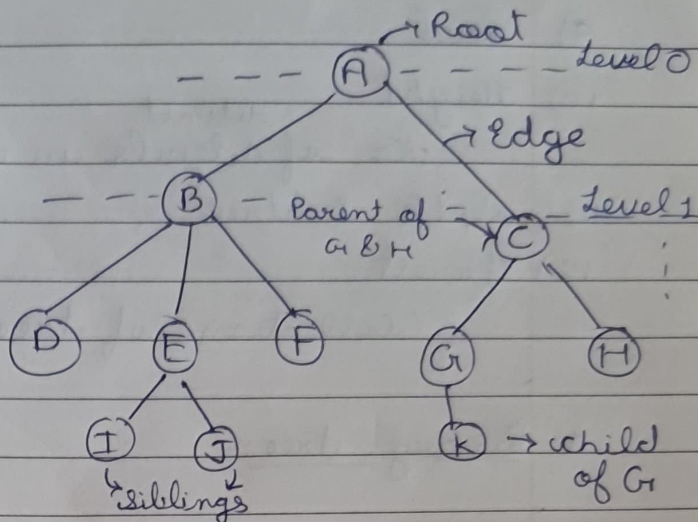
Ch: 4 Tree

Tree terminologies:-

- Root
- ~~child~~
- Sibilings
- Degree
- Internal node
- L



1) Root
2) Edge
   (In a tree has n nodes then it chan (n-1) edges)
3) Parent
   (In a tree, a parant node can have any no. of child nodes)

4) child
   (All nodes except root node are child nodes).
5) Siblings
6) Degree
   (Degree of node is total no. of {children of that node}.
   Degree of tree is

7) Internal node (eg. A, B, C, E, G)
   (same as non-terminal node, same as non-leaf node)
8) Leaf node (eg. I, J, K, H)
   (external or terminal node)

9) Level

10) Height

No. of distinct levels in the tree

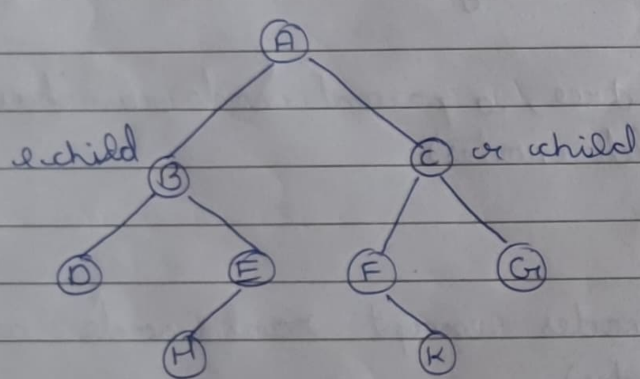
11) ~~Depth~~ Subtree

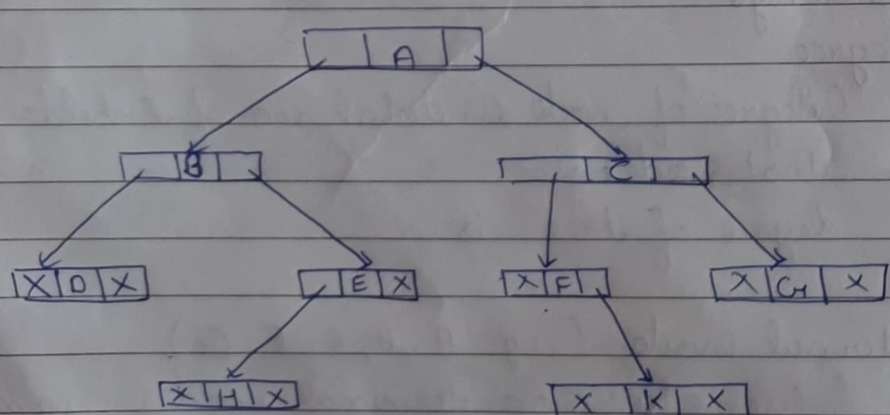
12) Forest

(Collection of trees)

* <u>Binary tree</u>:-

Each node has at most 2 childeren





Operations:-

- Inserting an element.
- Removing an element
- searching for an element.
- Traversing the tree.

