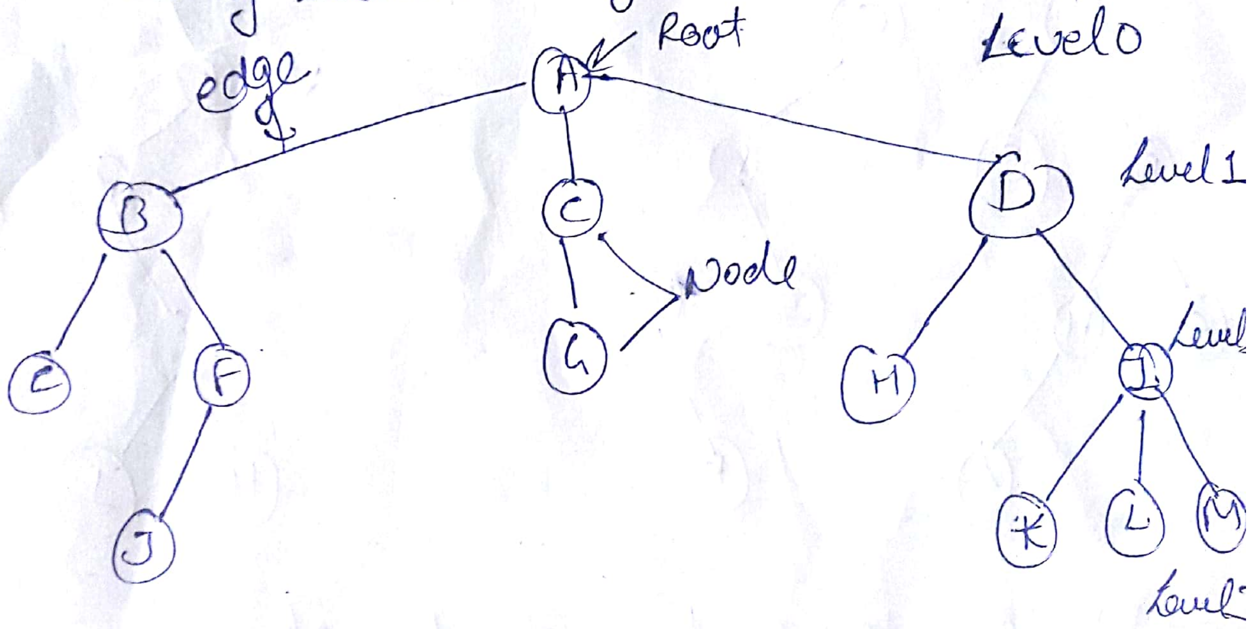
Module-4

TREE.

A tree is a non-linear data structure in which items are arranged in a sorted sequence. It is used to represent hierarchical relationship existing amongst several data items.

- There is a special data item called 'root' of tree.
- And remaining data items are partitioned into number of subsets, each of which is itself a tree.



Terminology:

Root - First item of tree.

Node - Each data item.

Degree of Node - A - 3, B - 2, H - 0

Degree of Tree - Max. no. of nodes in a tree

Terminal/Leaf Node - Node having no child.

Level - levels of child of tree.

Edge - Connecting line of two nodes.

Depth - No. of Level+1

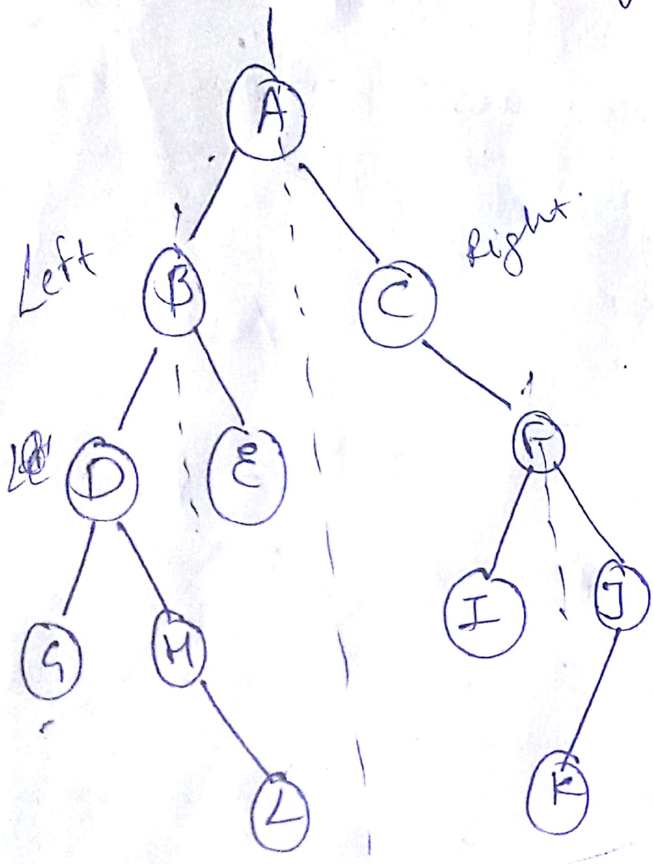
Forest - Set of disjoint trees.

