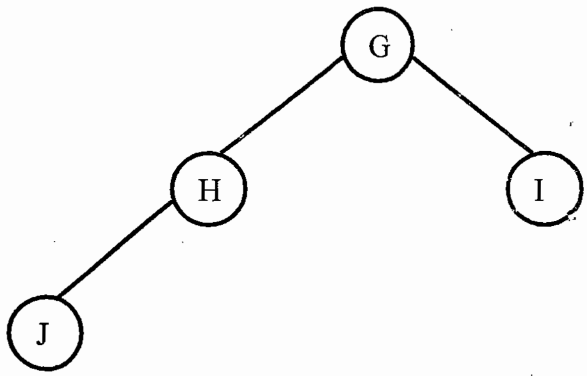


Right Skewed Tree

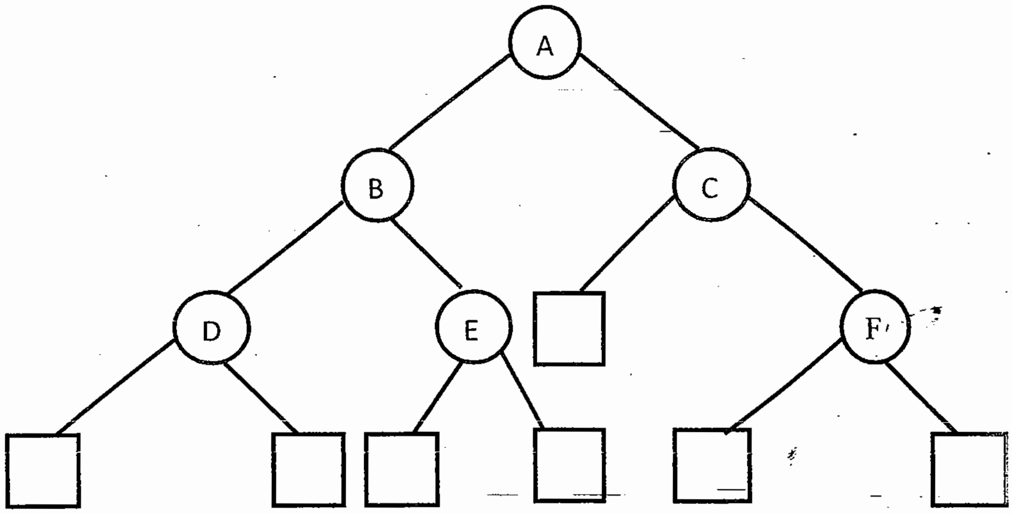
Extended Binary Tree or 2-Tree – A binary tree T is said to be a 2-tree or an extended binary tree if each node N has either 0 or 2 children.

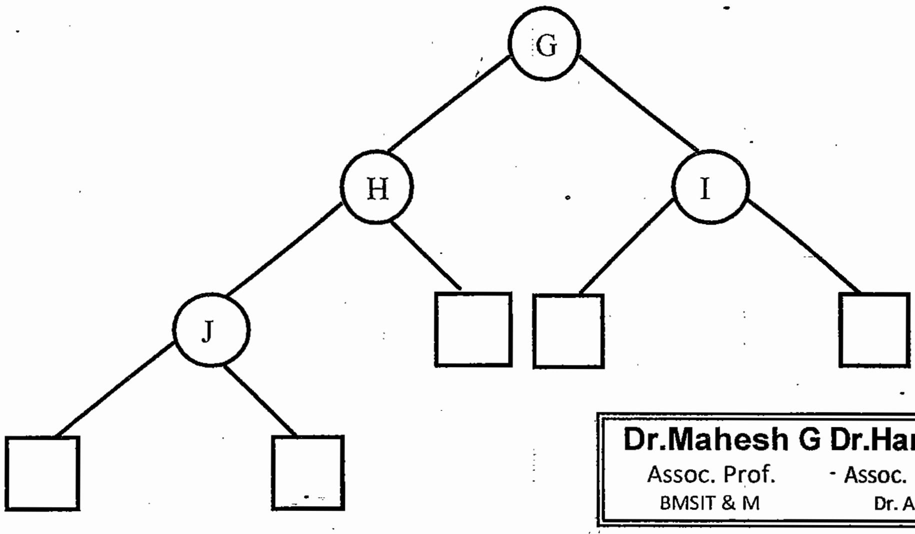
Example : Consider the following binary trees





The extended binary trees for the above binary tress are as shown below.





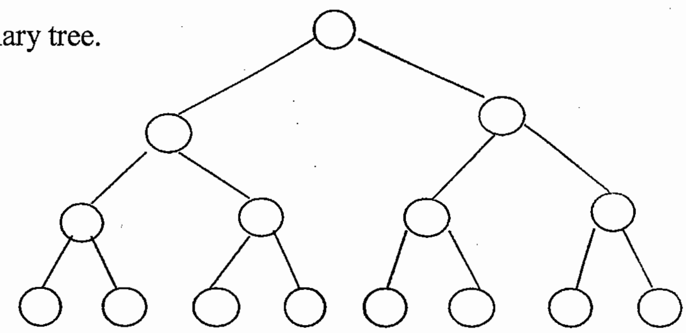
Dr.Mahesh G Assoc. Prof. BMSIT & M
Dr.Harish G Assoc. Prof. Dr. AIT

Note:
1) The square nodes in an extended binary tree are called ***external nodes.***
2) The original (circular) nodes of the binary tree are called as ***internal nodes.***

Properties of a Binary Tree

Lemma 1: The maximum number of nodes on level 'i' of a binary tree is 2^i for $i \geq 0$

Proof: Consider the following complete binary tree.



Number of nodes at level 0 = 1 = 2^0
Number of nodes at level 1 = 2 = 2^1
Number of nodes at level 2 = 4 = 2^2
Number of nodes at level 3 = 8 = 2^3
....
Hence, Number of nodes at level i = 2^i

Lemma 2: The Total number of nodes in a full binary tree of level 'i' is $2^{i+1} - 1$.

Proof:
Total number of nodes in a full binary tree of level 'i' is given by
$N_t = 2^0 + 2^1 + 2^2 + 2^3 \ldots\ldots\ldots + 2^i$
This series is in geometric progression whose sum is $S = a(r^n - 1) / (r - 1)$
Where,

a is the first term = 1
r is the common ratio = 2
n is the number of terms = i+1

Hence $N_t = a(r^n - 1) / (r - 1) = 1(2^{i+1} - 1) / (2 - 1) = 2^{i+1} - 1$

Lemma 3: The maximum number of nodes in a binary tree of depth $k = 2^k - 1$.

Proof: From Lemma 2, we have the Total number of nodes in a full binary tree of level 'i' is $2^{i+1} - 1$. Also, by definition, we have depth of a tree = maximum level +1.

Hence $k = i + 1$;

Substituting this value in $N_t = 2^{i+1} - 1$, we get $N_t = 2^k - 1$

Lemma 4: For any non empty binary tree, T, if n_0 is the number of leaf nodes and n_2 is the number of nodes of degree 2, then $n_0 = n_2 + 1$. (Relation between number of leaf nodes and degree-2 nodes)

Proof: Let n_0 be the number of nodes with degree 0, n_1 be the number of nodes with degree 1 and n_2 be the number of nodes with degree 2. Then the total number of nodes in the tree is $N_t = n_0 + n_1 + n_2$ ....(1)

Let 'B' be the total number of branches in the tree. The total number of nodes in a tree is equal to the total number of branches plus one i.e. $N_t = B + 1$..................(2)

Branches stem from a node with degree 1 or degree 2.
If there is node with degree 1, then the number of branches = 1. Hence, for n_1 number of nodes with degree 1, number of branches = n_1
If there is node with degree 2, then the number of branches = 2. Hence, for n_2 number of nodes with degree 2, number of branches = $2n_2$
With this total number of branches $B = n_1 + 2n_2$ ....(3)

From (1) and (2) we have $n_0 + n_1 + n_2 = B + 1$ ....(4)
Substituting (3) in (4) we get,
$n_0 + n_1 + n_2 = n_1 + 2n_2 + 1$
$n_0 = n_1 + 2n_2 + 1 - n_1 - n_2$
Therefore, $n_0 = n_2 + 1$

Binary Tree Representation

Binary trees can be represented using dynamic memory allocation (Linked Representation) or sequential allocation (Array Representation).

Linked Representation

In this representation, each node in a tree has three fields.

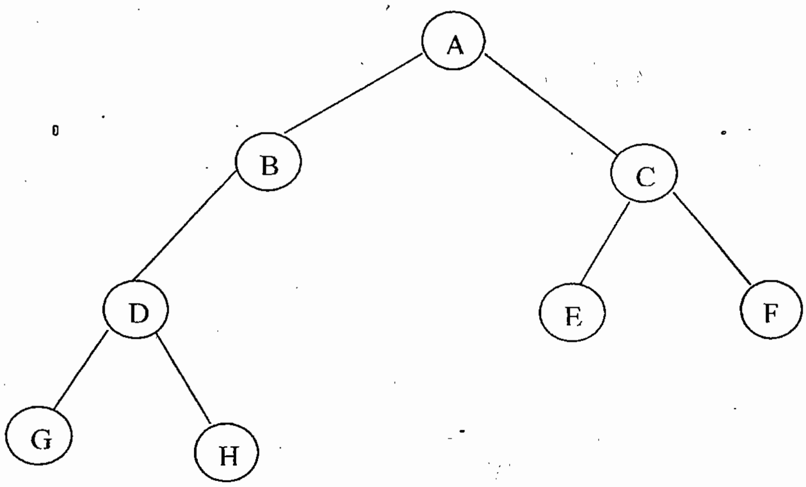
1. info – Contains the actual information
2. llink – Contains the address of left sub tree
3. rlink – Contains the address of right sub tree

So, a node can be represented using the structure as shown below.