} Most operations

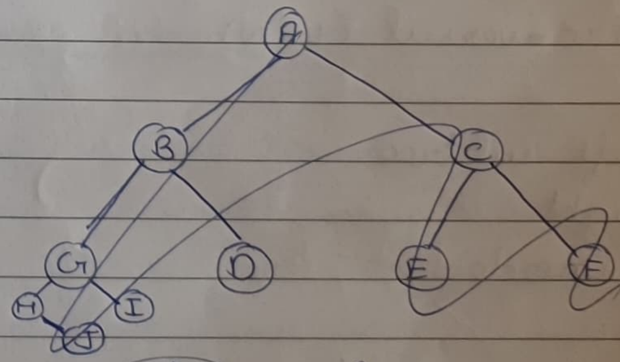
Traversal technique:-

1. Inorder (LNR)
2. Preorder (NLR)
3. Postnode (LRN)

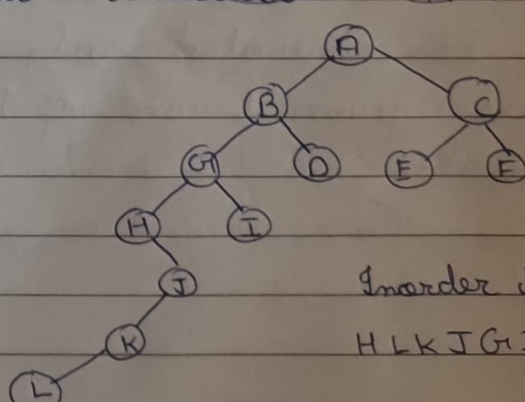
N - node
L - Left child
R - Right child

* Inorder traversal algorith:-

1. Traverse the left subtree recursively
2. Visit the root
3. Traverse the right subtree recursively



In order traversal: H J



Inorder traversal:

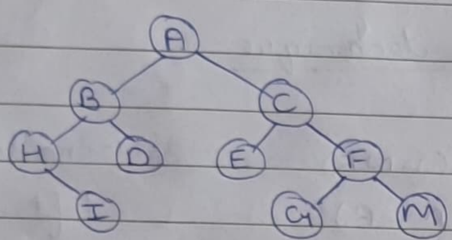
H L K J G I B D A E C F

PAGE NO.:

* Preorder traversal (NLR) :-

Algorith :-

1. Visit root node
2. Traverse left subtree
3. " right "



Inorder: H I B D A E C G F M

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

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

* Postorder traversal (LRN):-

1. Traverse left subtree
2. " right "
3. Visit root node

* The tree can be generated only if at least 2 traversals are given & one of them is inorder traversal.

* ~~Types of binary tree:-~~

* **Tree :-**

A tree in a non-linear data structure and a hierarchy consisting of a collection of nodes such that each node of the tree stores a value and a list of references to other nodes (the "children").

Binary tree:-

It is defined as a tree data structure where each node has at most 2 children. Since each element in binary tree can have only 2 children, we typically name them the left & right child.

* **Binary search tree :-**

Binary search tree is a node-based binary tree which has foll. properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left & right subtree each ~~much~~ must also be a binary search tree.

* Threaded Binary Tree:-

Program for binary search tree & its traversal

```
#include <stdio.h>
#   "     <stdlib.h>

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

struct node* search(struct node*, int);
struct node* insert( "      " *, int);
void inorder(struct node*, ~~int~~);
 "   preorder(  "      " *, ~~int~~);
 "   postorder(  "     " *, ~~int~~);
int height (struct node*);
void main()
{
    struct node *root, *ptr;
    root = NULL;
    int ch, k;
    while(1)
    {
      printf("\n1.Search 2.Insert 3.Preorder 4.Inorder
              5.Postorder 6.Height 7.Exit \nEnter
              your choice:");
      scanf("%d", &ch);
```

```
switch(ch)
{
  case 1: printf("Enter data to be searched:");
       scanf("%d", &k);
       ptr = search(root, k);
       if (ptr==NULL)
          printf("%d not present\n", k);
       else
          printf("%d present\n", k);
       break;
  case 2: printf("Enter element to insert:");
       scanf("%d", &k);
       root = insert(root, k);
       break;
  case 3: preorder(root);
          break;
  case 4: inorder(root);
          break;
  case 5: postorder(root);
          break;
  case 6: printf("Height of tree =%d\n", height(root));
         break;
  case 7: exit(1);
  default: printf("Wrong choice\n");
 }
}
}
```

