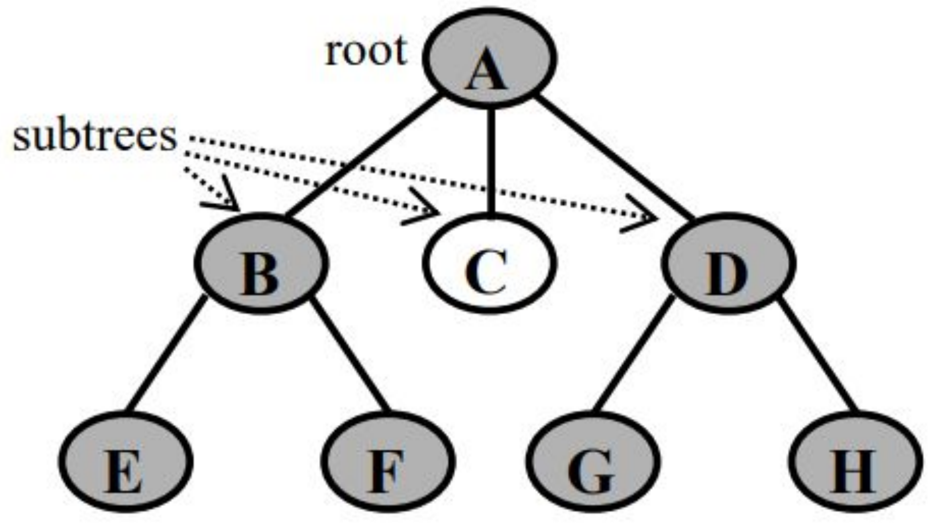
TREES

Definition: A tree is a set of finite set of one or more nodes that shows parent-child relation such that:

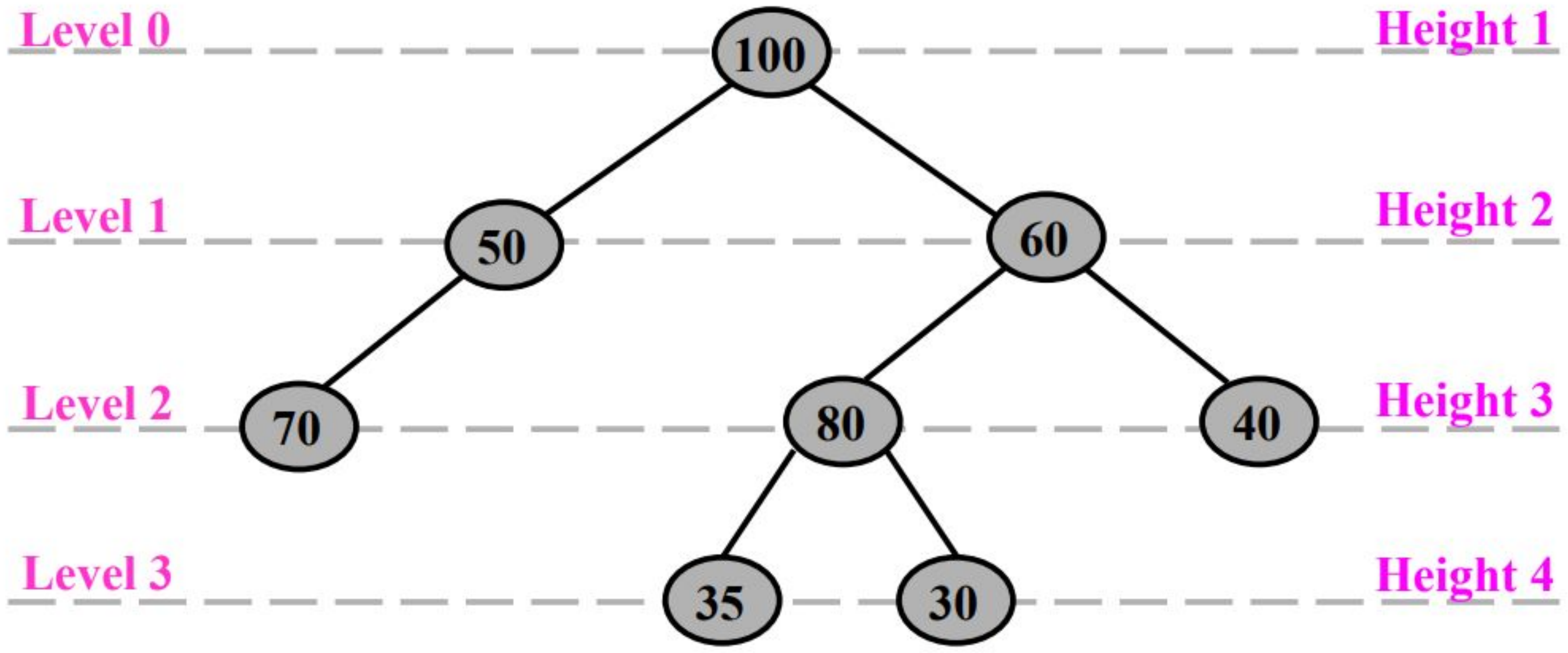
- There is a special node called the root node
- The remaining nodes are partitioned into disjoint subsets T_1, T_2, T_3 ……T_n, $n \geq 0$ where T_1, T_2, T_3 ……T_n which are all children of root node are themselves trees called *subtrees*.