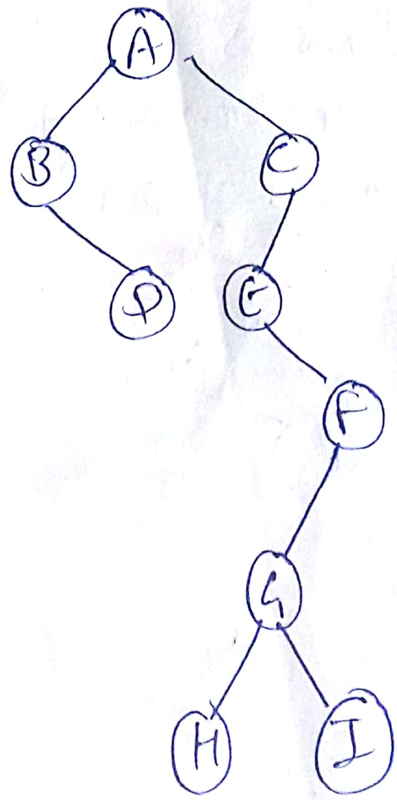
Traversing: shifting from one node to other.

→ Preorder – Root Left Right

→ Inorder – Left Root Right

→ Postorder – Left Right Root





→ Write orders of traversing for above trees.

Pre: A B D G H L C F I J K.

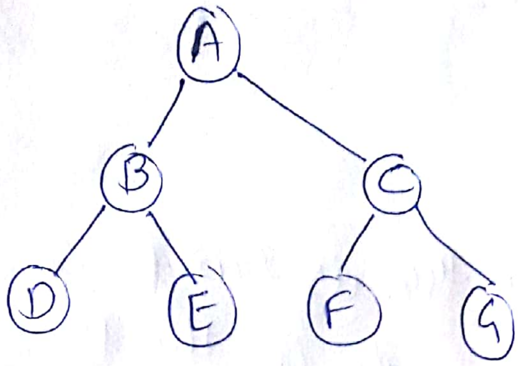
Post: G L H D E B I K J F C A.

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

Create tree from following traversing result

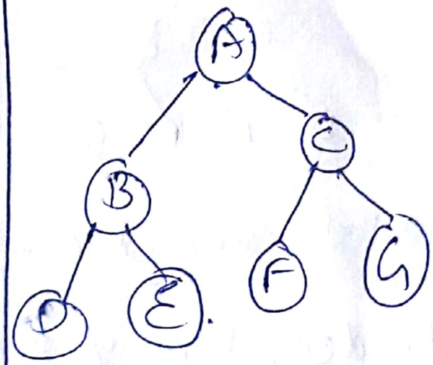
Pre- A B D E C F G

In- D B E A F C G.



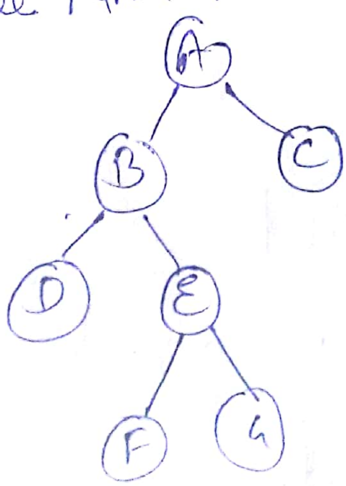
Post:

D E B F G C A



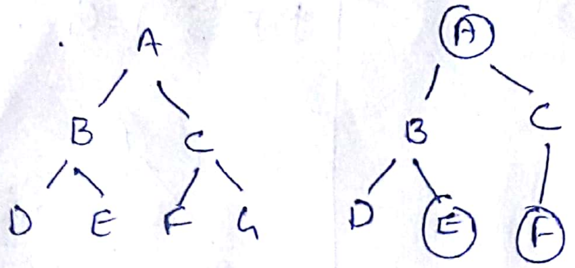
Binary Tree and Types

Strictly Binary Tree: If every non-terminal node in a binary tree consist of non-empty left subtree and right subtree, then such a tree is called strictly binary tree.