PAGE NO.:

```
struct node* search(struct node * ptr, int key)
{
    if(ptr==NULL)
    {
     //printf("%d not found \n", key);
       return NULL;
    }
    else if(key < ptr->data)
        return (ptr->left, key); return search(ptr->left,
                                                  key);
    else if (key > ptr->data)
        return (ptr->right, key); search(ptr->right,
                                              key));
    else
        return ptr;
}

struct node* insert(struct node*ptr, int key)
{
    if (ptr ==NULL)
    {
      ptr = (struct node*) malloc (sizeof(struct node));
      ptr->data = key;
      ptr->left = ptr->right = NULL;
    }
    else if(key < ptr->data)
        ptr->left = insert (ptr->left, key);
    else if (key > ptr->data)
        ptr->right = insert (ptr->right, key);
    else
        printf("Duplicate key \n");
    return ptr;
}
```

PAGE NO.:

```
void inorder (struct node * ptr)
{
    if (ptr = = NULL)
        return;
    inorder (ptr->left);
    printf ("%d   ", ptr->data);
    inorder (ptr-> right);
}
void preorder (struct node * ptr)
{
    if (ptr = = NULL)
        return;
    printf ("%d   ", ptr->data);
    preorder (ptr->left);
    preorder ( ptr-> right);
}
void postorder (struct node *ptr)
{
    if (ptr = = NULL)
        return;
    postorder (ptr-> left);
    postorder ( ptr-> right);
    printf ("%d   ", ptr->data);
}
int height (struct node* ptr)
{
    int l, r ;
    if (ptr = = NULL)
        return 0;
    l = height (ptr-> left);
    r = height (ptr-> right);
```

```
if(l>r)
    return (l+1);
else
    return (r+1);
}
```

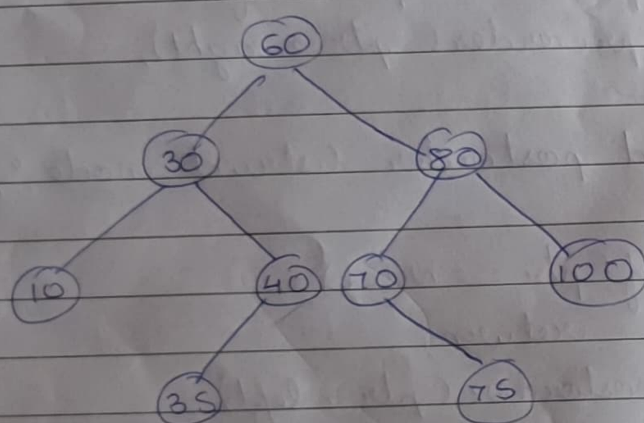
* Binary search tree :-

Deletion of node:-

i) Node with 0 children (leaf node) ⇒ 10, 35, 75, 100

ii) " " 1 child (L/R) ⇒ 40, 70

iii) " " 2 children (L & R) ⇒ 60, 30, 80



C Program:-

```
struct node* deletion (struct node *ptr, int key)
{
    struct node *temp, *succ;
    if(ptr == NULL)
    {
        printf("%d not found \n", key);
        return ptr;
    }
```

PAGE NO.:

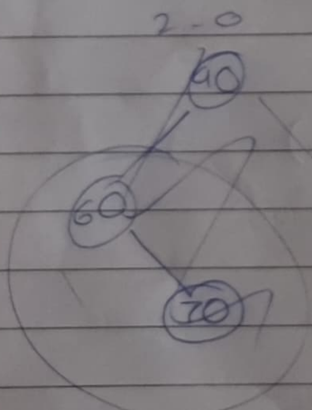
```
if( key < ptr->data)
    ptr->left = deletion(ptr->left, key);
else if( key > ptr->data)
    ptr->data = deletion (ptr->right, key);
else
{
  if (ptr->left != NULL && ptr->right != NULL)
  {
(iii)  succ = ptr->right;
       while(succ->left)
            succ = succ->left;
       ptr->data = succ->data;
       ptr->right = deletion(ptr->right, succ->data);
  }
  else
  {
       temp = ptr;
(ii) //  if(ptr->left != NULL)
            ptr = ptr->left;
       else if( ptr->right != NULL)
            ptr = ptr->right;
(i) // else
            ptr = NULL;
       free(temp);
  }
}
return ptr;
}
```

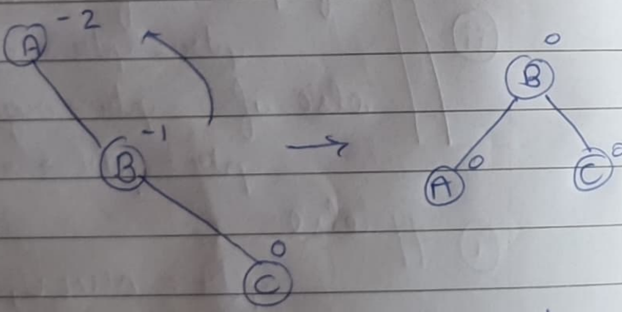
Ch:5 Height Balanced Trees.