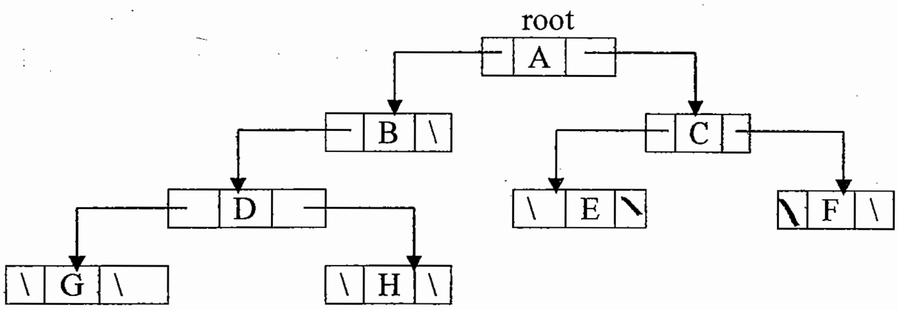
```
struct node
{
        int info;
        struct node *llink;
        struct node *rlink;
};
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Example: Consider the tree shown below.



The linked representation of the above tree is as shown below.

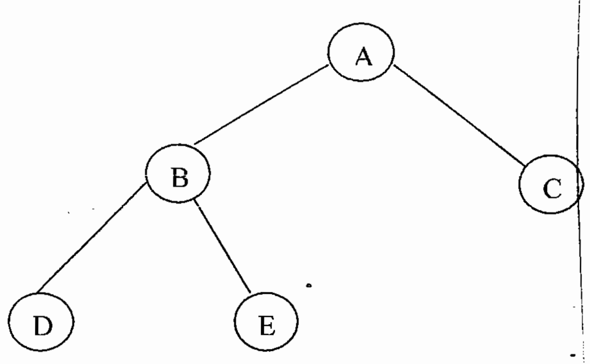


Note: A variable root always points to the root node. If root = NULL, then the tree is empty.

Array Representation

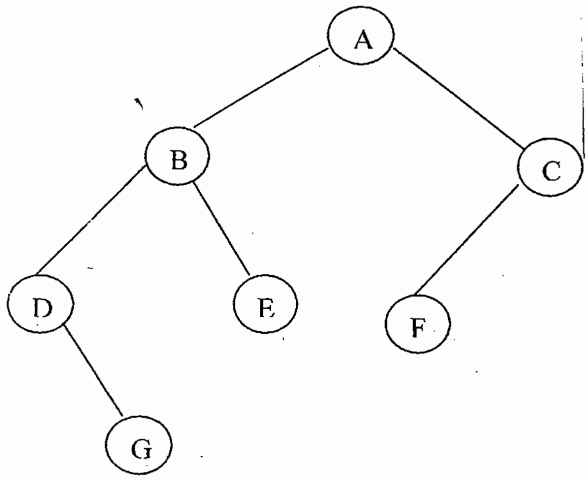
This representation of binary tree requires numbering of nodes starting with nodes on level 0, then on level 1 and so on from left to right with numbers 0, 1, 2, ……
These nodes are maintained in a 1-D array in appropriate subscripts according to their node number.

Example 1:



0	1	2	3	4
A	B	C	D	E

Example 2:



0	1	2	3	4	5	6	7	8
A	B	C	D	E	F	-	-	G

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Note:

1) The index i = 0 always gives the position of the root node.
2) Given the position of a node 'i'
   Left child's position = $2i + 1$
   Right child's position = $2i + 2$
   Parents position = $(i - 1) / 2$

Binary Tree Traversals

Definition

Traversing means visiting each node of the tree exactly once in a systematic manner. During traversing we may print the info field of each node visited.

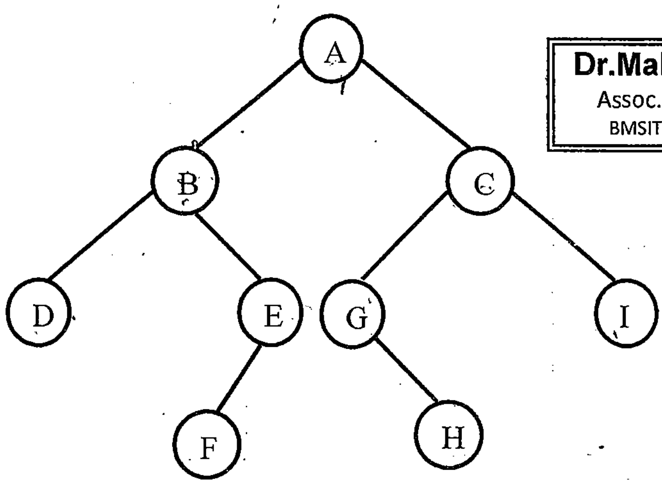
The different traversal techniques of a binary tree are

1) Pre-Order – It is defined as follows
Step 1: Visit the node
Step 2: Recursively traverse the left subtree in Pre-Order
Step 3: Recursively traverse the right subtree in Pre-Order

2) In-Order– It is defined as follows
Step 1: Recursively traverse the left subtree in In-Order
Step 2: Visit the node
Step 3: Recursively traverse the right subtree in In-Order

3) Post-Order– It is defined as follows
Step 1: Recursively traverse the left subtree in Post-Order
Step 2: Recursively traverse the right subtree in Post-Order
Step 3: Visit the node

Problem 1 – Traverse the below tree in Pre-Order, In-Order and Post-Order



Dr.Mahesh G Assoc. Prof. BMSIT & M
Dr.Harish G Assoc. Prof. Dr. AIT

Pre-Order (N L R)	In-Order (L N R)	Post-Order (L R N)
A B_P C_P	B_I A C_I	B_P C_P A
A B D_P E_P C_P	D_I B E_I A C_I	D_P E_P B C_P A
A B D E_P C_P	D B E_I A C_I	D E_P B C_P A
A B D E F_P C_P	D B F_I E A C_I	D F_P E B C_P A
A B D E F C_P	D B F E A C_I	D F E B C_P A
A B D E F C G_P I_P	D B F E A G_I C I_I	D F E B G_P I_P C A
A B D E F C G H_P I_P	D B F E A G H_I C I_I	D F E B H_P G I_P C A
A B D E F C G H I_P	D B F E A G H C I_I	D F E B H G I_P C A
A B D E F C G H I	D B F E A G H C I	D F E B H G I C A

Inserting an element into a Binary Tree based on direction

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

```
NODE insert_node(int item, NODE root)
{
        NODE temp;   //Node to be inserted
        NODE cur;    //Child node
        NODE prev;   //parent node
        char direction[20];  //directions where the node has to be inserted
        int i;

        temp = (NODE) malloc(sizeof(struct node));
        temp->info=item;
        temp->llink=NULL;
        temp->rlink=NULL;

        if(root = = NULL)
        {
                return temp;
        }

        printf("give the directions where u want to insert\n");
        scanf("%s", direction);

        prev=NULL;
        cur=root;
        for(i=0; i<strlen(direction) && cur!=NULL; i++)
        {
                prev=cur;
                if(direction[i] = = 'l')  //direction l move to left
                        cur=cur->llink;
                else
                        cur=cur->rlink;  //otherwise move to right
        }

        if(cur!=NULL || i!=strlen(direction))
        {
                printf("insertion not possible\n");
                free( temp);
                return root;
        }

        if(direction[i-1]=='l')
                prev->llink=temp;       //attach node to left of parent
        else
                prev->rlink=temp;       //attach node to right of parent

        return root;
}
```

Recursive Inorder Traversal

```
void inorder(NODE root)
{
        if(root = = NULL)
                return;
        inorder(root->llink);  L
        printf("%d\n",root->info); N
        inorder(root->rlink); R
}
```

Recursive Preorder Traversal

```
void preorder(NODE root)
{
        if(root = = NULL)
                return;
        printf("%d\n",root->info); N
        preorder(root->llink); L
        preorder(root->rlink); R
}
```

Recursive Postorder Traversal

```
void postorder(NODE root)
{
        if(root = = NULL)
                return;
        postorder(root->llink); L
        postorder(root->rlink); R
        printf("%d\n",root->info); N
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Function to print in the form of Tree

```
void display(NODE root, int height)
{
        int i;
        if(root = = NULL)
                return;
        display(root->rlink, height+1);
        for(i=1; i<=heightl; i++)
                printf("--");
        printf("%d\n",root->info);
        display(root->llink, height+1);
}
```

Main Function for Binary Tree

```
void main( )
{
        NODE root = NULL;
        int item, choice;
        for(;;)
        {
                printf("\n1:insert_node\n2:inorder\n3:preorder\n");
                printf("4:postorder\n5.display\n6:exit\n");
                printf("enter your choice\n");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: printf("enter the item to be inserted\n");
                                scanf("%d",&item);
                                root = insert_node(item, root);
                                break;

                        case 2: if(root = = NULL)
                                        printf("the tree is empty\n");
                                else
                                        inorder(root);
                                break;

                        case 3: if(root = = NULL)
                                        printf("the tree is empty\n");
                                else
                                        preorder(root);
                                break;

                        case 4: if(root = = NULL)
                                        printf("the tree is empty\n");
                                else
                                        postorder(root);
                                break;

                        case 5: if(root = = NULL)
                                        printf("the tree is empty\n");
                                else
                                        display(root,1);
                                break;

                        default : exit(0);
                }
        }
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Iterative Inorder Traversal

```
void inorder(NODE root)
{
        NODE cur, s[25];
        int top = -1;
        if(root = = NULL)
        {
                printf("tree is empty\n");
                return;
        }
        cur = root;
        for( ; ; )
        {
            while(cur != NULL)
            {
                push(cur, &top, s);
                cur = cur->llink;
            }
            if(top != -1)
            {
                cur = pop(&top, s);
                printf("%d\n", cur->info);
                cur = cur->rlink;
            }
            else
                break;
        }
}
```

Iterative Preorder Traversal

```
void preorder(NODE root)
{
        NODE cur, s[25];
        int top = -1;
        if(root = = NULL)
        {
                printf("tree is empty\n");
                return;
        }
        cur = root;
        for( ; ; )
        {
            while(cur != NULL)
            {
                printf("%d\n", cur->info);
                push(cur, &top, s);
                cur = cur->llink;
            }
            if(top != -1)
            {
                cur = pop(&top, s);
                cur = cur->rlink;
            }
            else
                break;
        }
}
```

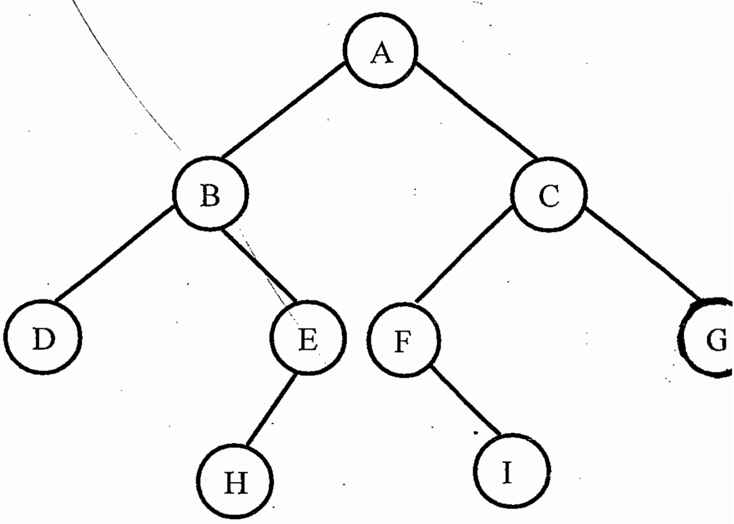
Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Level Order Traversal

The nodes in a tree are numbered starting with the root node on level 0, and continuing with nodes on level1, level2 and so on. Nodes on any level are numbered from left to right. Visiting the nodes according to this numbering scheme is called as level order traversal. This traversal technique uses a queue.

Example: For the tree shown below, level order traversing is A B C D E F G H I

Contents of the Queue	Output
A	
B C	A
C D E	A B
D E F G	A B C
E F G	A B C D
F G H	A B C D E
G H I	A B C D E F
H I	A B C D E F G
I	A B C D E F G H
QUEUE EMPTY STOP	A B C D E F G H I



Algorithm

Step 1: Insert the root into queue

Step 2: As long as the queue is not empty, delete an element from the queue and print the info field. If the left subtree exists insert it into the queue. If the right subtree exists insert into the queue.

Function for level order traversing of a binary tree

```
void level_order(NODE root)
{
        NODE q[100], cur;

        int f=0, r=-1;

        r = r + 1;
        q[r] = root;

        while( f<= r)
        {
                cur = q[f];
                f = f + 1;

                printf("%d\n", cur->info);

                if(cur->llink !=NULL)
                {
                        r = r + 1;
                        q[r] = cur->llink;
                }

                if(cur->rlink !=NULL)
                {
                        r = r + 1;
                        q[r] = cur->rlink;
                }
        }
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Binary Search Trees

Dictionary

A dictionary is a collection of pairs (key, item), where each key has an item associated with it. It is assumed that no two pairs have the same key.

Example: The dictionary of telephone numbers and the name of the person who holds that number is as shown below.

Number	Name
111111111	Mahesh
222222222	Harish
333333333	Ramesh
444444444	Suresh

ADT of Dictionary

Objects: ~~A~~ collection of n>0 pairs, where each pair has a key and associated item.

Dr.Mahesh G Assoc. Prof. BMSIT & M
Dr.Harish G Assoc. Prof. Dr. AIT

Functions:
For all,
d Є Dictionary
item, key, n Є integer

Dictionary Create( )	::=	Create an empty dictionary
Boolean IsEmpty(d, n)	::=	**If**(n == 0) return TRUE **else return** FALSE
void Insert(item, k, d)	::=	Insert the item with key k into d.
element Delete(d, k)	::=	Delete and return item with key k if present.
element search(d, k)	::=	**return** item with key k if present **return** NULL if no such element.

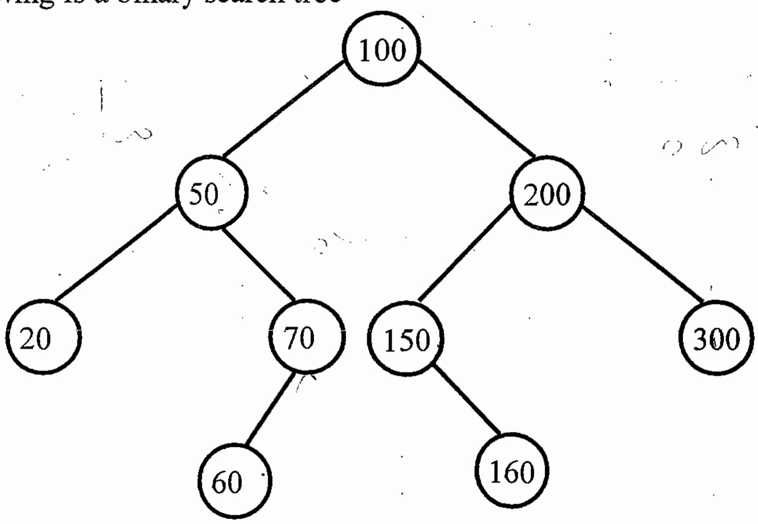
BST Definition

A binary search tree is a binary tree. It may be empty. If it is now empty, then it satisfies the following properties.

1) Each node has exactly one key and the keys in the tree are distinct.
2) The keys (if any) in the left subtree are smaller than the key in the root.
3) The keys (if any) in the right subtree are larger than the key in the root.
4) The left and right subtrees are also binary search trees.

This means, for each node say 'x', in the tree, if its left or right sub tree exists, then elements in the left subtree are less than info(x) and elements in the right subtree are greater than info(x).