Ex1 : Consider the following tree. Let us identify the root node and various subtrees:



- The tree has 8 nodes : A, B, C, D, E, F, G and H.
- The node A is the *root* of the tree
- We normally draw the trees with root at the top
- The node B, C and D are the children of node A and hence there are 3 subtrees identified by B, C and D

- The node A is the parent of B,C and D whereas D is the parent of G and H



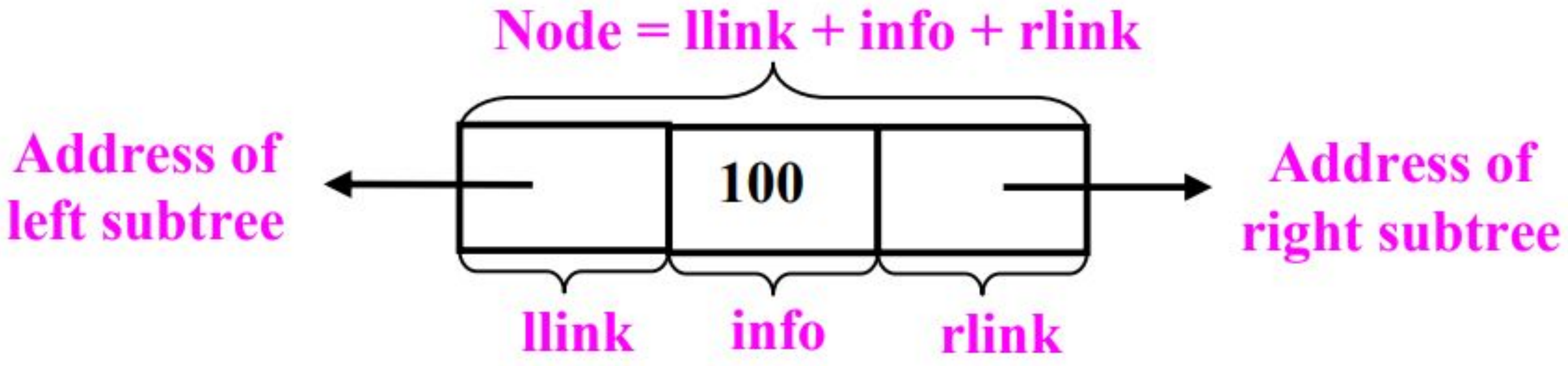
Number of levels = 4

Maximum height/depth of the tree = 4

Definition: A binary tree is a tree which has finite set of nodes that is either empty or consist of a root and two subtrees called left subtree and right subtree. A binary tree can be partitioned into three subgroups namely root, left subtree and right subtree.

- **Root** – If tree is not empty, the first node in the tree is called root node.
- **left subtree** – It is a tree which is connected to the left of root. Since this tree comes towards left of root, it is called left subtree.
- **right subtree** – It is a tree which is connected to the right of root. Since this tree comes towards right of root, it is called right subtree.

Note: In general, a tree in which each node has either zero, one or two subtrees is called a binary tree. The pictorial representation of a typical node in a binary tree is shown below:



Lemma: a) The maximum number of nodes on level *i* of a binary tree = 2^i for $i \geq 0$

b) The maximum number of nodes in a binary tree of depth $k = 2^k - 1$

Proof: Consider the following complete binary tree and observe the following factors from the complete binary tree:

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

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

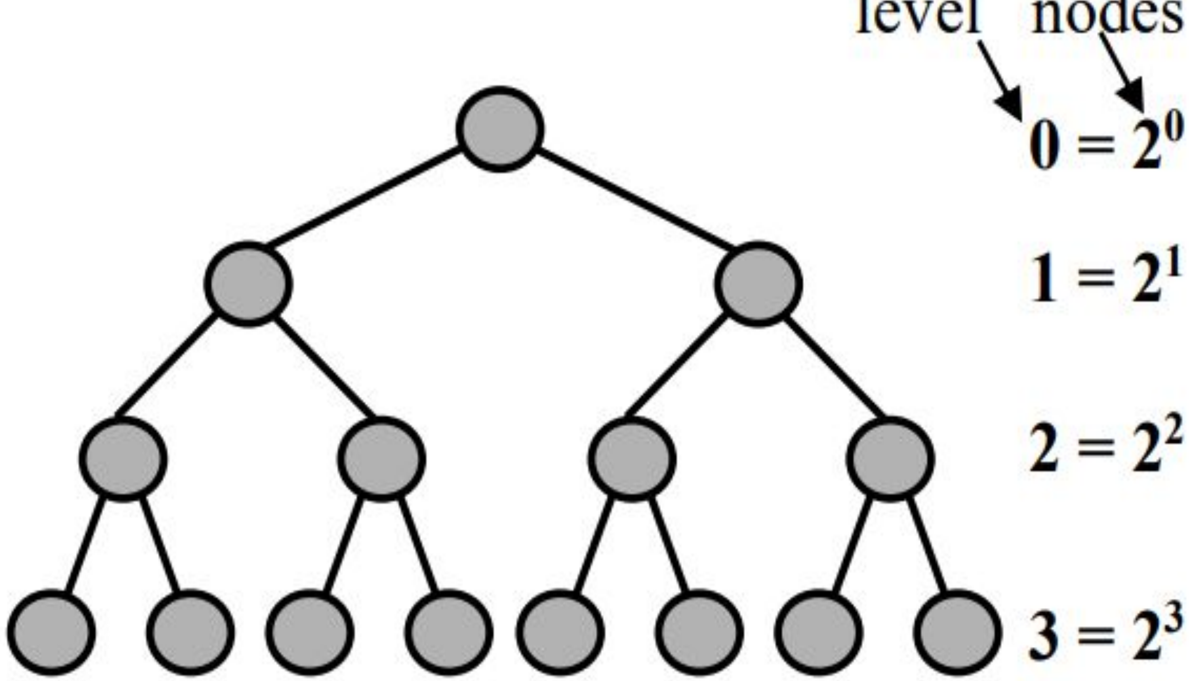
Number of nodes at level 2 = 4 = 2^2

Number of nodes at level 3 = 8 = 2^3

....

....

Number of nodes at level ***i*** = 2^i.



Total number of nodes in the full binary tree of level $i = 2^0 + 2^1 + 2^2 + \ldots\ldots 2^i$.

