TREES :

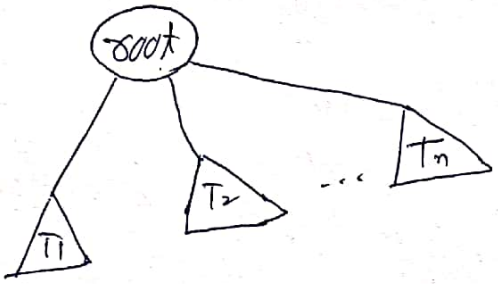
DS 25

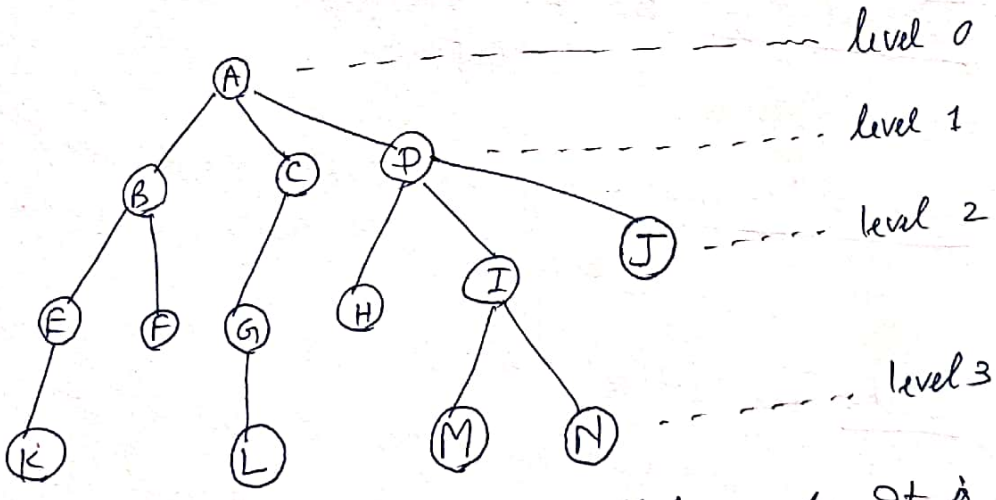
Basic concepts :-

* Arrays, stacks, queues and linked lists are linear data structures
* Trees and graphs are non-linear data structures.

Def^n (Tree) : A tree is an acyclic, simple, connected graph.

* It is a collection of nodes.
* The collection may be empty ($= \phi$).
* It has a special node "r", called the root and zero or more non-empty subtrees $T_1, T_2, T_3, \cdots, T_n$.
* The root of each subtree is said to a child of root, and root is the parent of each subtree.

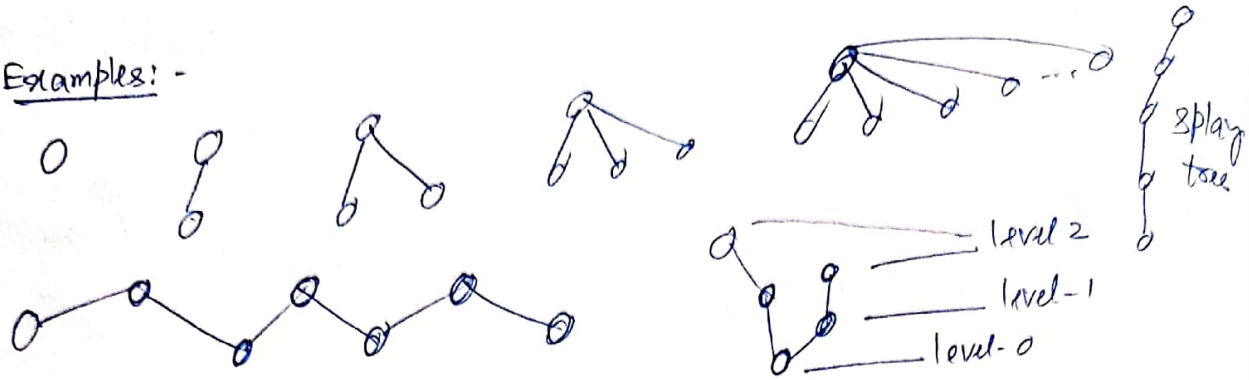




Node — Each element of a tree is called a node. It is the basic structure in a tree. It has some value and links (branches) to other nodes (child).