Example: The following is a binary search tree



Note:

1) Traversing a BST is same as traversing a binary tree. The inorder, preorder and postorder traversals of the above tree is given by,

Inorder : 20, 50, 60, 70, 100, 150, 160, 200, 300

Preorder : 100, 50, 20, 70, 60, 200, 150, 160, 300

Postorder : 20, 60, 70, 50, 160, 150, 300, 200, 100

2) Inorder traversal on a binary search tree will give sorted order of data in ascending order. This method of sorting is known as binary sort.

3) In preorder traversal, the root is processed first, before the left and right sub trees.

4) In inorder traversal, the root is processed between its subtrees.

5) In postorder traversal, the root is processed after the subtrees.

Inserting an element into BST

```
NODE insert_node(int item, NODE root)
{
        NODE cur,temp,prev;
        temp = (NODE)malloc(sizeof(struct node));
        temp->info = item;
        temp->llink = temp->rlink = NULL;

        if(root = = NULL)
                return temp;

        prev = NULL;
        cur = root;
        while(cur != NULL)
        {
                prev = cur;
                if(item < cur->info)
                        cur = cur->llink;
                else
                        cur = cur->rlink;
        }
            55 < 60
        if(item < prev->info)
                prev->llink = temp;
        else
                prev->rlink = temp;
        return root;
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Note: To avoid duplicate elements in tree the necessary modification is; change the while loop as shown below.

```
while(cur != NULL)
{
        prev = cur;
        if(item < cur->info)
                cur = cur->llink;
        else if(item > cur->info)
                cur = cur->rlink;
        else
        {
                printf("item already exists!cannot insert\n");
                free(temp);
                return root;
        }
}
```

Count the Nodes in a Tree

```
// assume count is a global variable which is initialized to zero, before calling the function in the main program.
void count_nodes(NODE root)
{
        if(root = = NULL)
                return;

        count_nodes (root->llink);

        count++;

        count_nodes (root->rlink);
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Count the Leaves in a Tree

```
// assume count is a global variable which is initialized to zero, before calling the function in the main program.
void count_leaves(NODE root)
{
        if(root = = NULL)
                return;

        count_leaves (root->llink);

        if(root->llink ==NULL && root->rlink==NULL)
                count++;

        count_leaves (root->rlink);
}
```

Height of a Tree

```
int max(int a, int b)
{
        If(a>b)
                return a;
        else
                return b;
}
int height(NODE root)
{
        if(root = = NULL)
                return 0;
        else
                return( max( height(root->llink), height(root->rlink) ) + 1);
}
```

Sum of all the nodes in a Tree

```
// assume sum is a global variable which is initialized to zero, before calling the function in the main program.
void add_nodes(NODE root)
{
        if(root = = NULL)
                return;

        add_nodes (root->llink);

        sum = sum + root->info;

        add_nodes (root->rlink);
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

// Main function to call the above functions

```
int count, sum;
void main( )
{
        NODE root = NULL;
        int item, choice;
        clrscr( );
        for(;;)
        {
                printf("\n1:insert_node\n2:count nodes\n3:count leaves\n");
                printf("4:height\n5.sum\n6:exit\n");
                printf("enter your choice\n");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: printf("enter the item to be inserted\n");
```

```
                scanf("%d",&item);
                root = insert_node(item, root);
                break;

            case 2: count = 0;
                count_nodes(root);
                printf("number of node = %d\n", count);
                break;

            case 3: count = 0;
                count_leaves(root);
                printf("number of leaves = %d\n", count);
                break;

            case 4: printf("the height of the tree is %d\n", height(root));
                break;

            case 5: sum = 0;
                add_nodes(root);
                printf("Sum of all the nodes  = %d\n", sum);
                break;

            default : exit(0);
        }
    }
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Recursive Search of a BST

```
NODE recursive_search(int item, NODE root)
{
    If(root==NULL || item == root->info)
        return root;

    if(item < root->info)
        return recursive_search(item, root->llink);

    return recursive_search(item, root->rlink);
}
```

Iterative Search of a BST

```
NODE iterative_search(int item, NODE root)
{
    NODE cur;
    cur = root;
    while( cur!=NULL && item!= cur->info)
    {
        If(item<cur->info)
            cur = cur->llink;
```

```
            else
                  cur = cur->rlink;
      }
      return cur;
}
```

Maximum Value in a BST

```
NODE maximum_node(NODE root)
{
      If(root==NULL)
            return root;

      while(root->rlink != NULL)
            root = root->rlink;

      return root;
}
```

Minimum Value in a BST

```
NODE minimum_node(NODE root)
{
      If(root==NULL)
            return root;

      while(root->llink != NULL)
            root = root->llink;

      return root;
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Create a copy of a Binary Tree

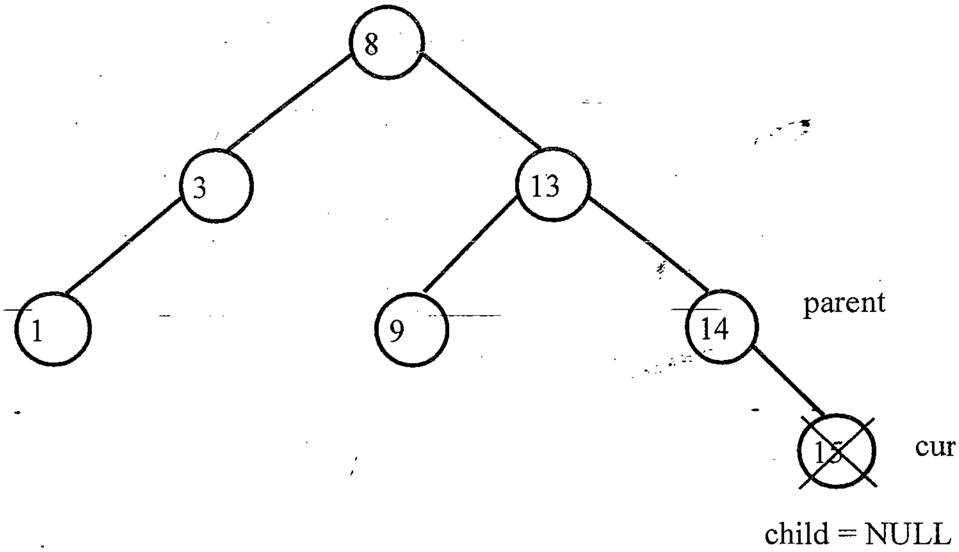
```
NODE copy(NODE root)
{
      NODE temp;
      If(root == NULL)
            return NULL;

      temp = (NODE)malloc(sizeof(struct node));

      temp->info = root->info;

      temp->llink = copy(root->llink);

      temp->rlink = copy(root->rlink);

      return temp;
}
```

Deletion from a BST

Deleting a node requires searching for the node and then delete with also maintaining the ordering and properties of binary search tree.

~~Note:~~ "cur" is found to be the node that gets deleted.

Case 1: Node to be deleted (cur) has no children.

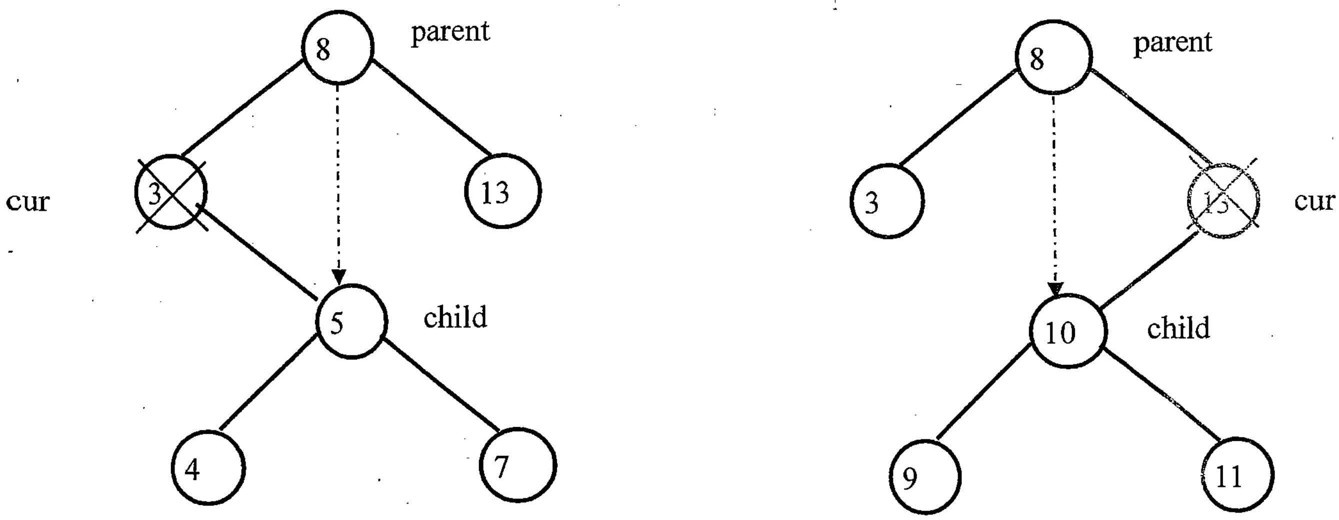


No adjustments is needed except to make parents right link as NULL as in this example. (In some cases, left link must be made NULL)

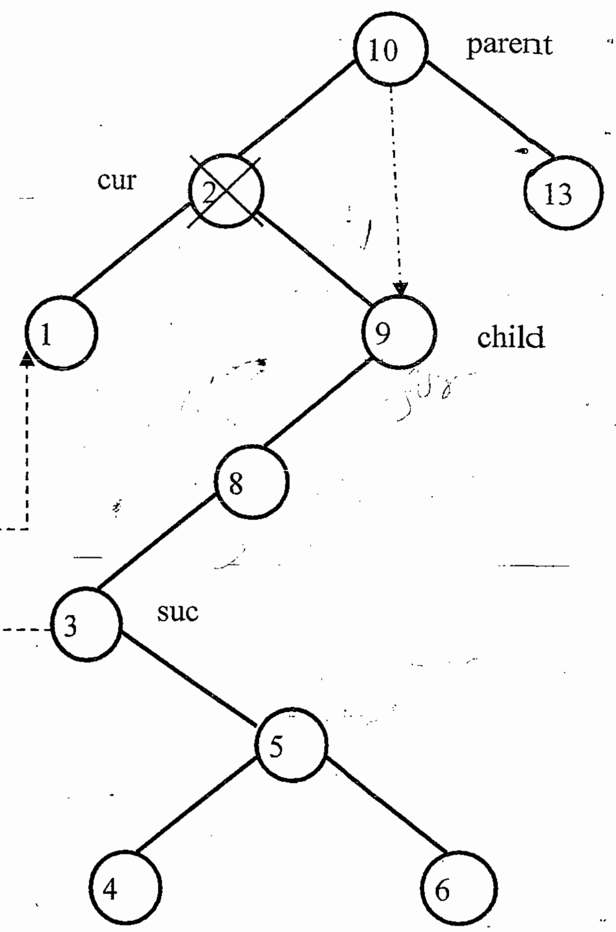
```
// child = NULL
// establish links and delete
If(cur == par->llink)
        par->llink=child;
else
        par->rlink=child;
free(cur);
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Case 2: Node to be deleted (cur) has only one subtree.



```
// Find the child
If(cur->llink == NULL)
        child = cur->rlink;
else if(cur->rlink == NULL)
        child = cur->llink;

// establish links and delete
If(cur == par->llink)
        par->llink=child;
else
        par->rlink=child;
free(cur);
```



Case 3: Node to be deleted (cur) has both subtrees

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Find the inorder successor of 'cur' as 'suc'. Inorder successor of any node will be the left most node of the right subtree of that node. Make the left subtree of 'cur' as the left subtree of 'suc'

```
// Find the child
child = cur->rlink;

// Find the inorder successor and make left subtree of cur as left subtree of suc
suc = cur->rlink;
while(suc->llink != NULL)
{
        suc = suc->llink;
}
suc->llink = cur->llink;

// establish links and delete
If(cur == par->llink)
        par->llink=child;
else
        par->rlink=child;
free(cur);
```