Types of binary trees

- Strictly binary tree (Full binary tree)
- Skewed tree
- Complete binary trees
- An expression tree
- Binary search trees

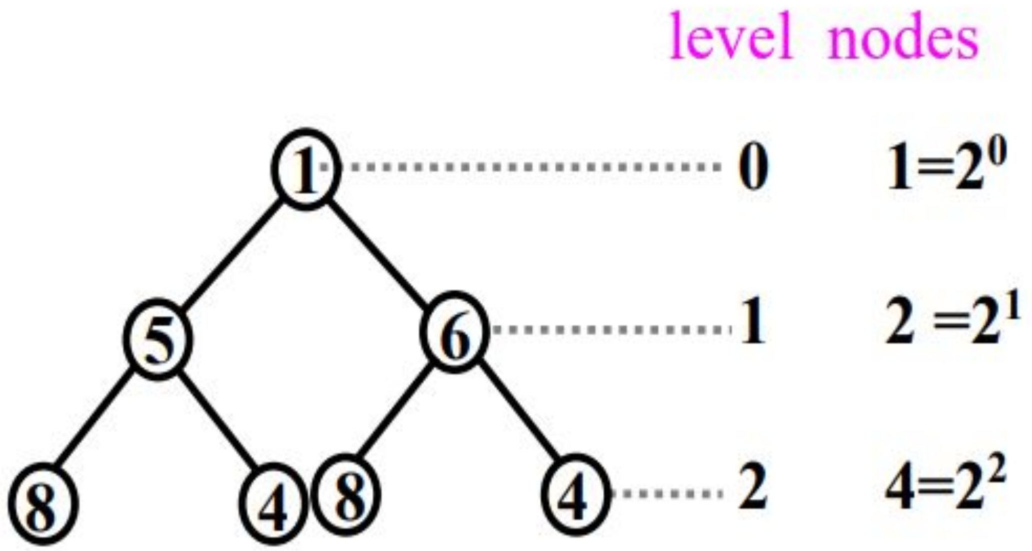
Strictly(Full) binary tree

Now, let us see "What is strictly binary tree?"

Definition: A binary tree having 2^i nodes in any given level i is called strictly binary tree. Here, every node other than the leaf node has two children. A strictly binary tree is also called *full binary tree* or *proper binary tree*.

For example, in full binary tree shown,

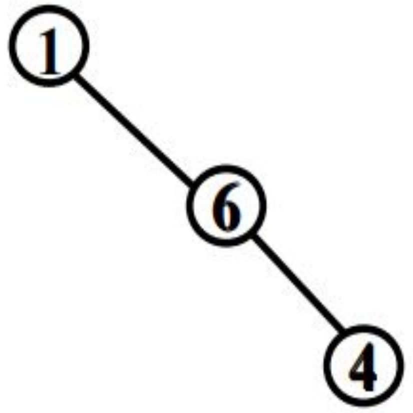
- The number of nodes at level $0 = 2^0 = 1$
- The number of nodes at level $1 = 2^1 = 2$
- The number of nodes at level $2 = 2^2 = 4$



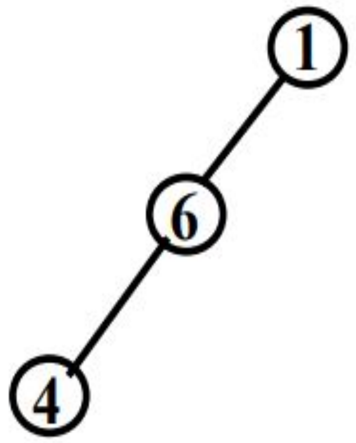
Skewed tree

Now, let us see "What is skewed tree?"

Definition: A skewed tree is a tree consisting of only left subtree or only right subtree. A tree with only left subtrees is called left skewed binary tree and a tree with only right subtree is called right skewed binary tree.



Right skewed tree

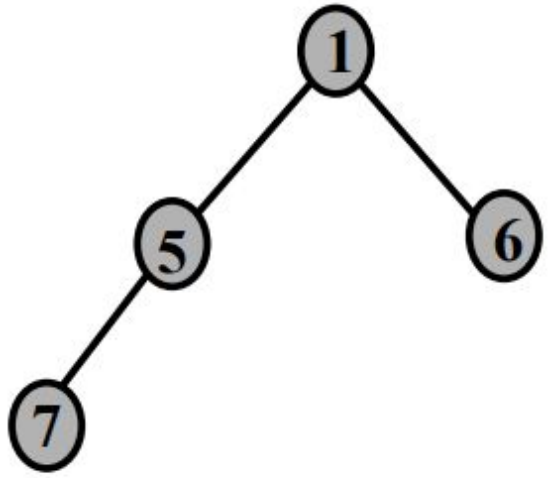


Left skewed tree

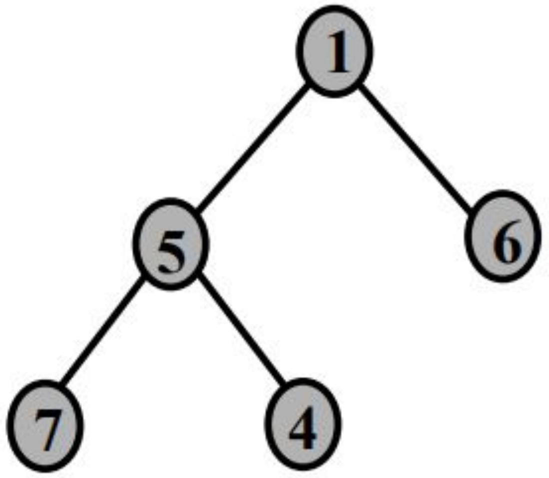
Complete binary tree

Now, let us see "What is a complete binary tree?"