Syllabus:
1) Definition
2) Height of the tree
3) Insertion & Deltion of node in AVL tree
4) Single & Double rotation of AVL tree

AVL Tree operations
i) Insertion
ii) Deletion

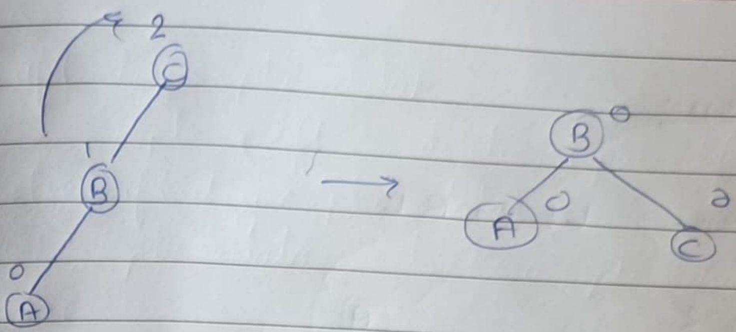
AVL Tree Rotation
i) Single → Left, → Right
ii) Double → Left-right, → Right-left





C is added in right subtree of right subtree of A

Then rotate at left
RR Rotation



A is added to right subtree of left subtree of C

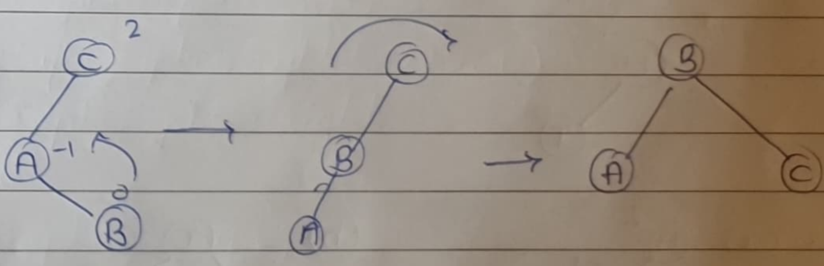
Then rotate at right

LL Rotation.

LR Rotation :-

Double rotations are bit tougher than single rotation which has also LR rotation = RR + LL rotations i.e., first RR is performed on subtree & then LL rotation is performed on full tree, by full tree we mean the 1st node from path of inserted node.

Eg.

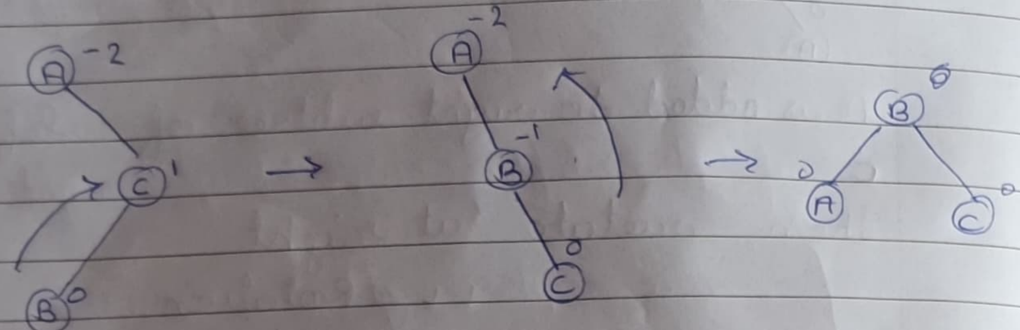


B is added at right subtree of left subtree of C

RL rotation :-

RL rotation = LL + RL

Eg.

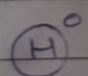


B is added at left subtree
of right subtree of A

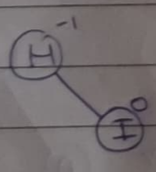
Eg. Construct an AVL tree having foll.
elements in it.

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

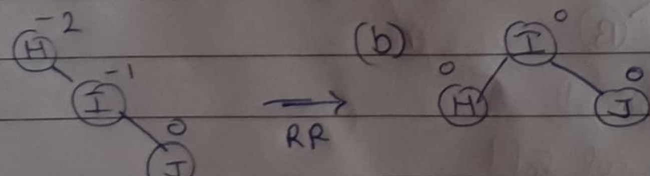
1.



2.