// Function to delete a node from a binary search tree

```
NODE deletenode(int item, NODE root)
{
        NODE child, suc, par, cur;
        If(root == NULL)
        {
                printf("tree is empty\n");
                return root;
        }
        par = NULL;
        cur = root;
        while(cur != NULL && item != cur->info)
        {
                par = cur;
                if(item < cur->info)
                        cur = cur->llink;
                else
                        cur = cur->rlink;
        }
        If(cur == NULL)
        {
                printf("item not found\n");
                return root;
        }
        If(cur->llink == NULL)
                child = cur->rlink;
        else if(cur->rlink == NULL)
                child = cur->llink;
        else
        {
                child = cur->rlink;
                suc = cur->rlink;
                while(suc->llink != NULL)
                {
                        suc = suc->llink;
                }
                suc->llink = cur->llink;
        }
        If(par == NULL)       // if node to be deleted is the root
        {
                free(cur);
                return child;
        }
        If(cur == par->llink)
                par->llink=child;
        else
                par->rlink=child;
        free(cur);
        return root;
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Construction of Binary Search Tree

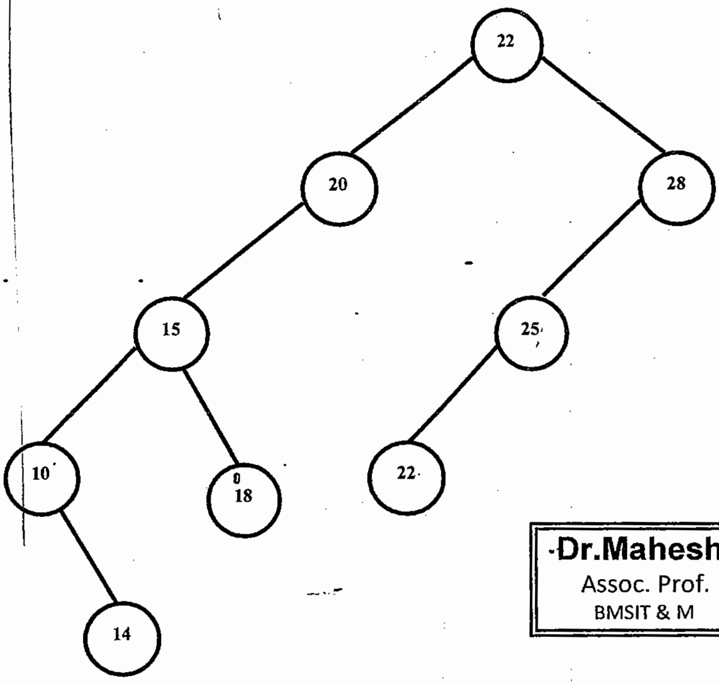
Note:

1) First item to be inserted will be the root node
2) If item to be inserted is less than root insert to appropriate position in left sub tree
3) If item to be inserted is greater than root insert to appropriate position in right sub tree

Problem 1: Construct binary search tree for the following input

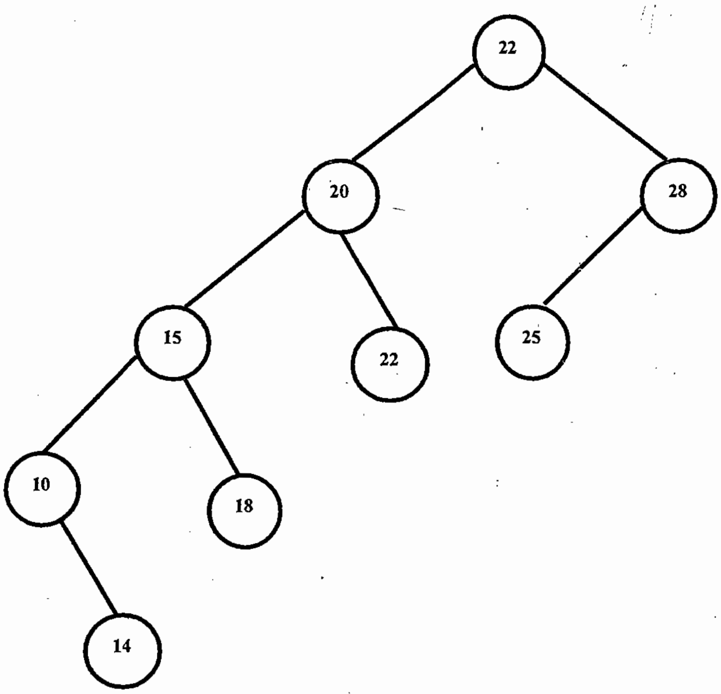
22, 28, 20, 25, 22, 15, 18, 10, 14

Assumption: If item to be inserted is equal to root insert to right sub tree



Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Assumption: If item to be inserted is equal to root insert to left sub tree



Design, Develop and Implement a menu driven Program in C for the following operations on **Binary Search Tree (BST)** of Integers

a. Create a BST of **N** Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (**KEY**) and report the appropriate message
d. Delete an element(ELEM) from BST
e. Exit

```
#include<stdio.h>

struct node
{
	int info;
	struct node *llink;
	struct node *rlink;
};
typedef struct node* NODE;

// Function to a create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
NODE insert_node(int item, NODE root)
{
	NODE cur,temp,prev;
	temp = (NODE)malloc(sizeof(struct node));
	temp->info = item;
	temp->llink = temp->rlink = NULL;

	if(root = = NULL)
		return temp;

	prev = NULL;
	cur = root;
	while(cur != NULL)
	{
		prev = cur;
		if(item < cur->info)
			cur = cur->llink;
		else
			cur = cur->rlink;
	}

	if(item < prev->info)
		prev->llink = temp;
	else
		prev->rlink = temp;
	return root;
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

// Function to Traverse the BST in Inorder

```
void inorder(NODE root)
{
        if(root = = NULL)
            return;
        inorder(root->llink);
        printf("%d\t",root->info);
        inorder(root->rlink);
}
```

// Function to Traverse the BST in Preorder

```
void preorder(NODE root)
{
        if(root = = NULL)
            return;
        printf("%d\t",root->info);
        preorder(root->llink);
        preorder(root->rlink);
}
```

Dr.Mahesh G Assoc. Prof. BMSIT & M
Dr.Harish G Assoc. Prof. Dr. AIT

// Function to Traverse the BST in Postorder

```
void postorder(NODE root)
{
        if(root = = NULL)
            return;
        postorder(root->llink);
        postorder(root->rlink);
        printf("%d\t",root->info);
}
```

// Function to search for a key in BST

```
NODE search(int item, NODE root)
{
        If(root==NULL || item = = root->info)
            return root;

        if(item < root->info)
            return search(item, root->llink);

        return search(item, root->rlink);
}
```

// Function to delete a node from a binary search tree

```
NODE delete_node(int item, NODE root)
{
        NODE child, suc, par, cur;
        If(root == NULL)
        {
                printf(“tree is empty\n”);
                return root;
        }
        par = NULL;
        cur = root;
        while(cur != NULL && item != cur->info)
        {
                par = cur;
                if(item < cur->info)
                        cur = cur->llink;
                else
                        cur = cur->rlink;
        }
        If(cur == NULL)
        {
                printf(“item not found\n”);
                return root;
        }
        If(cur->llink == NULL)
                child = cur->rlink;
        else if(cur->rlink == NULL)
                child = cur->llink;
        else
        {
                child = cur->rlink;
                suc = cur->rlink;
                while(suc->llink != NULL)
                {
                        suc = suc->llink;
                }
                suc->llink = cur->llink;
        }
        If(par == NULL)        // if node to be deleted is the root
        {
                free(cur);
                return child;
        }
        If(cur == par->llink)
                par->llink=child;
        else
                par->rlink=child;
        free(cur);
        return root;
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

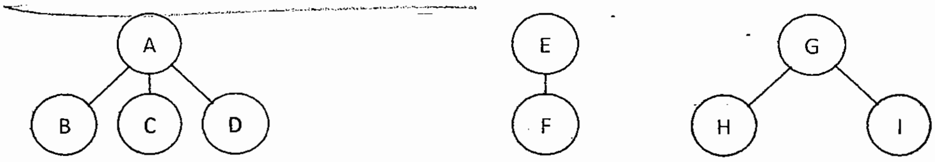
```
void main( )
{
        int choice, item;
        NODE root = NULL, temp, parent;
        for( ; ; )
        {
            printf("1.Create\n");
            printf("2.Traverse the Tree in Preorder, Inorder, Postorder\n");
            printf("3.Search\n");
            printf("4.Delete an element from the Tree\n");
            printf("5.Exit\n");
            printf("Enter your choice\n");
            scanf("%d", &choice);

            switch (choice)
            {
                case 1: printf("Enter the item to be inserted \n");
                        scanf("%d", &item);
                        root = insert_node(item, root);
                        break;

                case 2: if (root == NULL)
                            printf("Tree Is Not Created");
                        else
                        {
                            printf("\nThe Inorder display : ");
                            inorder(root);
                            printf("\nThe Preorder display : ");
                            preorder(root);
                            printf("\nThe Postorder display : ");
                            postorder(root);
                        }
                        break;

                case 3: printf("Enter Element to be searched \n");
                        scanf("%d", &item);
                        temp = search(item, root);
                        if(temp = = NULL)
                            printf("Element does not exists\n");
                        else
                            printf("The element %d is found\n", temp->info);
                        break;

                case 4: printf("Enter Element to be deleted \n");
                        scanf("%d", &item);
                        root = delete_node(item, root);
                        break;

                default: exit(0);
            }
        }
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Forests

Definition

A forest is a collection of ~~zero or~~ more trees. The following shows a forest with three trees.

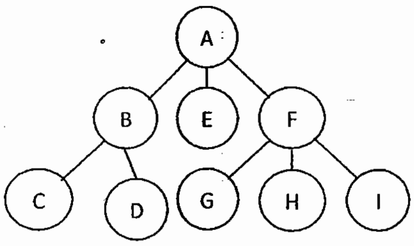


Converting a Tree into a Binary Tree

Algorithm

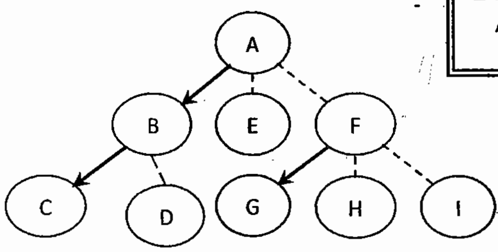
1) Identify the branch from the parent to its first or leftmost child. These branches from each parent become left pointers in the binary tree
2) Connect siblings, starting with the leftmost child, using a branch for each sibling to its right sibling.
3) Remove all unconnected branches from the parent to its children

Example: Consider the following tree .

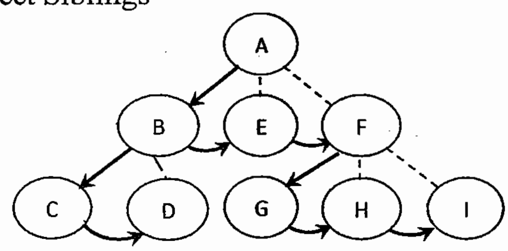


Step 1: Identify all left most children

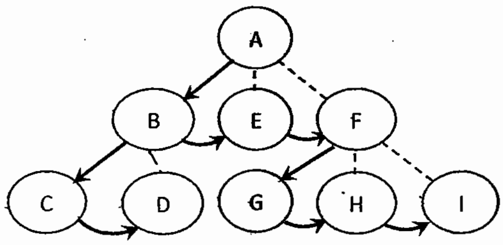
Dr.Mahesh G Assoc. Prof. BMSIT & M
Dr.Harish G Assoc. Prof. Dr. AIT



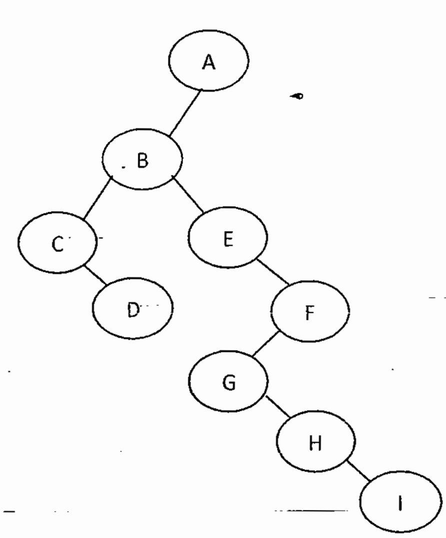
Step 2: Connect Siblings



Step 3: Delete the remaining branches



Step 4: The resulting binary tree is



Transforming / Converting a Forest into a Binary Tree

Definition: If $T_1, T_2, \ldots\ldots T_n$ is a forest of trees, then the binary tree corresponding to this forest is denoted by $B(T_1, T_2, \ldots\ldots T_n)$,

1) Is empty if n = 0

2) Has a root equal to root(T_1);

Has left subtree equal to $B(T_{11}, T_{12}, \ldots\ldots T_{1m})$, where $T_{11}, T_{12}, \ldots\ldots T_{1m}$ are subtrees of root(T_1)

Has right subtree equal to $B(T_2, \ldots\ldots T_n)$,

Procedure

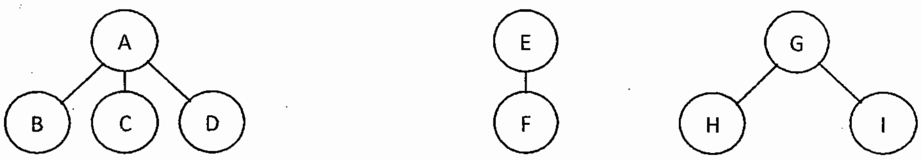
1) Convert each tree in the forest into a binary tree.

2) The root node of the first binary tree obtained is the root for the entire tree.