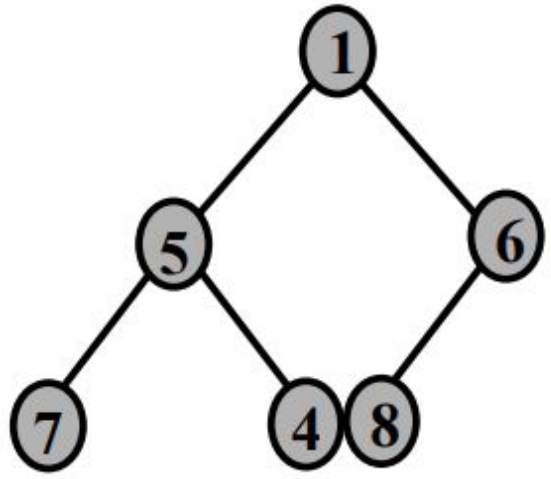
Definition: A complete binary tree is a binary tree in which every level, *except possibly the last* level is completely filled. If the nodes in the last level are not completely filled, then all the nodes in that level should be filled only from left to right. For example, all the trees shown below are complete binary trees.



(a)

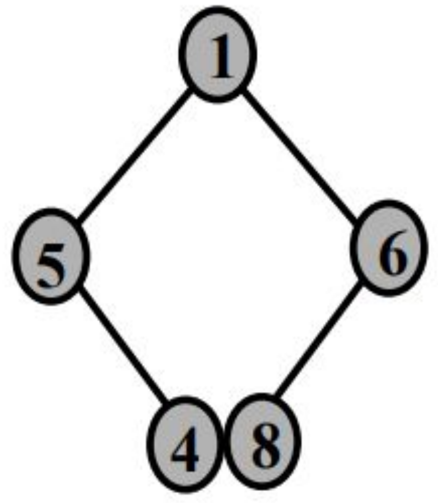


(b)

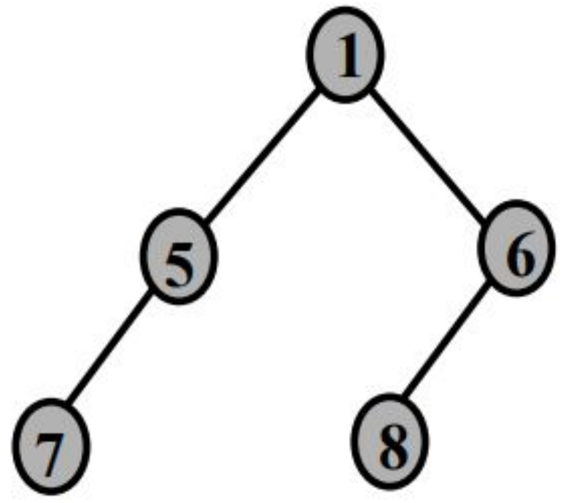


(c)

The tree shown below is not complete binary tree since the nodes in the last level are not filled from left to right. There is an empty left child for node 5 in figure (d) and there is an empty right child for node 5 in figure (e). All the nodes should be as left as possible.



(d)



(e)

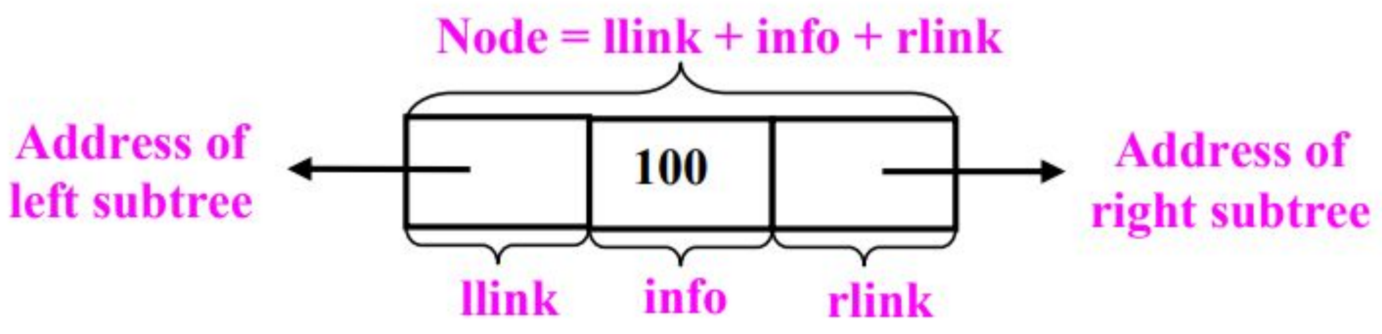
linked representation, a node in a tree has three fields:

- **info** – which contains the actual information
- **llink** – which contains address of the left subtree
- **rlink** – contains address of the right subtree.

So, a node can be represented using self-referential structure as shown below:

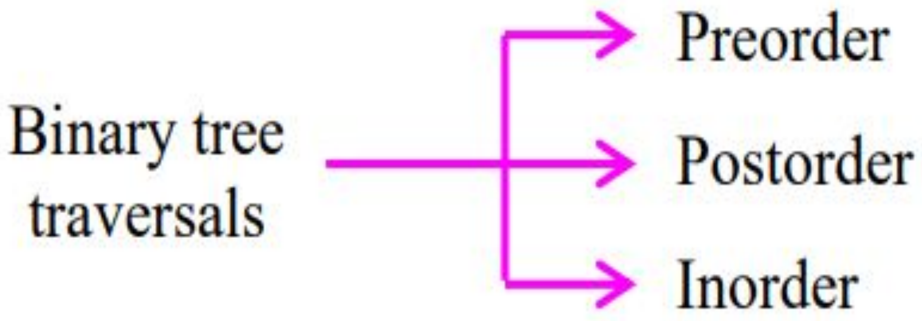
```
struct node
{
        int             info;
        struct node     *llink;
        struct node     *rlink;
};

typedef struct node*  NODE;
```

The pictorial representation of a typical node in a binary tree is shown below:



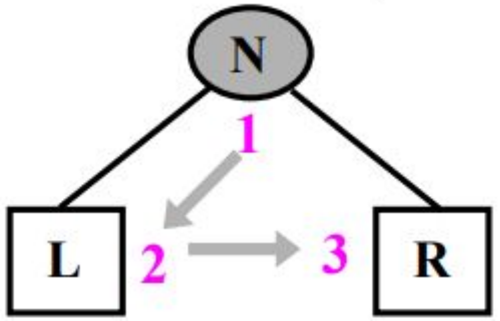
Definition: Traversing is a method of visiting each node of a tree exactly once in a systematic order. During traversal, we may print the **info** field of each node visited.