Root: It is the first node in the hierarchical arrangement of data items. It's level is zero.

Examples:-



Parent: The immediate predecessor of a node is its parent.

Child: Each immediate successor of a node is its child.

Sibling: The child nodes of same parent node are called siblings.

Degree of a node: The number of subtrees of a node in a tree is called degree of that node.

Degree of a tree: The maximum degree of nodes in a given tree is called the degree of the tree.

Terminal Node: A node with degree zero is called a terminal node or a leaf.

Non-terminal node:- Any node (except the root node) whose degree is not zero is called non-terminal node.

Level: The entire tree structure is levelled in such a way that the root node is always at level 0. If a node has level-i then its child nodes are at level (i+1).

Edge: It connects two nodes in a tree. (parent to child)

Path: It is the sequence of consecutive edges from the source node to the destination node.

Depth: The depth of a node(n_i) is the length of the unique path from root node $\xrightarrow{to}$ n_i. That is depth of root node = 0.

Height: The height of a node(n_i) is the longest path from the node(n_i) $\xrightarrow{to}$ a leaf node.

That is leaves are at height 0.

* Height of (tree) = Height of (Root).

Ancestor and Descendant: If there is a path from $n_1 \xrightarrow{to} n_2$, then n_1 is an ancestor of n_2 and n_2 is a descendant of n_1.

BINARY TREE

A binary tree "T" is a non-linear data structure consists of finite set of elements called nodes such that

i) either it is empty (null tree, empty tree)

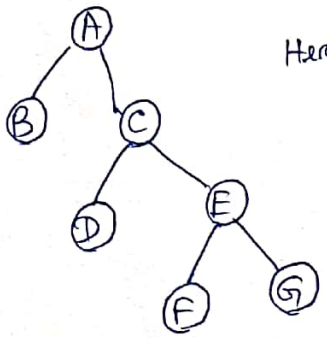
or ii) it has a distinguished node ie root node "R", and the remaining nodes of T form an ordered pair of disjoint binary trees T_1 and T_2 which are called left subtree and right subtree respectively.

Strictly Binary Tree :

If every non-leaf node in a binary tree has non-empty left and right subtrees, the tree is termed as a strictly binary tree.

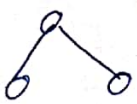
* if (no. of leaves $= n$) $\Rightarrow$ No. of nodes $= 2n - 1$

i)



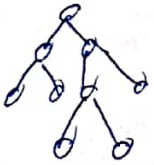
Here leaves = 4, nodes = $2 \times 4 - 1 = 7$

ii)



Here leaves = 2, nodes = $2 \times 2 - 1 = 3$

iii)



leaves = 5, nodes = $2 \times 5 - 1 = 9$

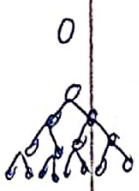
DS 26

Complete Binary Tree :-

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

i) if (a binary tree contains m nodes at level l)

⇒ No. of nodes at level l+1 = $\lceil 2m \rceil, \lfloor 2m \rfloor, 2m$



No. of nodes at level l+i = $\lceil 2^i m \rceil, \lceil 2^{l+i} \rceil$

ii) Total no. of nodes in a complete binary tree

$= 2^0 + 2^1 + 2^2 + \cdots + 2^d$

$= \sum_{k=0}^{d} 2^k$

$= 2^{d+1} - 1$

$= \frac{1(1-2^{d+1})}{1-2} = 2^{d+1} - 1.$

Formula: Geometric Series

iii) But total nodes in a complete binary tree = 8.