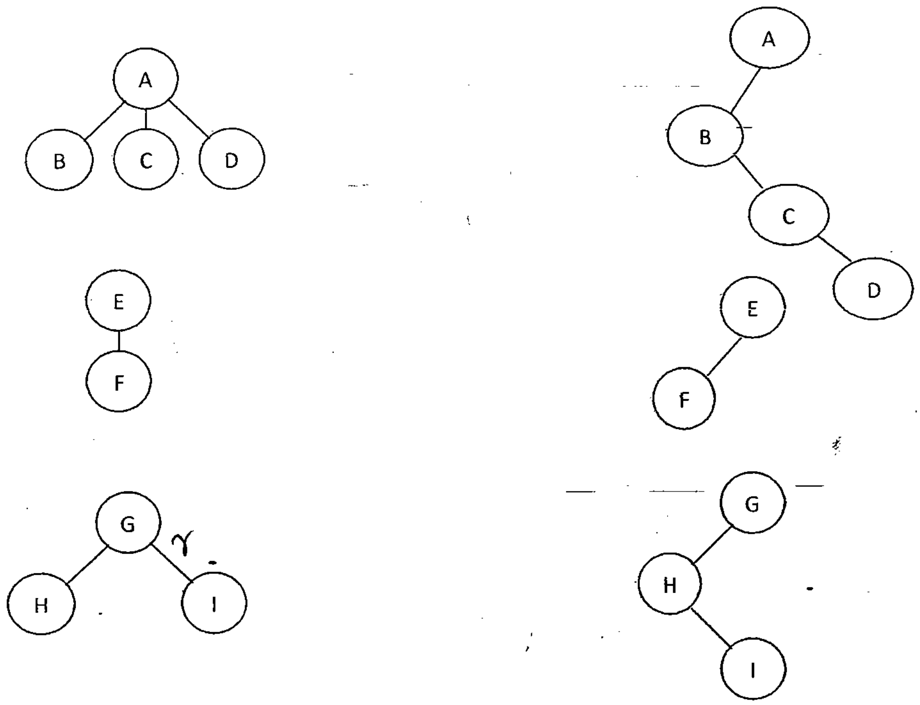
3) The root node of the second binary tree is attached as the right child of root node of first binary tree.

4) The root node of the third binary tree is attached as the right child of root node of second binary tree and so on.

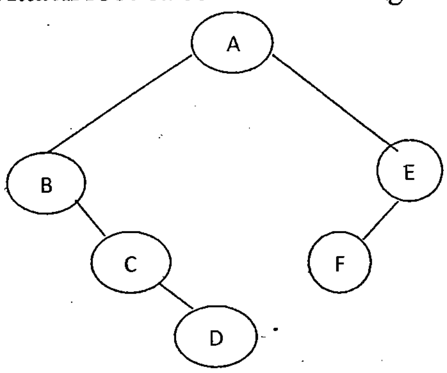
Example: Convert the following forest into a binary tree



Step 1: Convert each tree into a binary tree

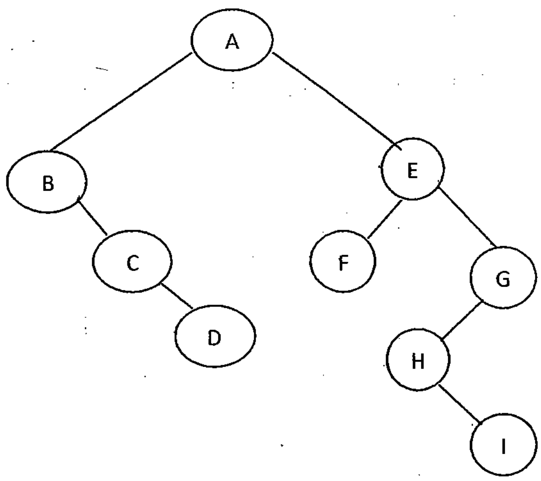


Step 2: Attach root of second tree as right child of root of first tree



Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Step 3: Attach root of third tree as right child of root of second tree



Creation of binary tree from preorder and inorder traversal

Note:

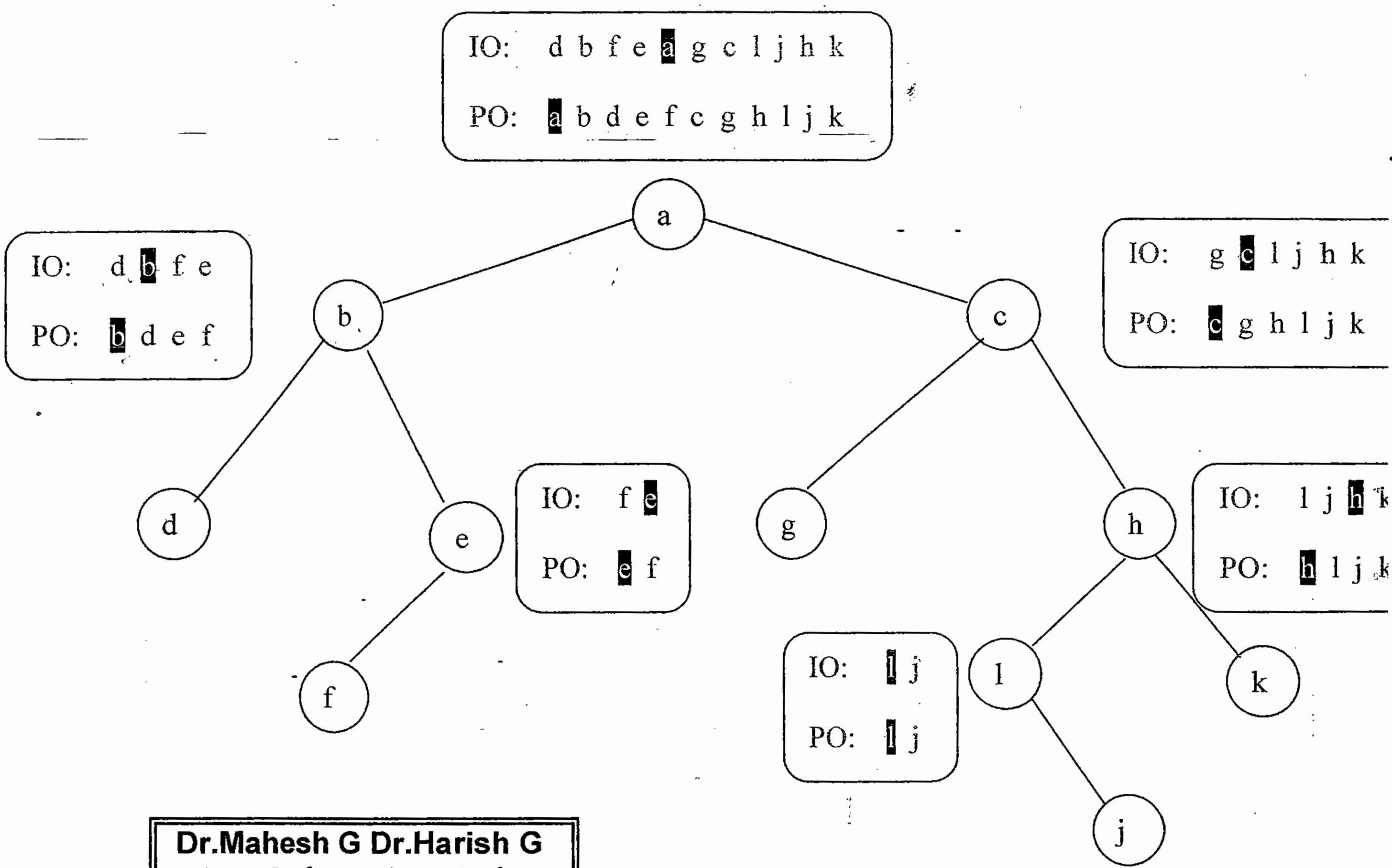
1) The first value in the preorder traversal gives the root of the tree.
2) In in-order traversal, initially the left subtree is traversed, then the root node and then the right subtree. Therefore, data before root node is the left subtree and data after the root node is the right subtree.

Example 1: Using the following inorder and preorder sequence, construct the binary tree.

Inorder : d b f e a g c l j h k

Preorder : a b d e f c g h l j k

Solution:



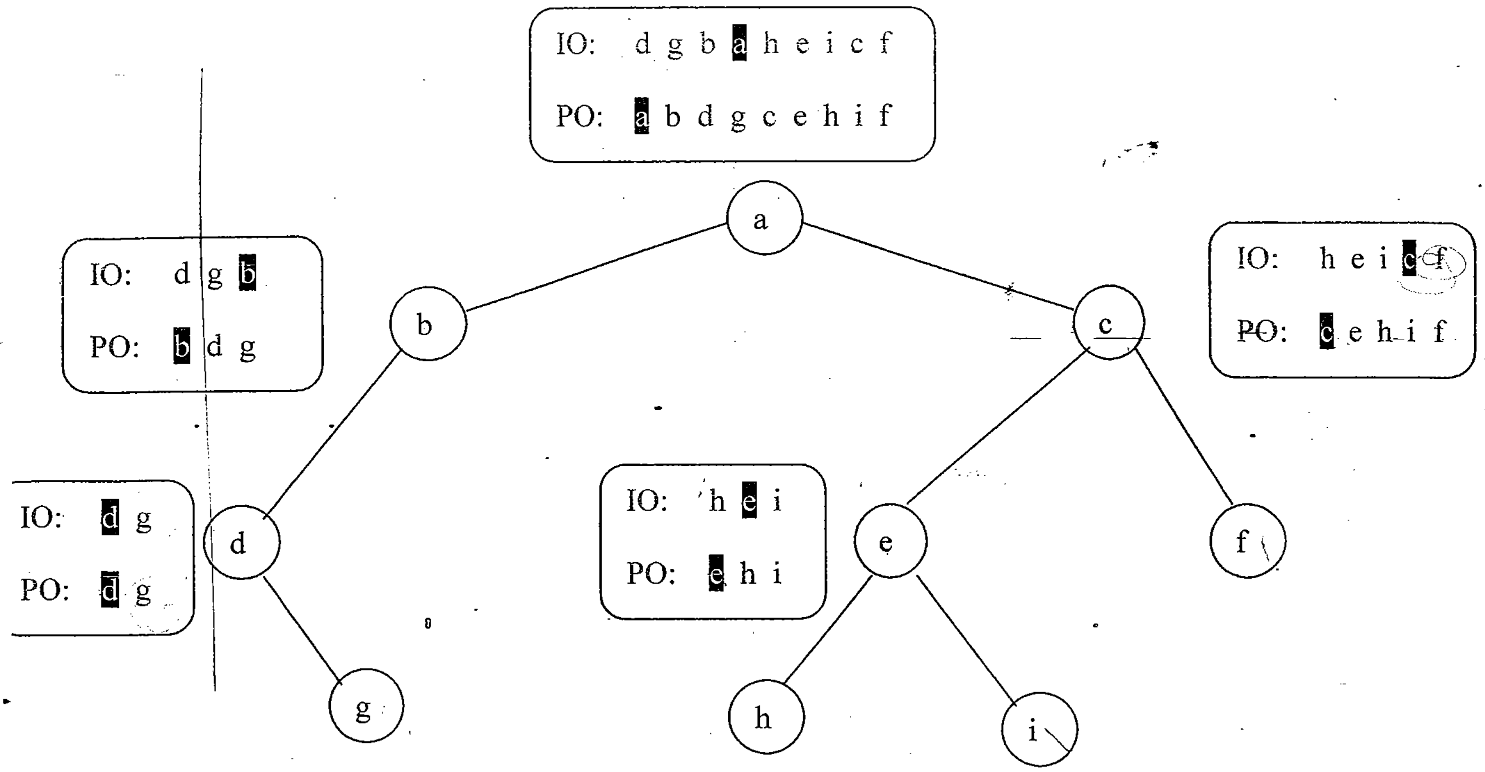
Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Example 2: Using the following inorder and preorder sequence, construct the binary tree.