Now, let us see "What are the different traversal techniques of a binary tree?" The various traversal techniques of a binary tree are shown below:

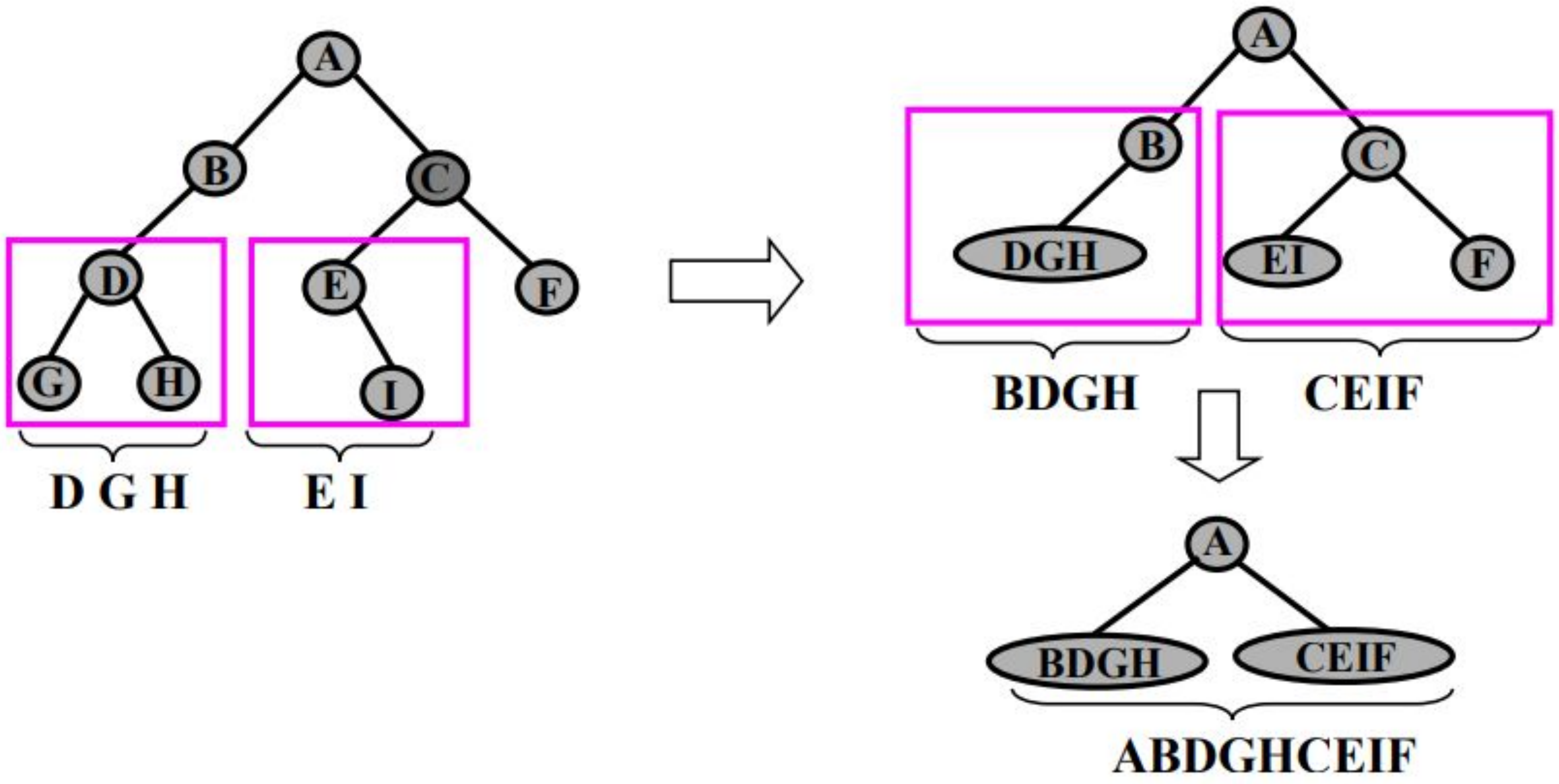


Definition: The preorder traversal of a binary tree can be recursively defined as follows:

1. Process the root Node **[N]**
2. Traverse the **L**eft subtree in preorder **[L]**
3. Traverse the **R**ight subtree in preorder **[R]**

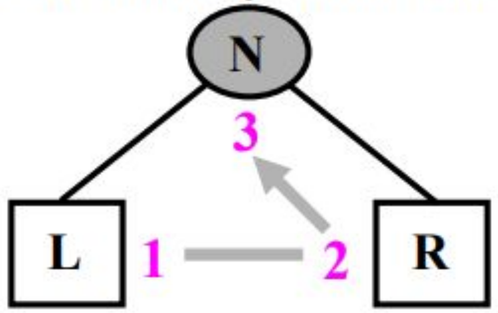


For example, let us traverse the following tree in preorder.