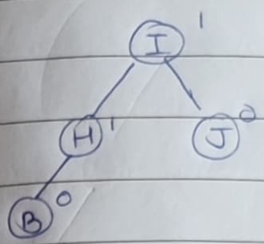


3. (a)

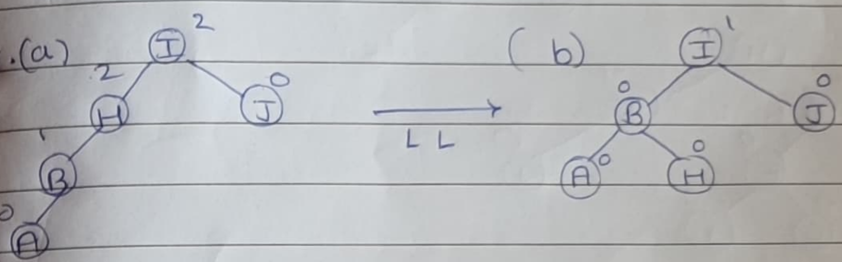


Write a little
description

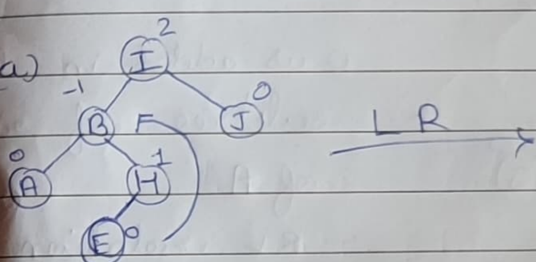
4.



5.(a)



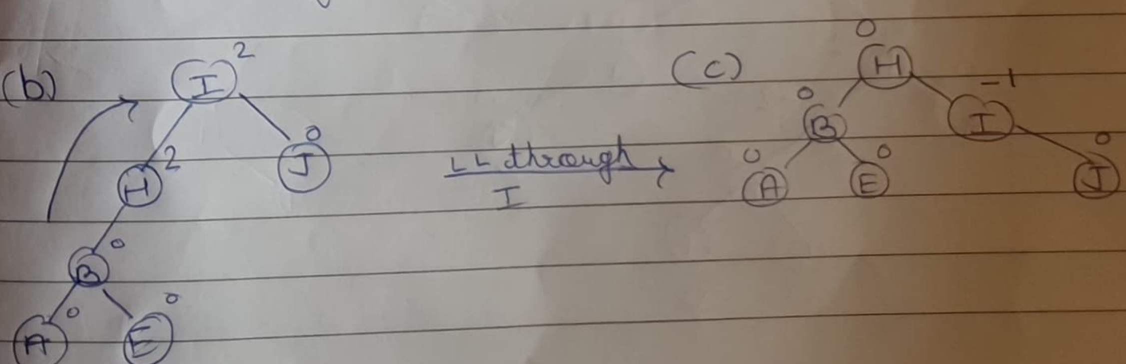
6(a)



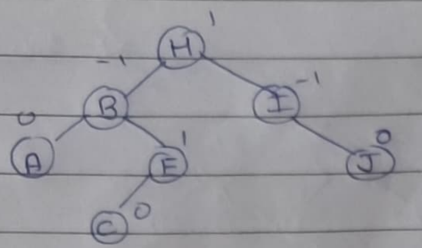
E is added
in right subtree
of left " of I

We 1st perform RR through B

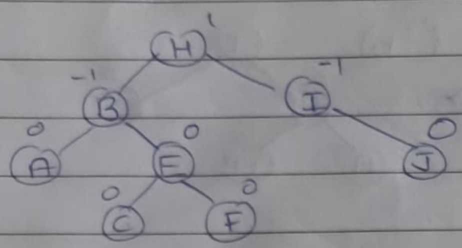
(b)



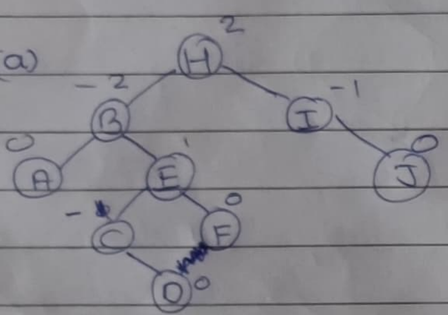
7.



8.