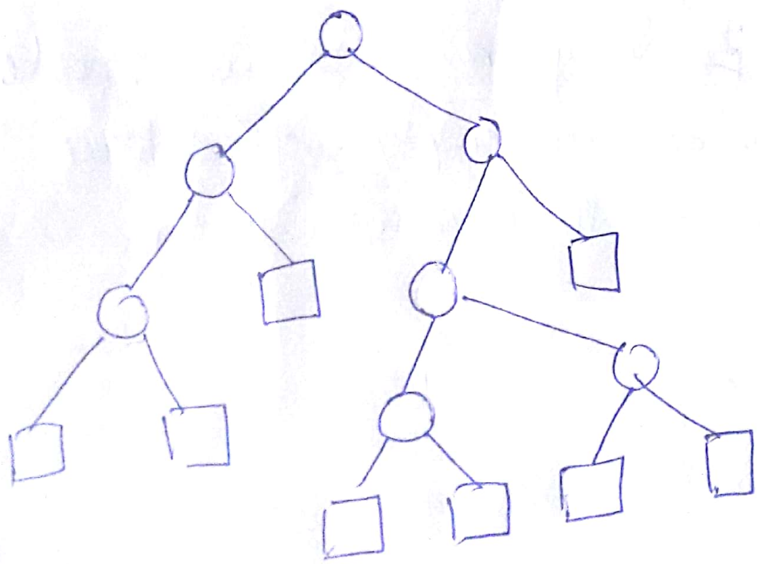
Complete Binary Tree - If all levels are completely filled except possibly the last level and last level has all keys as left as possible

Ex



Extended Binary Tree - A binary tree is said to be an extended binary tree if each node in the tree has either no child or exactly two children.

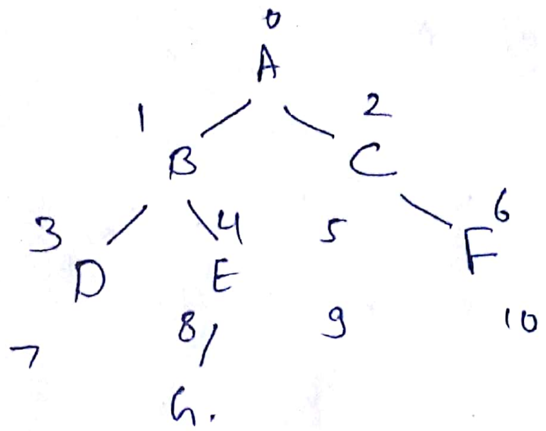
- Two children nodes - internal nodes ○
- no child → external nodes. □



Representation of trees in Memory:

1. Array Representation
2. Linked Representation.

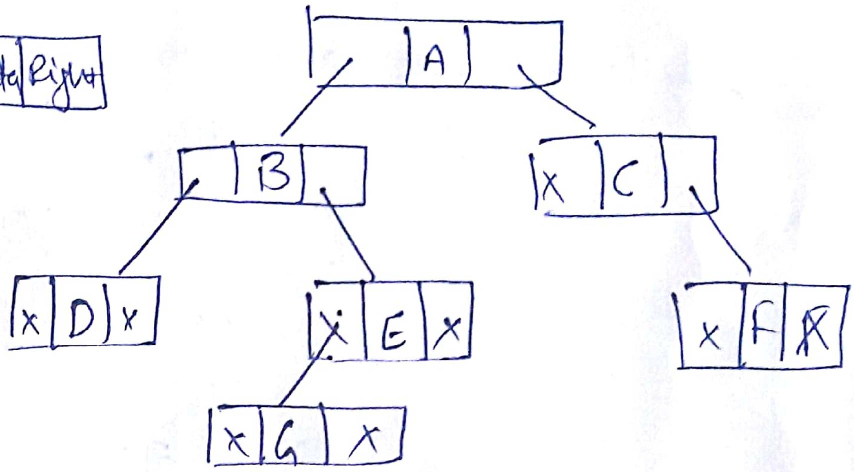
1. Array Representation:



0	A
1	B
2	C
3	D
4	E
5	-
6	F
7	-
8	G
9	-
10	-

2. Linked Representation.


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

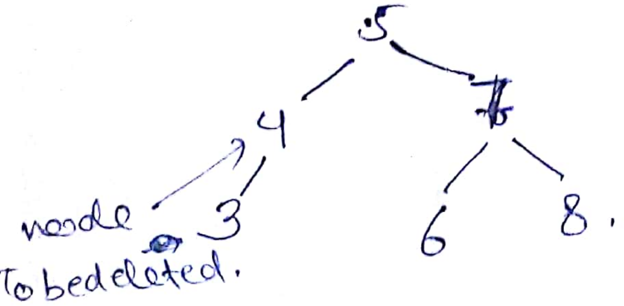


Binary Search Tree! It is a node based tree having following properties:

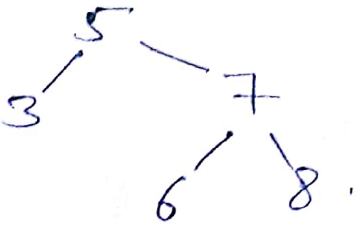
- → left subtrees have values less than root node
- right subtrees contains values greater than root node.
- If there are duplicate nodes, either avoid them or shift to right side of tree.

Ex- 8, 3, 6, 4, 7, 1, 10, 14, 13.

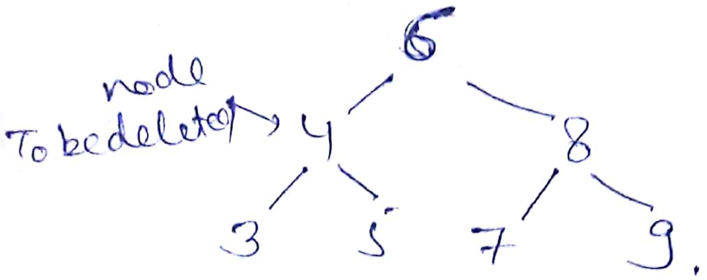
II. Having one child:



swap values of 3 with 4 & remove node (4)

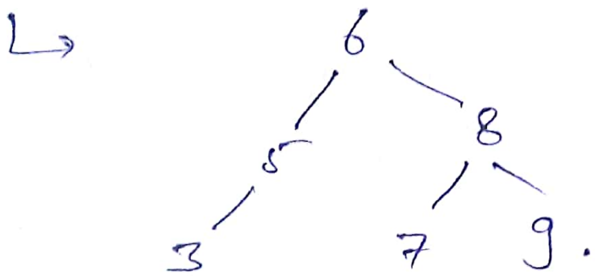


III. Having Two children.



I. find inorder traversing
3 4 5 6 7 8 9.

II. next node after 4 is 5
so, 5 will be root node.



AVL Tree:

AVL tree is a self balanced tree. That means, an AVL tree is also a BST (binary search tree) but it is a balanced tree. It was introduced in the year of 1962 by G.M. Adelson-Velsky and E.M. Landis.

Every node having balance factor - +1, -1, or 0

Balance factor = height of left subtree - height of right subtree.

Every AVL is binary tree but not all binary tree are AVL.

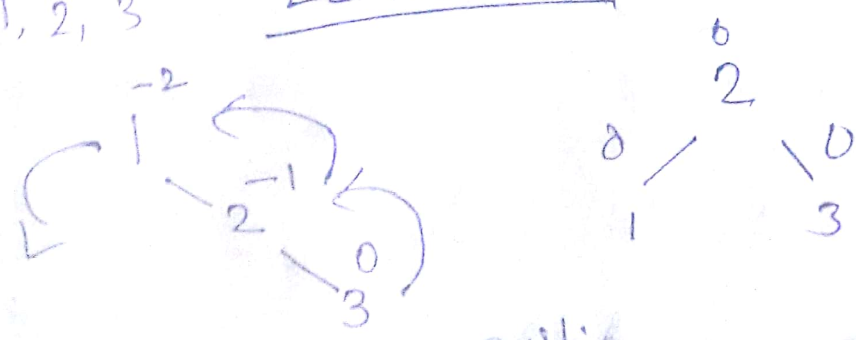
AVL Tree Rotations:

In AVL tree performing every operation we need to check Balance factor. If it is not balanced. then rotations are applied.