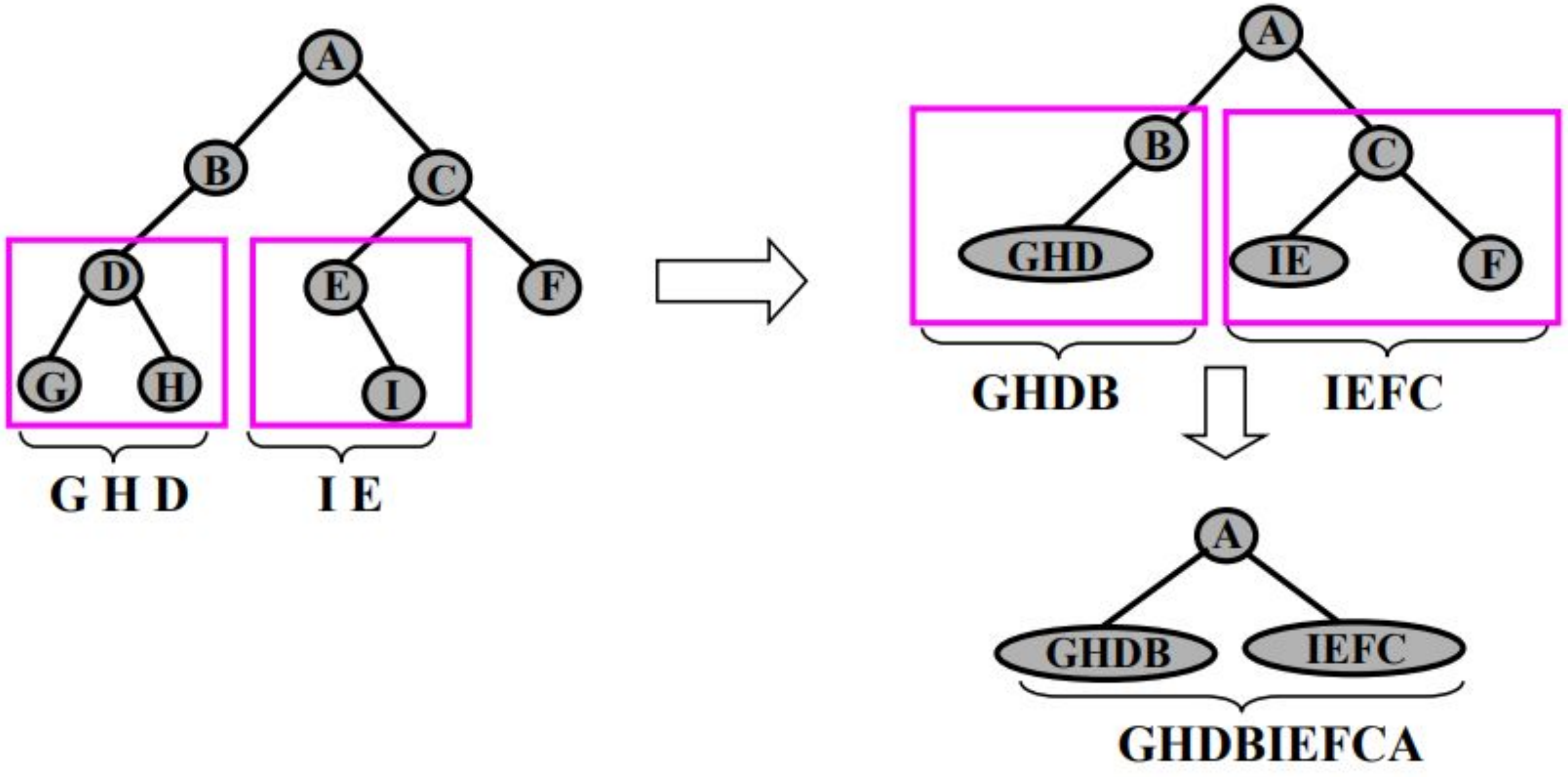


Definition: The postorder traversal of a binary tree can be recursively defined as follows:

1. Traverse the **L**eft subtree in postorder **[L]**
2. Traverse the **R**ight subtree in postorder **[R]**
3. Process the root **N**ode **[N]**

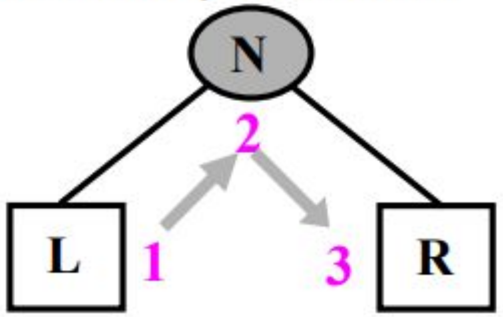


For example, let us traverse the following tree in preorder.