What is its depth.

But depth = d.

⇒ No of nodes = $2^{d+1} - 1 = 8$

⇒ $2^{d+1} = 8+1$

⇒ $2^d = \frac{8+1}{2}$

⇒ $\log_2(2^d) = \log_2(8+1) - \log_2 2$

⇒ $d = \log_2(8+1) - 1$

Example

∴ if no. of nodes = 15

⇒ Depth = $\log_2(15+1) - 1 = \log_2 16 - 1 = 4 - 1 = 3$ (Ans)

9.6

Almost Complete Binary Tree

A binary tree of depth "d" is an almost complete binary tree if i) Each leaf in the tree is either at level "d" or at level "d-1".

ii) For any node (n_i) in the tree with a right descendant at level "d", all the left descendants of (n_i) that are leaves are also at level d.

(i.e all the levels should be filled up from left to right).

Example:-

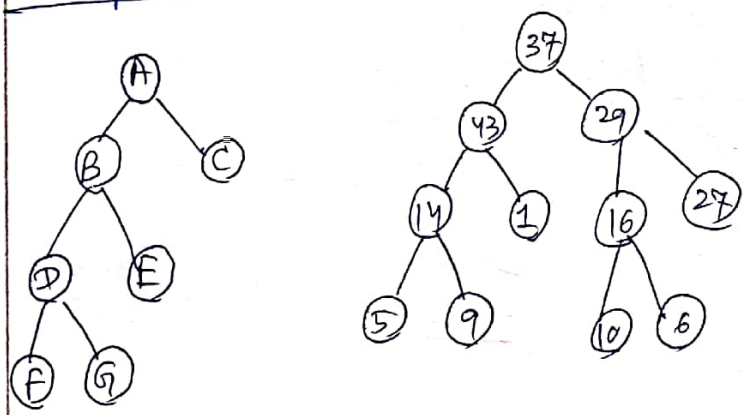
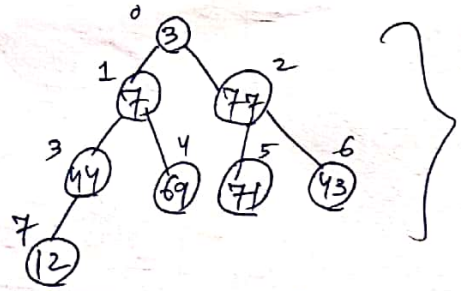


Fig - Binary Tree

3	7	77	44	69	71	43	12	



→ Almost Complete

→ Not Strictly Binary Tree

Fig: Almost Complete Binary Tree.

Memory Representation of Binary Tree

A tree can be represented by using

i) An array or

ii) Linked List.

#i Array Representation of Binary Tree

int t[MAX];

a) Root of the tree stored at 0th position, ie t[0]

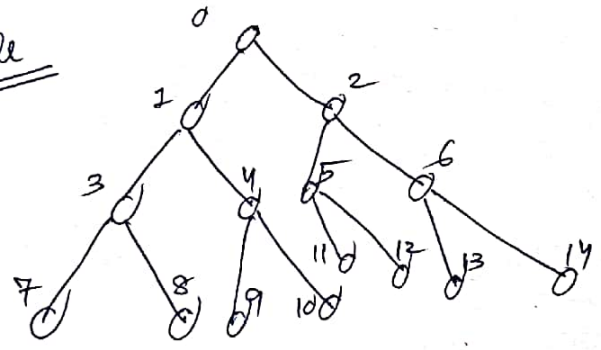
b) if a node is at ith position t[i]

⇒ Left child is stored in t[2i+1]

& Right child is stored at t[2i+2]

Parent is stored in t[(i-1)/2]

Example



Let i = 5

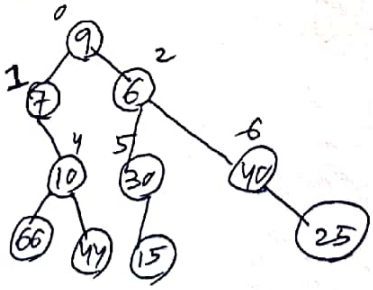
left_child(i) = t[2*5+1]
= t[11]

right_child[i] = t[2*5+2]
= t[12]

parent[i] = t[(5-1)/2]
= t[2].