9.(a)



D is added in left subtree of right subtree of A

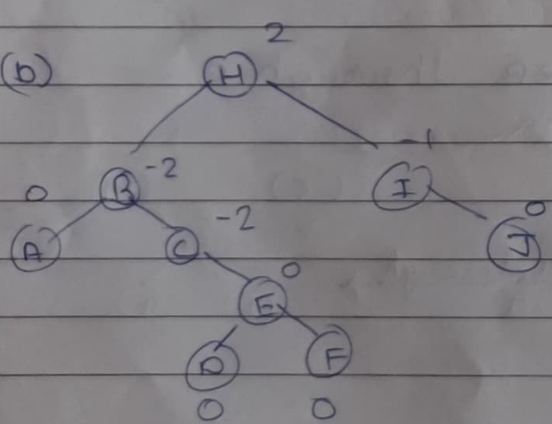
∴ RL rotation

||

LL + RR

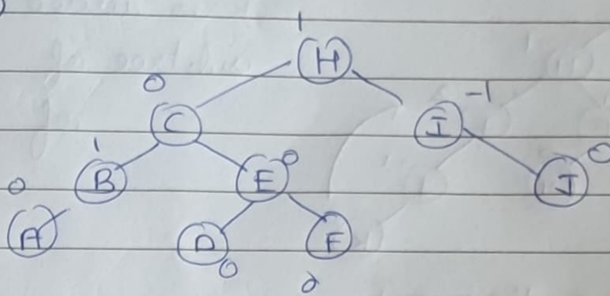
LL through E

(b)



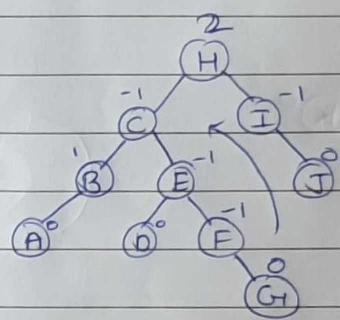
- RR through B

(c)



10) Insert G

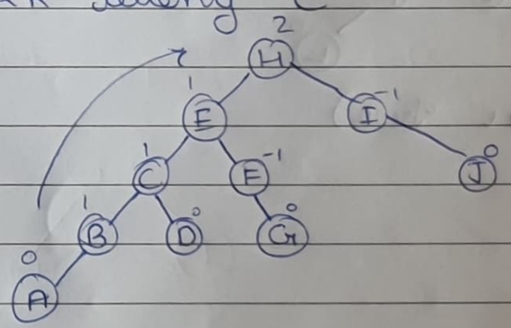
(a)



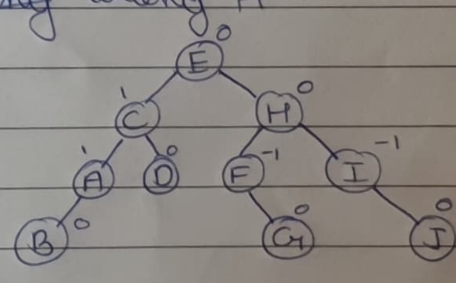
G → right subtree of left subtree of H

LR Rotation = RR + LL

(b) RR along C



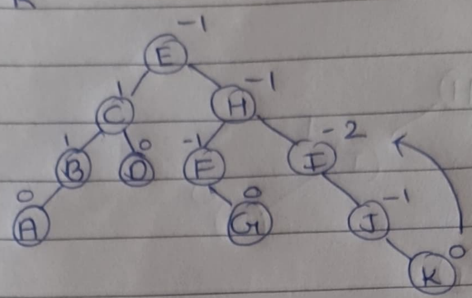
(c) LL ~~along~~ along H



PAGE NO.:

11) Insert K

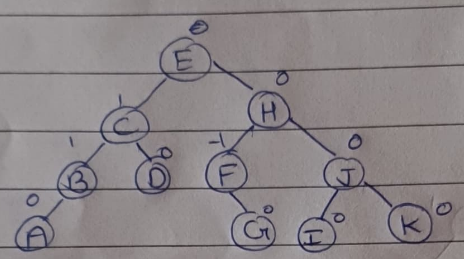
(a)



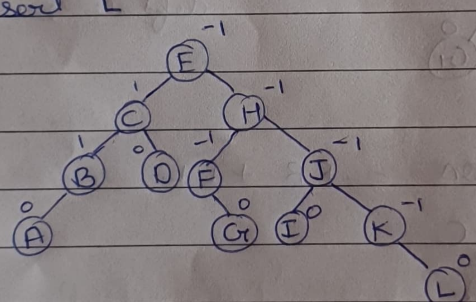
K is added in right subtree of right ,, of I

RR Rotation throug I

(b)



12) Insert L



(Final AVL Tree)