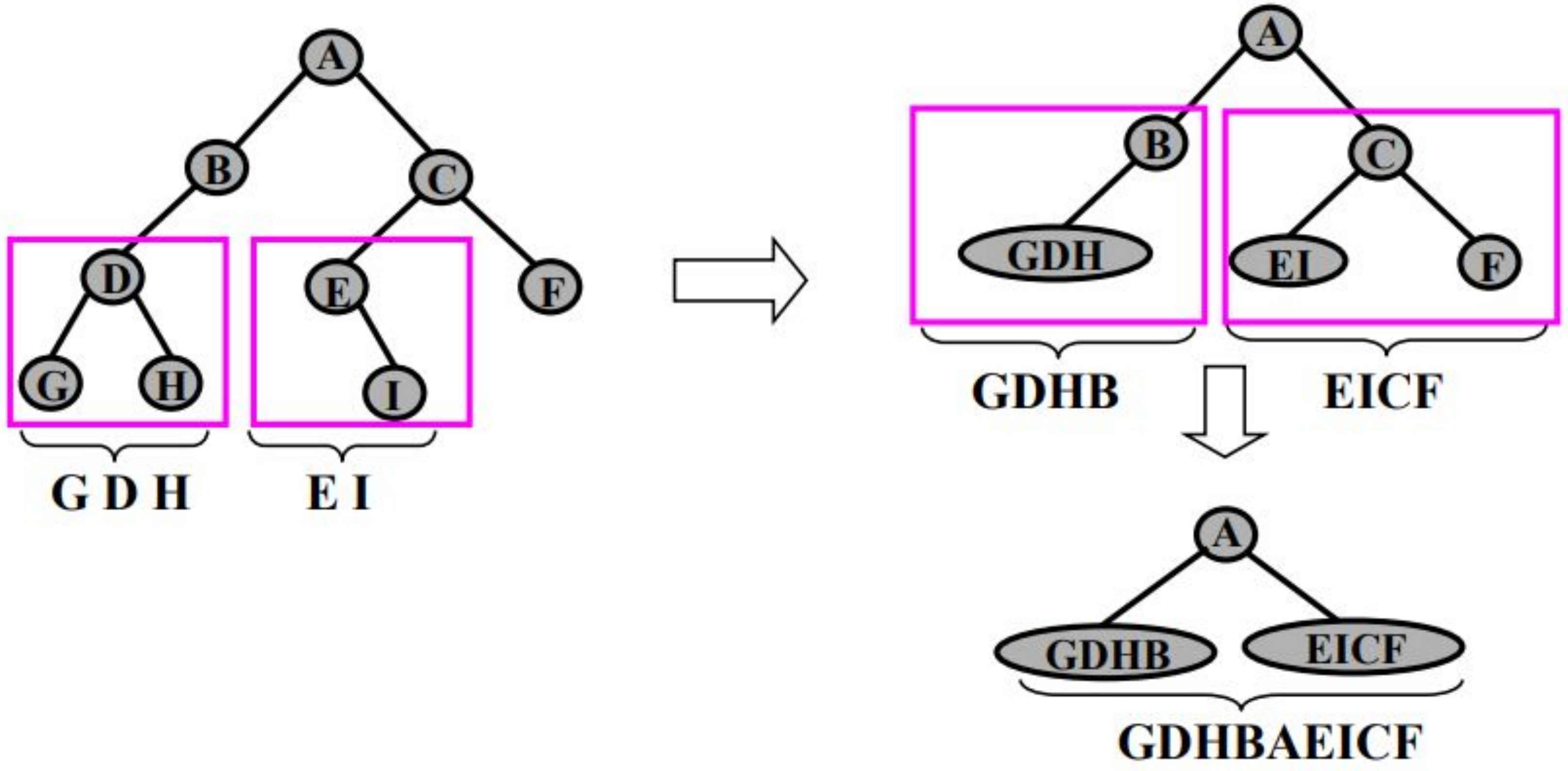


Definition: The inorder traversal of a binary tree can be recursively defined as follows:

1. Traverse the **L**eft subtree in inorder **[L]**
2. Process the root **N**ode **[N]**
3. Traverse the **R**ight subtree in inorder **[R]**



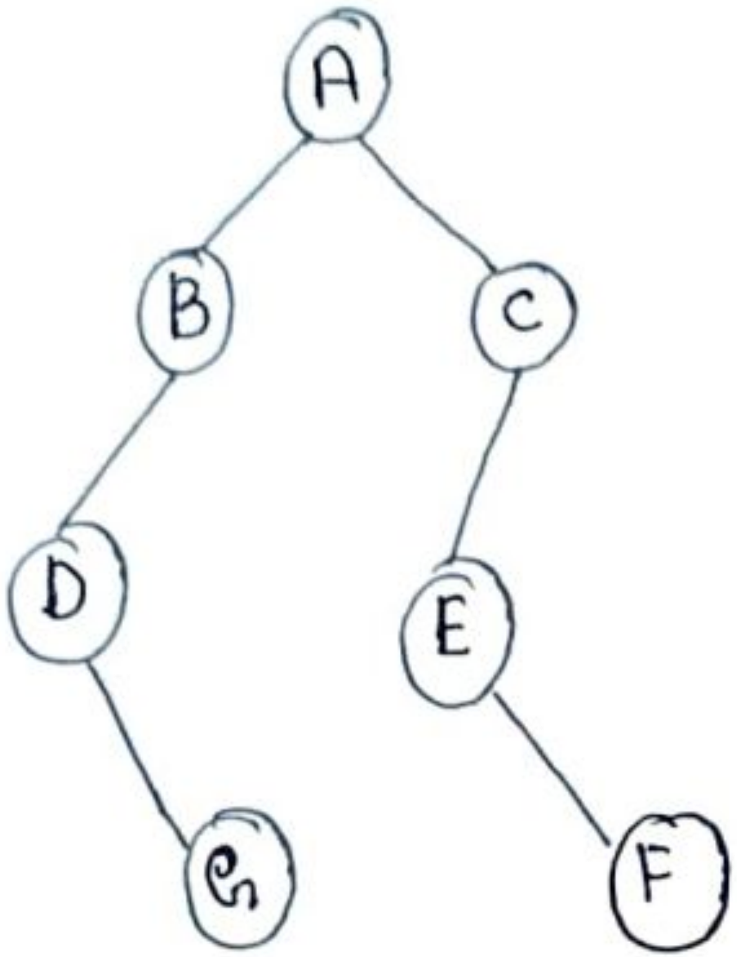
For example, let us traverse the following tree in preorder.