Example:

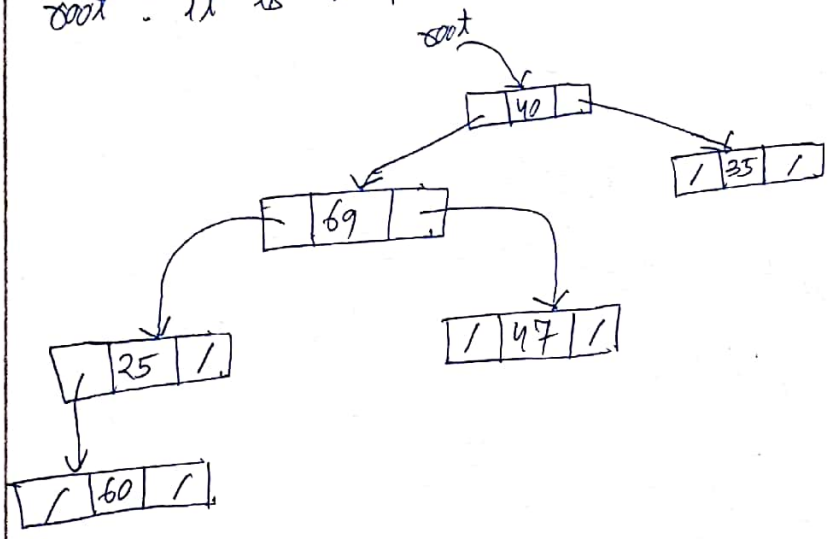
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
9	7	6		10	30	40			66	44	15			25



#ii) Linked List Representation of Binary Tree

* A node has 3-fields

Value : store the data item
left : pointer, which stores the address of left child
right : pointer, stores the " " right child
root : it is a pointer, which stores the address of root node.



BINARY TREE TRAVERSAL

Defi^n: - The process of going through a tree in such a way that each node is visited once and only once is called traversing the tree.

3-Basic operations during Traversing
- i) Visiting the root
- ii) Traversing left subtree
- iii) Traversing right subtree

3-Types of Traverse

1) Pre-order (Depth-First)
2) In-order (Symmetric order)
3) Post-order (

DS 27

PREORDER :

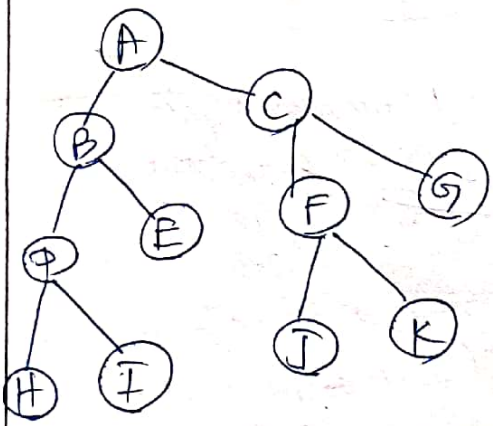
1) Visit the root
2) Traverse the left subtree in preorder
3) Traverse the right subtree in preorder

INORDER :

1) Traverse the left subtree inorder
2) Visit the root
3) Traverse the right subtree inorder.

POSTORDER :

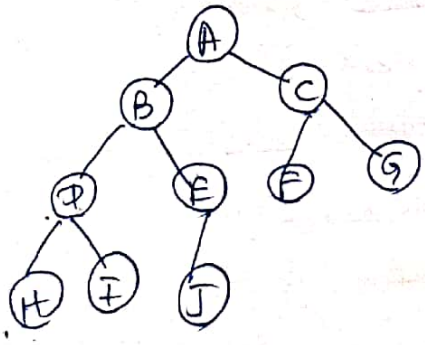
1) Traverse the left subtree postorder
2) Traverse the rightsubtree postorder
3) Visit the root.