- Rotation
  - Single
    - LL - Left Rotation
    - RR - Right Rotation
  - Double
    - LR - Left Right Rotation
    - RL - Right Left Rotation

Ex: 1, 2, 3

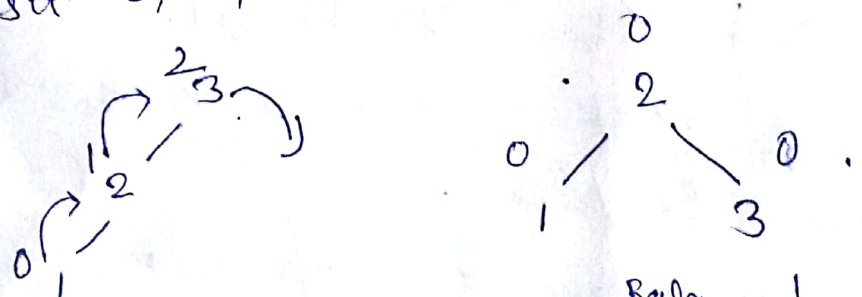
LL Rotation



move nodes to one position left

Single Right (RR) Rotation:

Insert 3, 2, 1.

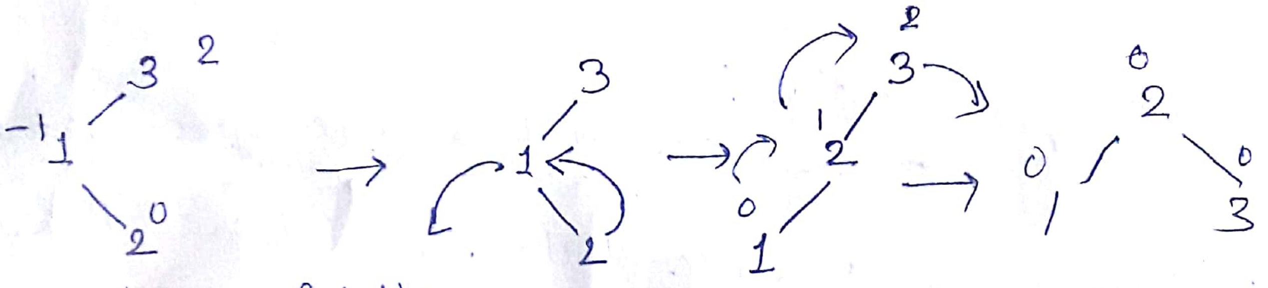


Move nodes to one position right

Balanced.

Left Right (LR) Rotation:

3, 1, 2



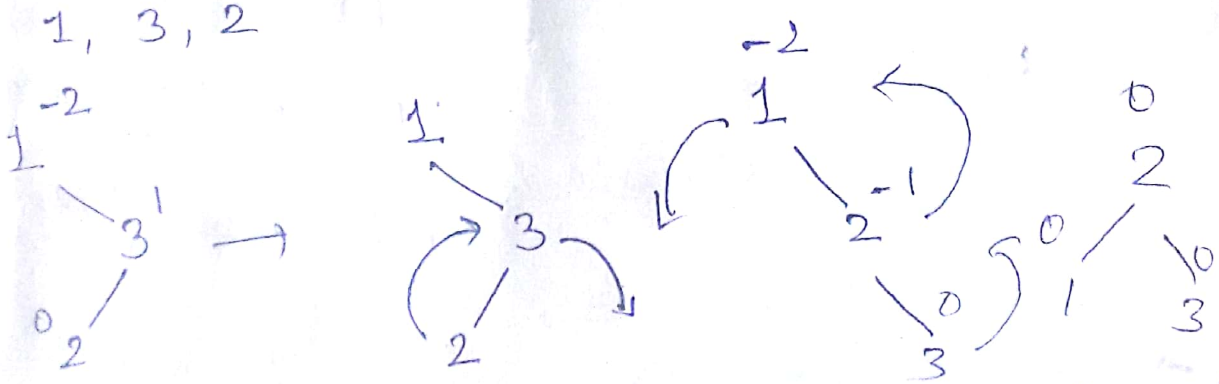
Applying LL Rotation

RR Rotation

Balanced.

Right Left Rotation: (RL Rotation)

1, 3, 2

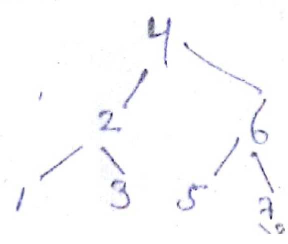


applying right rotation

apply left rotation.

Balanced

→ Construct an AVL tree from numbers 1 to 8



Ex- 10, 20, 15, 25, 30, 16, 18, 19.