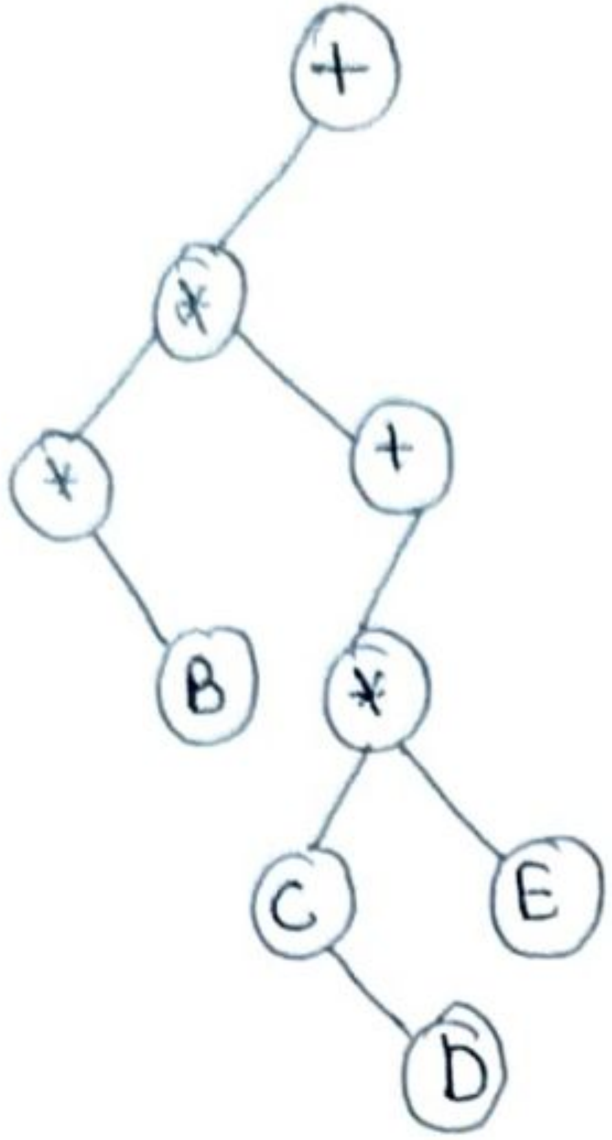


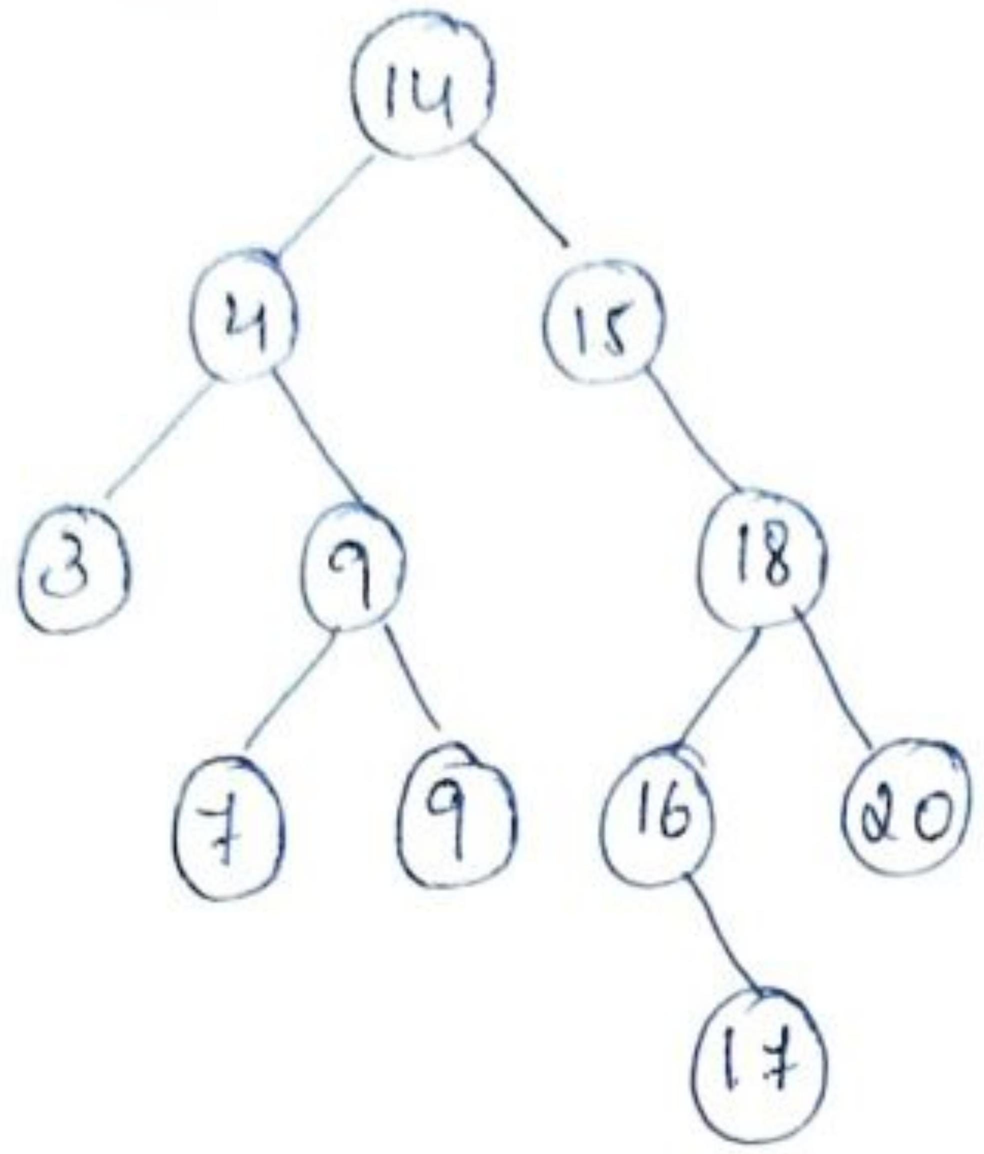
```
void preorder(NODE root)
{
        if ( root == NULL ) return;

        printf("%d ",root->info);          /* visit the node */
        preorder(root->llink);             /* visit left subtree in preorder*/
        preorder(root->rlink);             /* visit right subtree in preorder*/
}
```

```
void postorder(NODE root)
{
        if ( root == NULL ) return;

        postorder(root->llink);            /* visit leftsubtree in postorder*/
        postorder(root->rlink);            /* visit rightsubtree in postorder*/
        printf("%d ", root->info);         /* visit the node */
}
```

```
void inorder(NODE root)
{
        if ( root == NULL ) return;

        inorder(root->llink);              /* visit left-subtree in inorder */
        printf("%d ",root->info);          /* visit the node */
        inorder(root->rlink);              /* visit right-subtree in inorder */
}
```


A
B
C
D
E
G
F


+
*
*
+
B
*
C
E
D


14
4
15
3
9
18
7
9
16
20
17