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

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

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



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

Inorder: H D I B E A F C G

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

Recursive Algorithm for Tree Traversal

Algorithm (Preorder Tree Traversal)

Assumption:-

1) A binary tree is already created and it is in memory
2) node holds the address of the root node of the tree.

```
Preorder (node)
1) if (node ≠ NULL)
2)     Process (node).
3)     Preorder (left [node])
4)     Preorder (right [node])
5) End of if
6) Exit.
```

Algorithm (Inorder Tree Traversal)

Assumption:-

- A Binary tree is already created and present in the memory
- node holds the address of the root node of the tree.

```
Inorder (node)
1. if (node ≠ NULL)
2.     Inorder (left [node])
3.     Process (node)
4.     Inorder (right [node])
5. End of if
6. Exit
```

Algorithm (Postorder Traversal)

Assumption:
- A binary tree is already created and it is in memory
- node holds the address of the root node of the tree

```
Postorder (node)
1. if (node ≠ NULL)
2.     Postorder (left [node])
3.     Postorder (right [node])
4.     Process (node)
5. End of if
6. Exit.
```

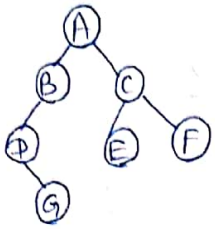
Non-Recursive Algorithm for Tree Traversal

Preorder Traversal (Non-Recursive)

Assumption:
1. A Binary tree is already created and present in memory
2. root holds the address of the root node of the tree.

```
Preorder NR (root)
1.  top = 0, stack [0] = NULL, node = root
2.  while (node ≠ NULL) {
3.      process value (node → value)
4.      if (node → right != NULL)
5.          top = top+1; stack [top] = node → right ; // Push right child into stack
6.      if (node → left != NULL)
7.          node = node → left
8.      else
9.          node = stack [top]; top = top-1; // Pop node from stack
10. } // end of while
```

Example:



Steps	Node Processed	Stack
1) node = A		0
2) Process A, Push right child into Stack, top=1	A	0 C
3) node = B	B	
4) node = D, process, push right to Stack	D	0 C G
5) node = G, top=1	G	0 C
6) Pop element from stack, node = C, top=0, top=1	C	0 F
7) node = E	E	
8) Pop from Stack, node = F	F	0
9) set node = NULL		Stack is Empty
10) As node = NULL the algorithm is completed		

DS 28

CONSTRUCTION OF BINARY TREE:

Input: Preorder nodes: A B D E F C G H J L K

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

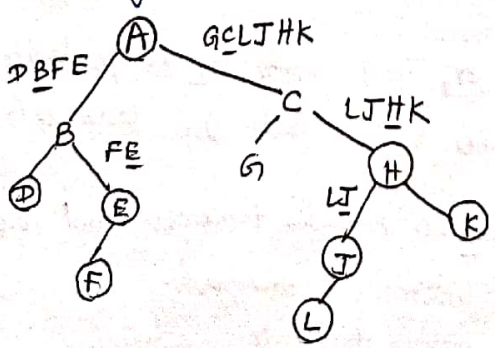
Construction Steps

1) The first node in the preorder traversal is the root of the tree

2) Find that root position in the inorder list of nodes.
The nodes precede to root node in the inorder traversal are the nodes of the left subtree of the root node, & the nodes succeed to the root node are the nodes in the right subtree of the root node.

3. Now consider 2-sets of inorder and preorder traversals of the left & right subtrees of the root. i) The first set is the nodes appear precede to the root node in inorder traversal & the combination of those nodes only appear in preorder traversal just after root node.
ii) The second set is the nodes appear after the root node in inorder traversal and the nodes in the preorder traversal except the root node and the nodes considered in the first set.