Inorder : d g b a h e i c f

Preorder : a b d g c e h i f

Solution:



Dr.Mahesh G Assoc. Prof. BMSIT & M
Dr.Harish G Assoc. Prof. Dr. AIT

Creation of binary tree from postorder and inorder traversal

Note:

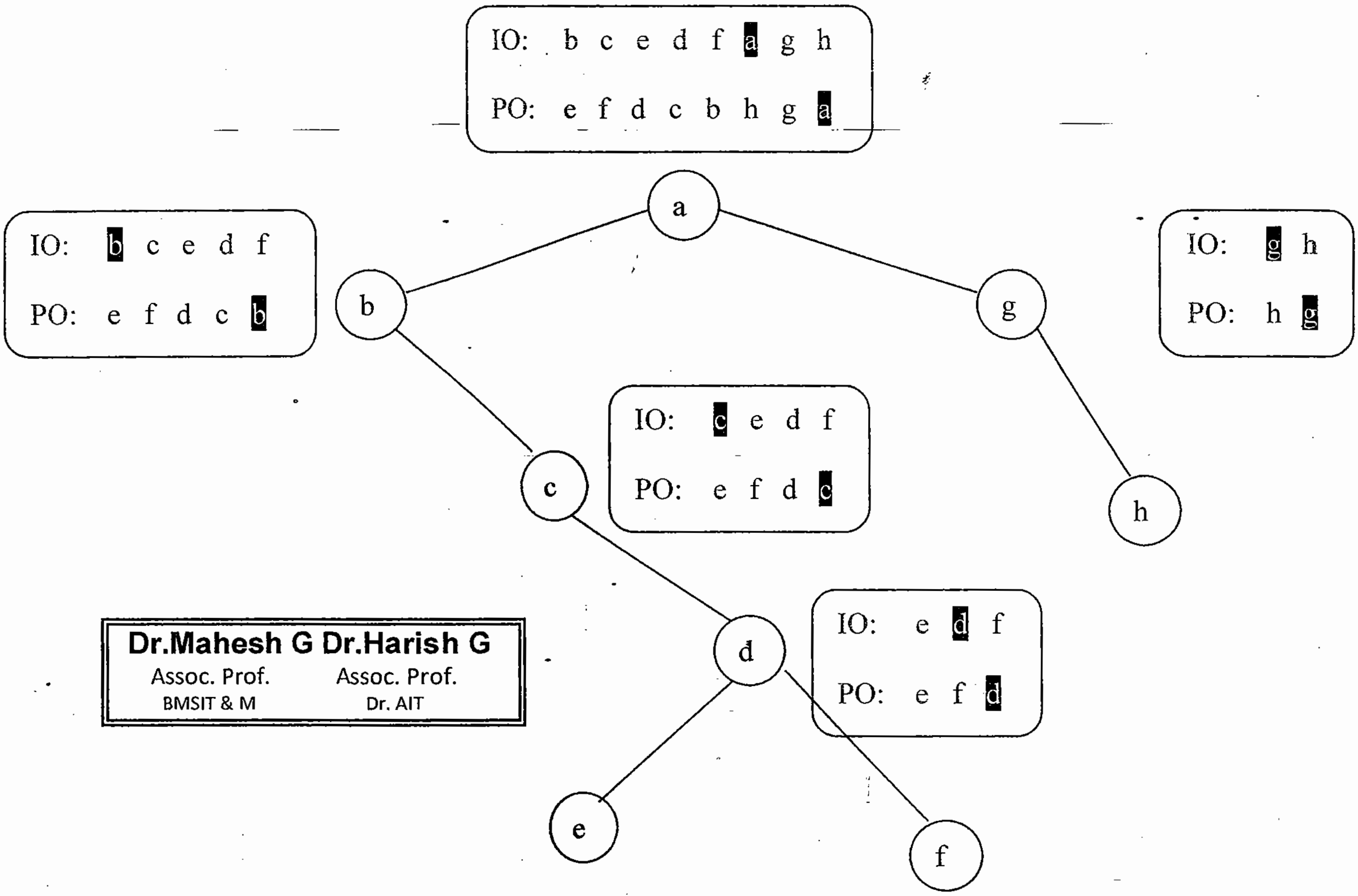
1) The last value in the postorder traversal gives the root of the tree.
2) In in-order traversal, initially the left subtree is traversed, then the root node and then the right subtree. Therefore, data before root node is the left subtree and data after the root node is the right subtree.

Example 1: Using the following inorder and postorder sequence, construct the binary tree.

Inorder : b c e d f a g h

Postorder : e f d c b h g a

Solution:



Threaded Binary Trees

Disadvantages of Binary Trees

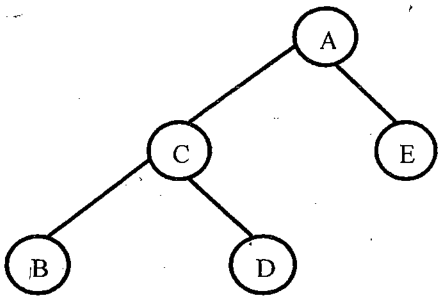
- ✓ In a binary tree, more than 50% of link fields have NULL values and hence more memory is wasted.
- ✓ Traversing a binary tree using iterative or recursive programs involves stack (explicit / implicit) and most of the traversal time is wasted on pushing and popping elements into the stack.
- ✓ Computations of predecessor and successor of a given node is time consuming.
- ✓ Only downward movement is possible in a binary tree.

Need for Threaded Binary Trees

Binary tree traversal algorithms are written using recursion or iteratively using the programmer written stack. Either way making use of implicit or explicit stack for each call makes the binary tree traversal inefficient, particularly if the tree must be traversed frequently.

The reason we use stack is that, at each step, we cannot access the next node in the sequence directly and we must use **backtracking.**

Consider the tree shown below and its inorder traversal is BCDAE .



Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

— Here we follow the left pointers to get the left most node B. after visiting 'B' since the B's right subtree is empty, we must go back to C (recursion or stack is used).
— Similarly after visiting 'D' since the D's right subtree is empty we go back to A (recursion or stack is used).
— The next node visited is A and then the right subtree node E.
From this example, it is clear that the nodes whose right subtree is empty uses recursion or stack for backtracking. This leads to the threaded concept which eliminates recursion and stacks and thus makes the traversal more efficient.

Threaded Concept

Definition: A threaded binary tree may be defined as follows:
A binary tree is threaded by making all right child pointers that would normally be null point to the inorder successor of the node, and all left child pointers that would normally be null point to the inorder predecessor of the node."
This pointer which points to the inorder successor / predecessor is called a thread and is represented by dotted lines to distinguish it from ordinary links.

Right in-threaded Binary Tree

Here each NULL right link is replaced by a special link (thread), to the in-order successor of that node.
Using right threads it is easy to do an inorder traversal of the tree since we need to only follow either an ordinary link or a thread to find the next node to visit.

Note:

1. The right thread from the last node will be made NULL (fig a) or points back to the header node if present (fig b).
2. Here an extra field rthread is used where

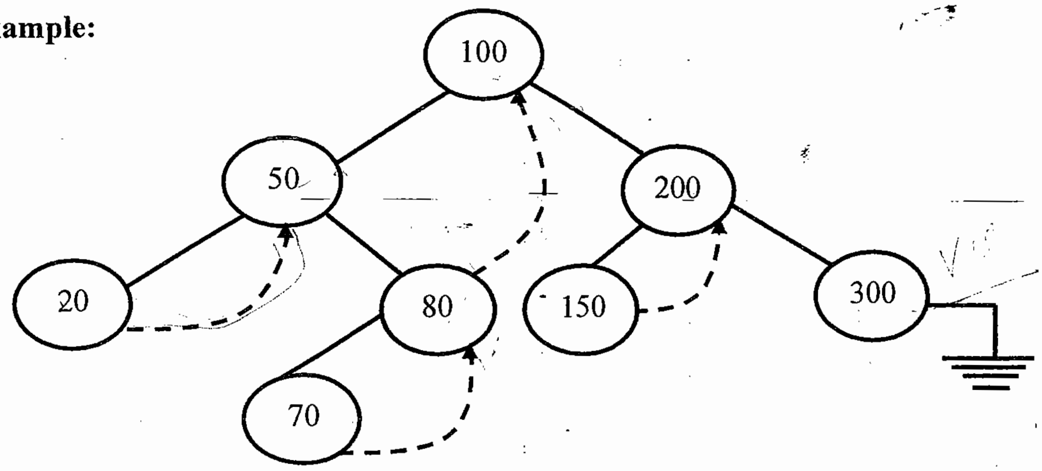
    if(rthread = = 1)

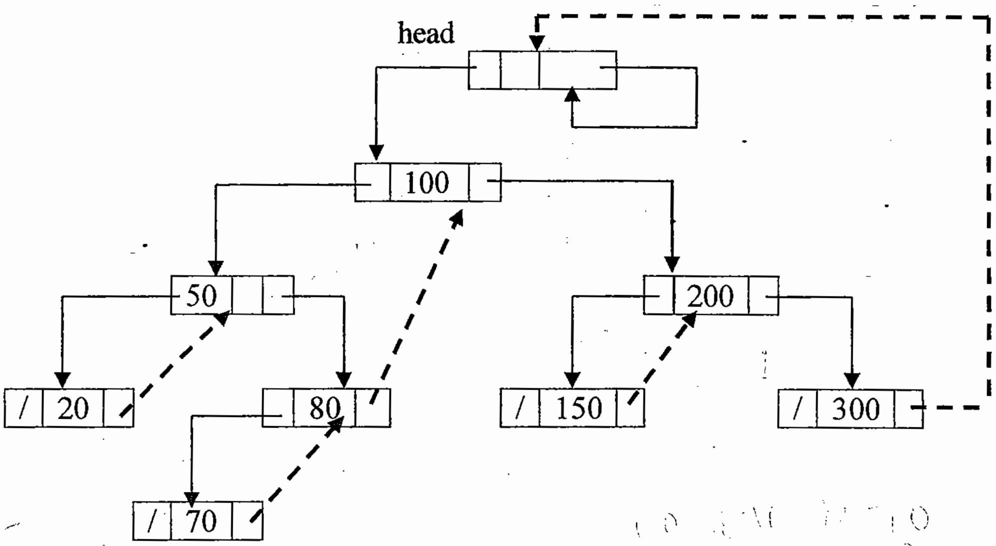
        rlink contains a thread

    otherwise if (rthread = =0) then

        rlink is an ordinary link connecting the right subtree.

Example:





The structure definition for right in-thread is shown below.

```
struct node
{
        int  info;
        struct node *llink;
        struct node *rlink;
        int rthread;
};
```

```
/*Function to find the inorder successor*/
NODE inorder_successor(NODE x)
{
        NODE temp;
        temp = x->rlink;
        if( x -> rthread = =1)
                return temp;
        while( temp -> llink ! = NULL)
                temp = temp -> llink;
        return temp;
}

/*Function to traverse the tree in inorder*/
void inorder(NODE head)
{
        if(head -> llink = = head)
        {
                printf("tree is empty");
                return;
        }
        cur = head;
        for(;;)
        {
                cur = inorder_successor(cur);
                if(cur = = head)  break;
                printf("%d", cur->info);
        }
}
```


Note: In main function we should create head and make head->rlink = head->rlink = head and head->rthread=0;

Left in-threaded Binary Tree

Here each NULL left link is replaced by a special link (thread) to the inorder predecessor of that node.

Note:

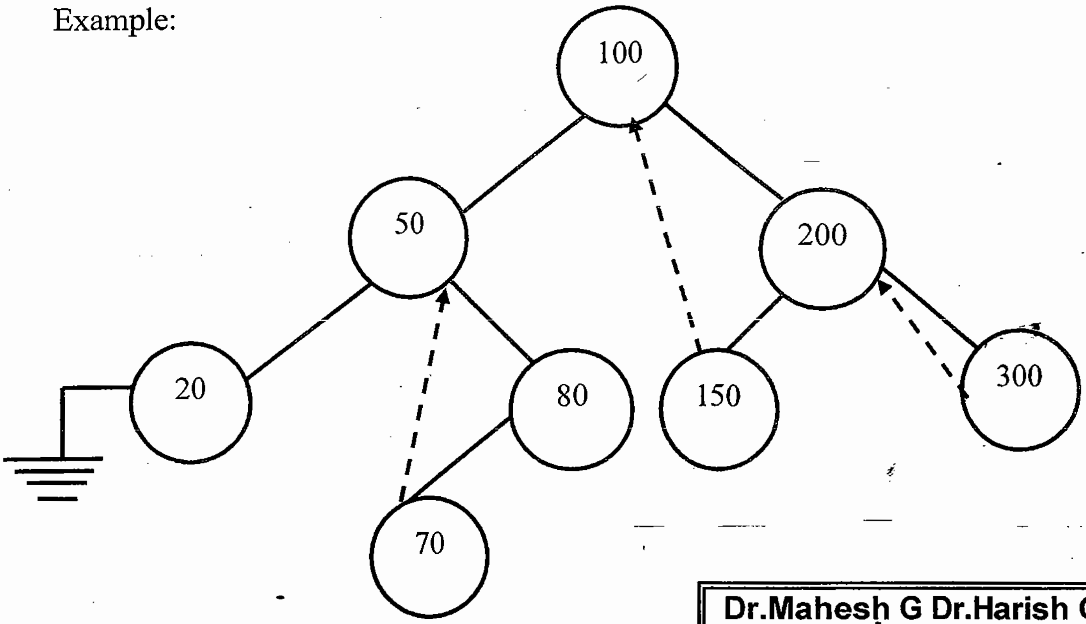
1. The left thread from the first node will be made NULL (fig a) or points back to the header node if present (fig b).
2. Here an extra field lthread is used where

```
        if(lthread = = 1)
                llink contains a thread
        otherwise if (lthread = =0) then
                llink is an ordinary link connecting the left subtree.
```