4) Consider the two sets of preorder and inorder traversals for left and right subtrees, & repeat steps – 2 and 3 till entire tree is constructed.

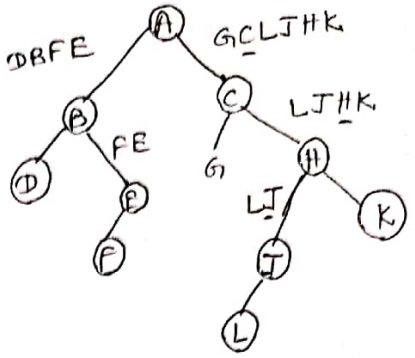


L10

Problem #2:

Postorder: DFEBGLJKHCA

Inorder: DBFEAGCLJHK



Problem #3:

Preorder: A B D G H K C E F

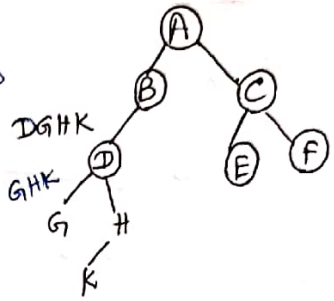
Postorder: G K H D B E F C A

set 1: {DGHK, GKHD}

set 2: {EF, EF}

n_1

n_2



Steps:

1) The first node in the preorder traversal and last node of the postorder traversal is considered as the root node of the tree.

2) Let n_1 = successor of the root node in the preorder traversal

n_2 = predecessor of the root node in the postorder traversal.

if $(n_1 == n_2) \Rightarrow$ ($n_1 = n_2$) shall be left/right child of the root node

Hence the tree is not unique.

else if $(n_1 \neq n_2) \Rightarrow$ n1 is left child & n_2 will be right child of root.

3) Find position of n_2 in preorder & n_1 in postorder traversal list

Now consider the two sets of preorder and postorder traversals of left and right subtrees of the root.

1st set: nodes after n_1 and before n_2 in preorder and nodes precede to node n_1 in postorder traversal.

2nd set: nodes after n_2 in preorder traversal and nodes in between n_1 and n_2 in postorder traversal.

4. Repeat step-2, & 3 till the entire tree is constructed.

Construct Binary Trees (Problem)

1) Preorder: GBQACKFPDERH

Inorder: QBKCFAGPEDHR

2) Postorder: DGEBHIFCA

Inorder: DBGEACHFI

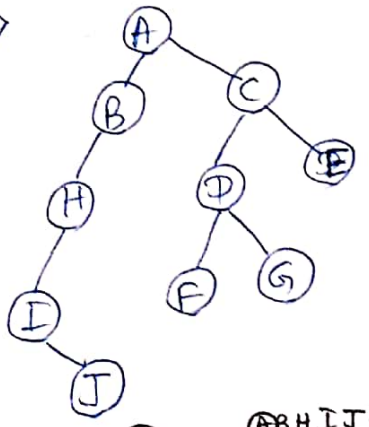
3) Inorder: a+b−c*d−e/f+g−h

Postorder: abc−+de−fg+h−/*

4) a + b * (c−d) + e / (f+g−h)

Problems: (Binary Tree Traversal)

1)



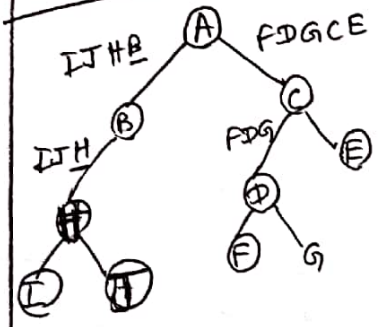
i) Preorder: A, B, H, I, J, C, D, F, G, E

ii) Inorder: IJHBAFDGCE

iii) Postorder: JIHBFGDECA

From (i) & (ii)

ABHIJCDFGE
IJHBAFDGCE



From (ii) & (iii)