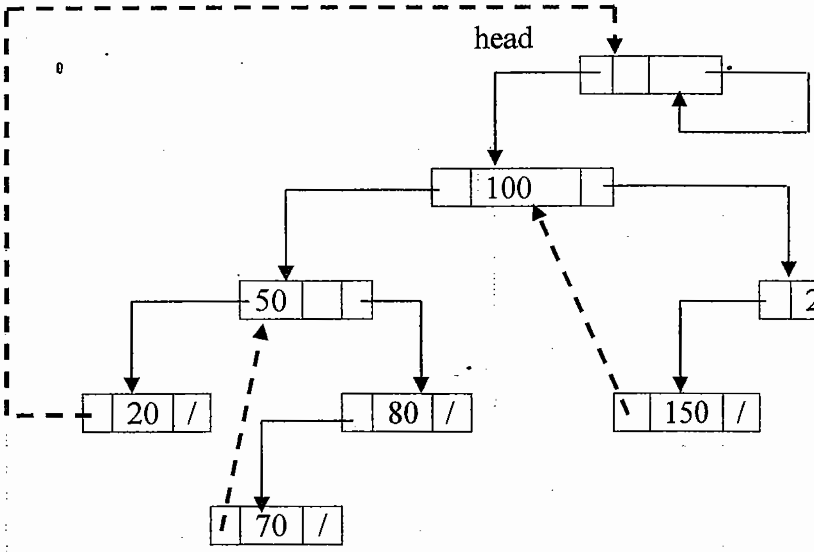
The structure definition for left in-thread is shown below.

```
Struct node
{
        int  info;
        struct node *llink;
        struct node *rlink;
        int lthread;
};
```

Example:



Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT



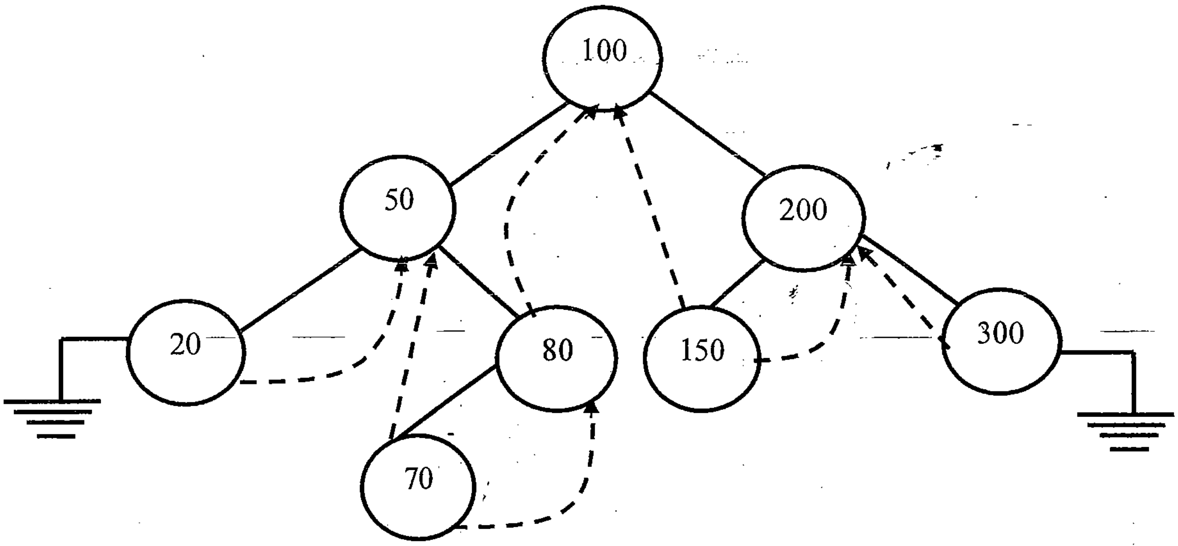

```
/*Function to find the inorder predecessor*/
NODE inorder_predecessor(NODE x)
{
        NODE temp;
        temp=x->llink;
        if(x->lthread = =1)
                return temp;
        while(temp->rlink!=NULL)
                temp=temp->rlink;
        return temp;
}
```

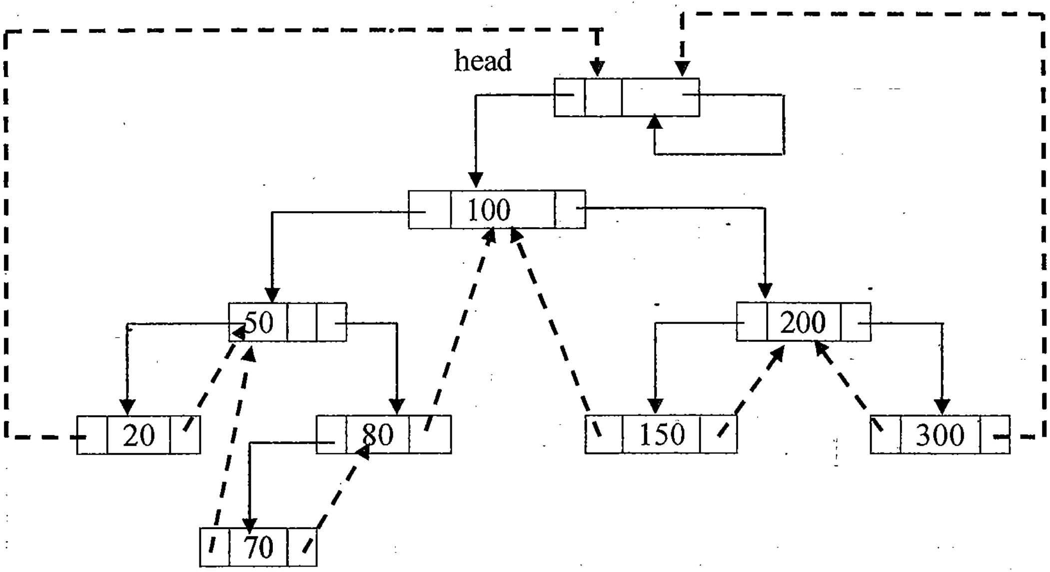
Fully in-threaded or In-threaded Binary Tree

If both left link and right link are used for threading, then it is called in-threaded binary tree. Here if a node has

rlink = NULL → replace NULL by the address of the in-order successor of that node.

llink = NULL → replace NULL by the address of the in-order predecessor of that node.





1. The left thread from the first node (no predecessor) and the right thread from the last node (no successor) will be made NULL or points back to the header node if present.
2. Here 2 extra fields lthread and rthread is used.

The structure definition is shown below.

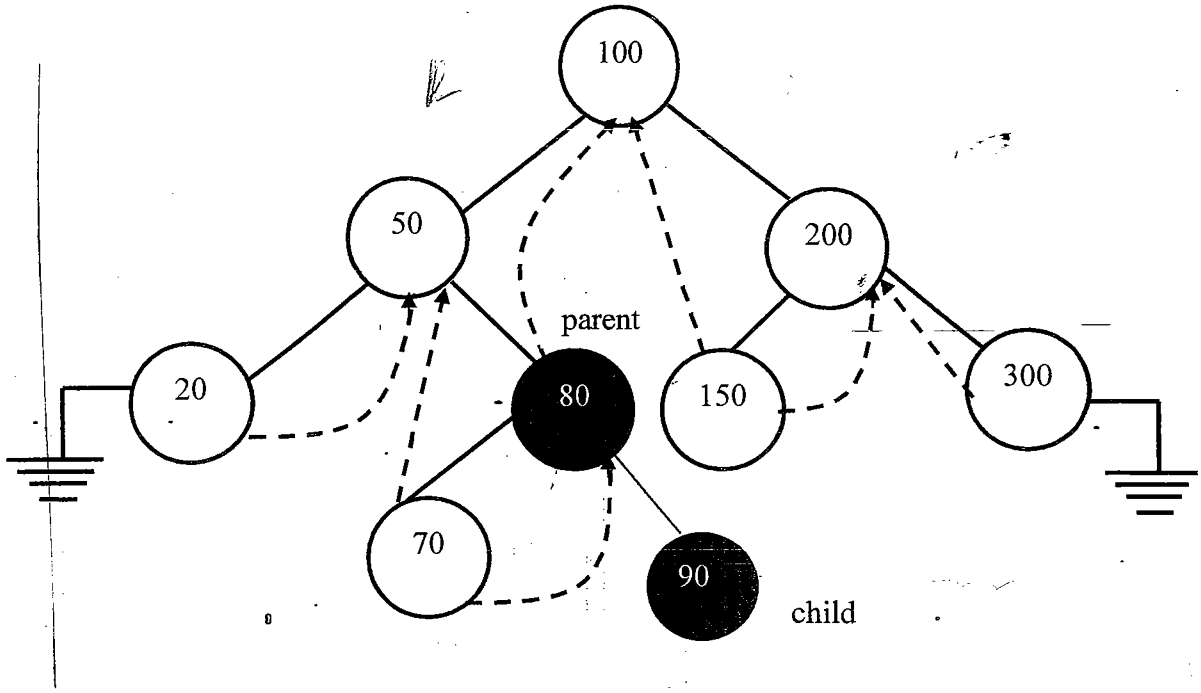
```
Struct node
{
        int info;
        struct node *llink;
        struct node *rlink;
        int lthread;
        int rthread;
};
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Inserting a node into a In-Threaded Binary Tree

Function insert_~~right~~ that inserts a new node, child, as the right child of a node parent in a in-threaded binary tree.

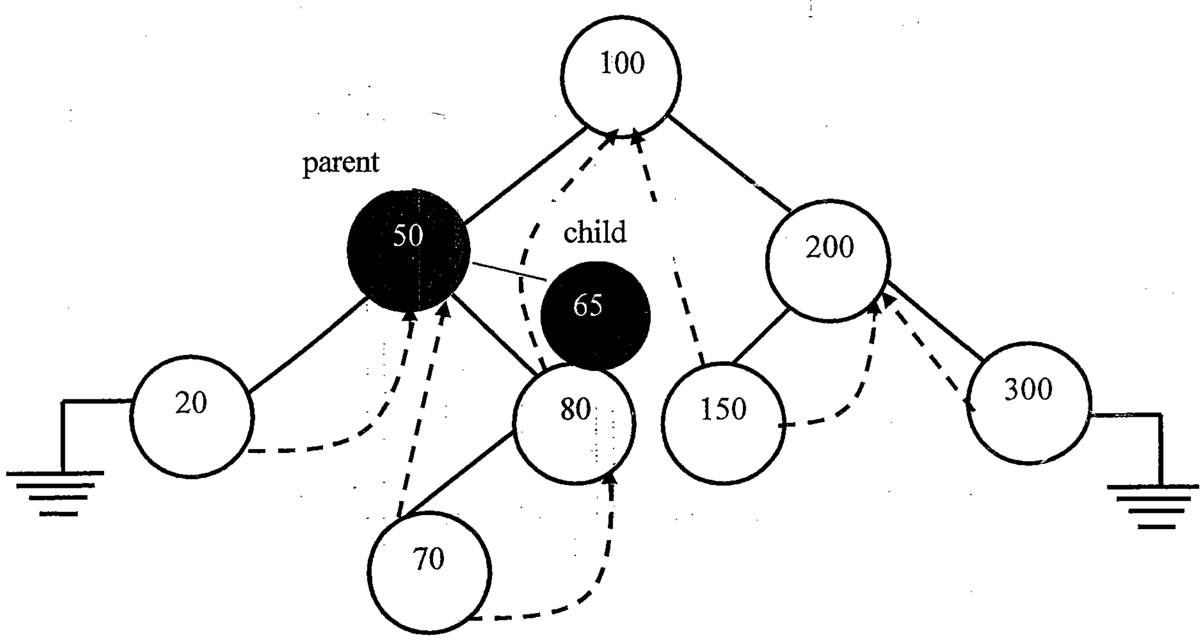
Case 1: If parent has an empty right sub tree



child->rlink = parent ->rlink;
child->rthread = parent->rthread;
child->llink = parent;
child->lthread = 1;
parent->rlink = child;
parent->rthread = 0;

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Case 2: If parent has a right subtree



- ✓ After insertion, child becomes the inorder predecessor of a node (suc) that has lthread =1 which has to be updated to point to child.
- ✓ The node suc was previously the inorder successor of parent

```
If(child->rthread==0)
{
        suc = inordersuc(child);
        suc->llink = child;
}
```

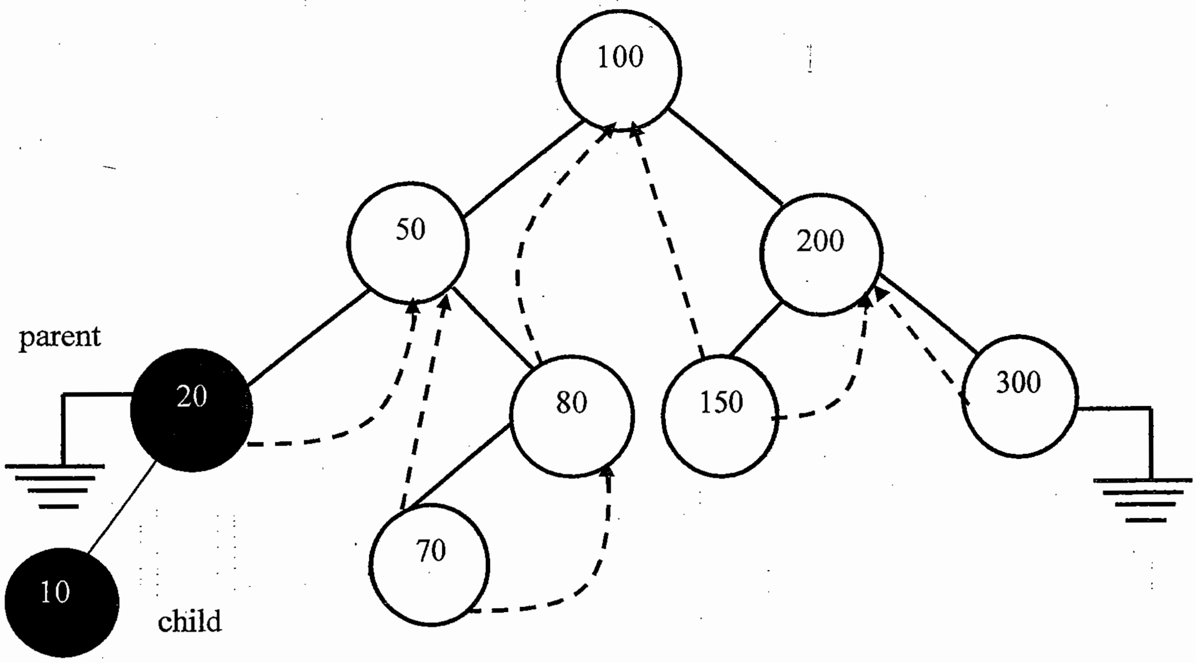
The complete code for insertion is

```
void insert_right(NODE parent, NODE child)
{
        NODE suc;
        child->rlink = parent ->rlink;
        child->rthread = parent->rthread;
        child->llink = parent;
        child->lthread = 1;
        parent->rlink = child;
        parent->rthread = 0;

        If(child->rthread==0)
        {
                suc = inordersuc(child);   in order succes of child
                suc->llink = child;
        }

}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Function insert_left, that inserts a new node, child, as the left child of a node parent in a in-threaded binary tree.

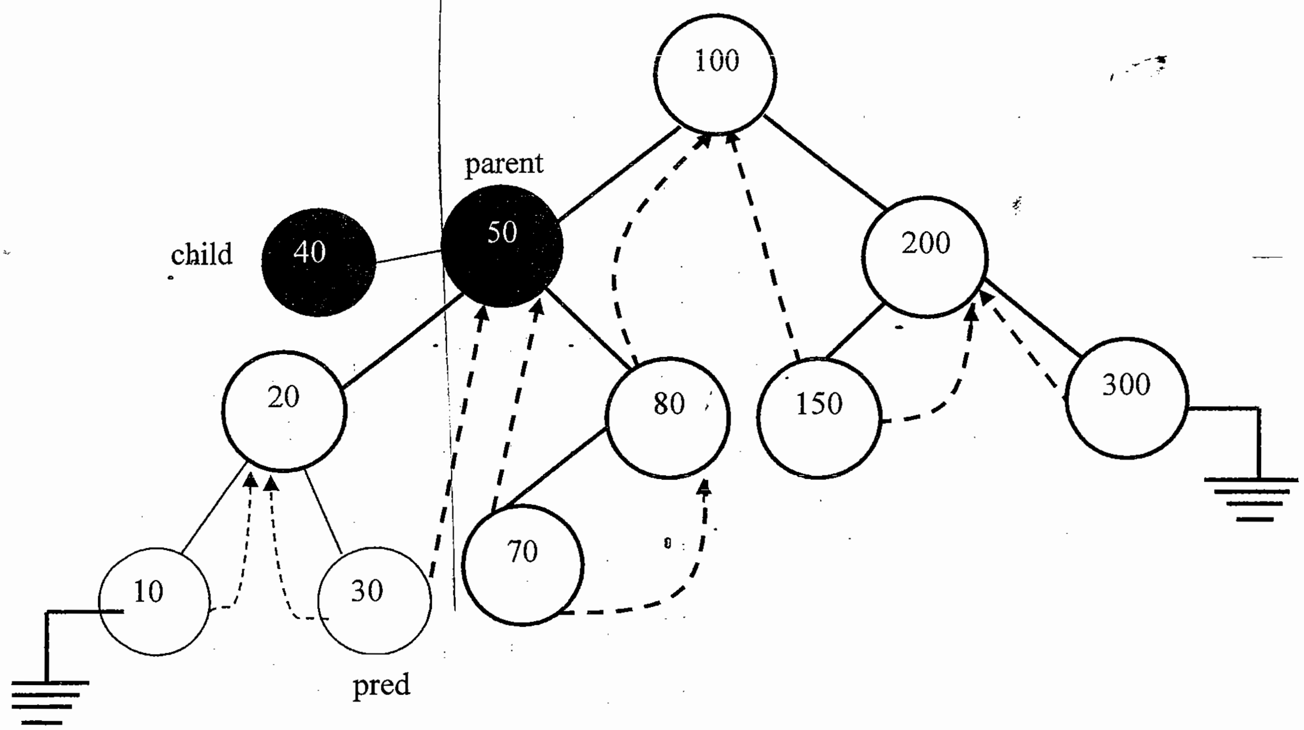
Case 1: If parent has an empty left sub tree



```
child->llink = parent ->llink;
child->lthread = parent->lthread;
child->rlink = parent;
child->rthread = 1;
parent->llink = child;
parent->lthread = 0;
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Case 2: If parent has a left subtree



- ✓ After insertion, child becomes the inorder successor of a node (pred) that has rthread =1 which has to be updated to point to child.
- ✓ The node pred was previously the inorder predecessor of parent

```
If(child->lthread==0)
{
        pred = inorderpred(child);
        pred->rlink = child;
}
```

The complete code for insertion is

```
void insert_left(NODE parent, NODE child)
{
        NODE suc;
        child->llink = parent ->llink;
        child->lthread = parent->lthread;
        child->rlink = parent;
        child->rthread = 1;
```

```
        parent->llink = child;
        parent->lthread = 0;

        If(child->lthread==0)
        {
                pred = inorderpred(child);
                pred->rlink = child;
        }
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Advantages

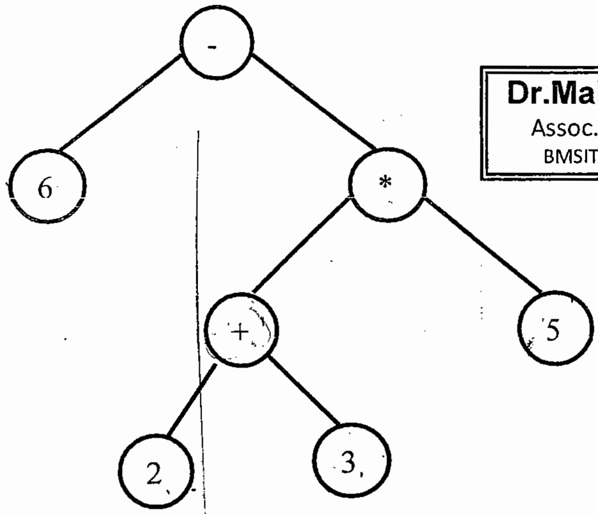
1. Faster traversal and less memory usage during traversal, since no stack need to be maintained.
2. Greater generality, since one can go from a node to its successor or predecessor (traversing nodes), and this simplifies algorithms that require moving forward and backward in a tree [unthreaded binary tree].

Disadvantages

1. Two extra fields per node are needed to check whether a link is a thread or not, hence requires more memory space.
2. Insertion and deletion operations are time consuming.

Expression Tree - Expression tree is a binary tree in which each internal node corresponds to operator and each leaf node corresponds to operand.

Example – For the expression 6 – (2 + 3) * 5, the expression tree is



Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Pre-Order (N L R)	In-Order (L N R)	Post-Order (L R N)
$- 6_P *_P$	$6_I - *_I$	$6_P *_P -$
$- 6 *_P$	$6 - *_I$	$6 *_P -$
$- 6 * +_P 5_P$	$6 - +_I * 5_I$	$6 +_P 5_P * -$
$- 6 * + 2_P 3_P 5_P$	$6 - 2_I + 3_I * 5_I$	$6\ 2_P 3_P + 5_P * -$
$- 6 * + 2\ 3_P 5_P$	$6 - 2 + 3_I * 5_I$	$6\ 2\ 3_P + 5_P * -$
$- 6 * + 2\ 3\ 5_P$	$6 - 2 + 3 * 5_I$	$6\ 2\ 3 + 5_P * -$
- 6 * + 2 3 5 (Prefix Expression)	6 - 2 + 3 * 5 (Infix Expression)	6 2 3 + 5 * -(Postfix Expression)

Note:

1) Inorder Traversing of an expression tree gives Infix expression
2) Preorder Traversing of an expression tree gives Prefix expression
3) Postorder Traversing of an expression tree gives Postfix expression

Creation of Expression Tree for postfix expression

```
While( not end of input)
{
        Obtain the next input symbol

        if(symbol is an operand)
        {
                Create a node (temp) with info field = symbol        //operand
                push(node(temp), top, s)
        }
```

```
        else
        {
                Create a node (temp) with info field = symbol        //operator
                temp -> rlink = pop(top, s);
                temp ->llink = pop(top, s);
                push(node (temp), top, s);
        }
}
return( pop(top, s));    // return the root node of the tree
```

Evaluation of Postfix Expression using Expression Tree

Step 1: If root->llink = = root->rlink = = NULL

return root->info

Step 2: If root->info is operator

return evaluate(root->llink) operator evaluate(root->rlink)

Program to create an expression tree for a postfix expression and evaluate the same

```
#include<stdio.h>
#define stacksize 50

struct node
{
        int info;
        struct node *llink;
        struct node *rlink;
};
typedef struct node* NODE;


void push(NODE item, int *top, NODE s[ ])
{
        if(*top = = stacksize -1)
        {
                printf("Stack overflow\n");
                return;
        }
        *top = *top + 1;
        s[*top] = item;
}

NODE pop(int *top, NODE s[ ])
{
        NODE item;
        if(*top = = -1)
        {
                return -1;
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

```
        }
        item = s[*top];
        *top = *top - 1;
        return(item);
}

NODE create_exp_tree(char postfix[])
{
        NODE s[stacksize], temp;
        int i, n, top;
        char symbol;
        top = -1;
        n = strlen(postfix);
        for(i=0; i<n; i++)
        {
            symbol = postfix[i];

            switch(symbol)
            {
                case '+':

                case '-':

                case '*':

                case '/':

                case '%':

                case '^':
                case '$': temp = (NODE) malloc(sizeof(struct node));
                          temp -> info = symbol;
                          temp -> rlink= pop(&top, s);
                          temp -> llink= pop(&top, s);
                          push(temp, &top, s);
                          break;


                default: temp = (NODE) malloc(sizeof(struct node));
                          temp -> info = symbol - '0';
                          temp -> rlink= NULL;
                          temp -> llink= NULL;
                          push(temp, &top, s);
            }
        }
        return(pop(&top, s));
}
```

```
int evaluate(NODE root)
{
        if(root->llink = = NULL && root->rlink = = NULL)

                return root->info;

        if(root->info  = = '+')
                return (evaluate(root->llink) + evaluate(root->rlink) );
        else if(root->info  = = '-')
                return (evaluate(root->llink) - evaluate(root->rlink) );
        else if(root->info  = = '*')
                return (evaluate(root->llink) * evaluate(root->rlink) );
        else if(root->info  = = '/')
                return (evaluate(root->llink) / evaluate(root->rlink) );
        else if(root->info  = = '%')
                return (evaluate(root->llink) % evaluate(root->rlink) );
        else if(root->info  = = '^')
                return ((int)pow((double)evaluate(root->llink) , (double)evaluate(root->rlink) ) );
}

void main( )
{
        int res;
        NODE root = NULL;
        char postfix[70];

        clrscr();

        printf("enter the postfix expression\n");
        scanf("%s",postfix);

        root = create_exp_tree(postfix);

        res=evaluate(root);

        printf("the value of the postfix exp is %d", res);

        getch( );
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

AI assignment
DMS ~~assignment~~
DS record
~~ADE~~ 〃
DS assignment

6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2