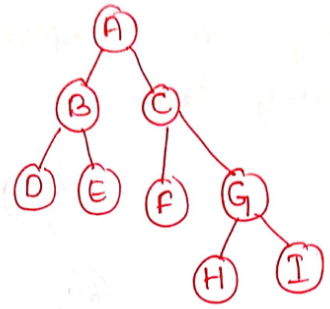
USN | | | | | | | | | | |

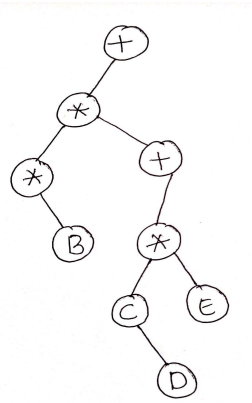
Internal Assessment Test 3 – Feb 2023

Sub:	Data Structures and Applications					Sub Code:	21CS32	Branch:	CSE
Date:	6/02/2023	Duration:	90 mins	Max Marks:	50	Sem / Sec:	III(A, B & C)		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1 (a)	Consider the tree given below:  1. Name the Non-Leaf Nodes **……[1 Mark]** **B C G** 2. Find the height of the tree. ……**[1 Mark]** **4** 3. Find the subtree height rooted at node E. …..**[1 Mark]** **1** 4. Find the Level of Node G. ……**[1 Mark]** **3** 5. Name the siblings of C. ……**[1 Mark]** **B** 6. Name the Leaf-nodes ……**[1 Mark]** **D E F H I**	[06]	CO4	L2
1 (b)	Show that 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 $n_0=n_2+1$ Proving the Lemma …… **[3 Marks]** Writing the conclusion ……**[1 Mark]** First, suppose that the two fork tree has n nodes, then how many edges will it have? The answer is N-1, because in addition to the root node, each of the remaining nodes has only one parent node, then the N nodes contribute to the N-1 edge of the tree. This is the thinking from the bottom up, and from the top down (from the root to the leaf node) thinking, easy to get each node's degrees and 0*n0 + 1*n1 + 2*n2 is the number of edges. So we have the equation N-1 = n1 + 2*n2, replace N with N0 + n1 + n2, get N0 + n1 + n2-1 = n1 + 2*n2, so there are	[04]	CO4	L3

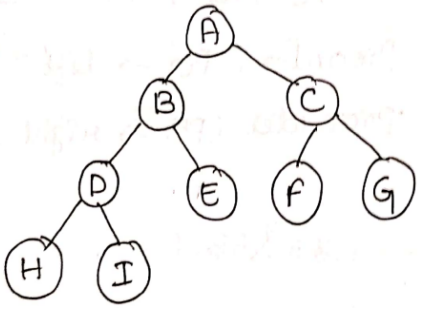
	N0 = n2 + 1. The proposition is to be proven.			

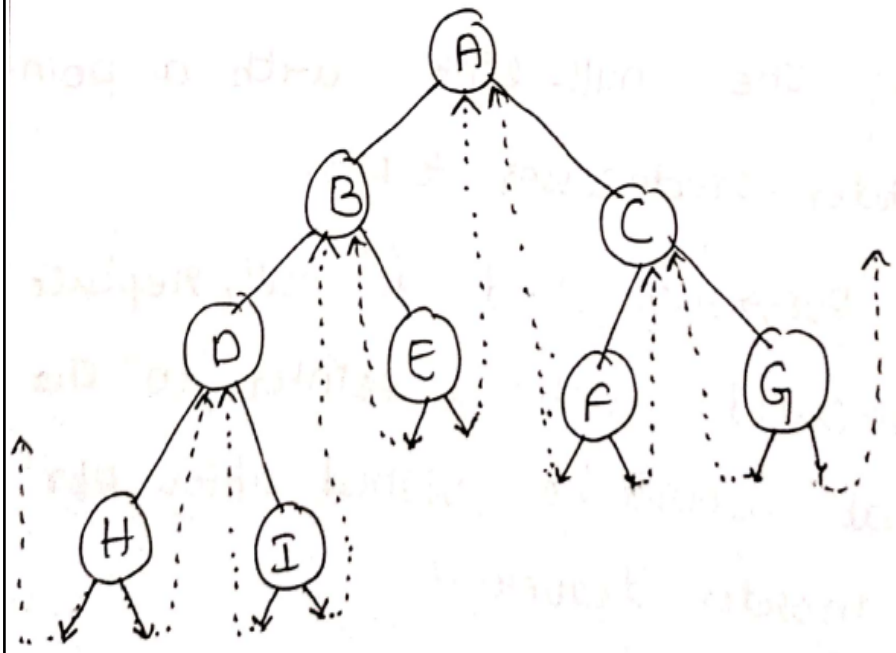
2	Consider the following tree:
	 
	
	Perform the following:
	1. Write a short note on the expression tree. …….. **[1 Mark]**
	
	The expression tree is a tree used to represent the various expressions. The tree data structure is used to represent the expressional statements. In this tree, the internal node always denotes the operators. The leaf nodes always denote the operands. The operations are always performed on these operands. The operator present in the depth of the tree is always at the highest priority.
	
	2. Perform the following Traversals on the given tree and write a Recursive C code:
	a. Pre-Order Traversal + C Function ……**[1+ 2 Mark]**
	+ * * B + * C D E
	void preorder(struct Tree *temp){
	if(temp!=NULL){
	printf("\t%d",temp->data);
	preorder(temp->lchild);
	preorder(temp->rchild);
	}
	
	}
	
	b. Post-Order Traversal + C Function ……**[1+ 2 Mark]**
	B * D C E * + * +
	void postorder(struct Tree *temp){
	if(temp!=NULL){
	postorder(temp->lchild);
	postorder(temp->rchild);
	printf("\t%d",temp->data);
	}
	
	}
	
	c. In-Order Traversal + C Function ……**[1+ 2 Mark]**
	* B * C D * E + +
	void inorder(struct Tree *temp){
		[10]	CO4	L3

```
            if(temp!=NULL){
               inorder(temp->lchild);
               printf("\t%d",temp->data);
               inorder(temp->rchild);

            }

         }
```

3(a)	For the given tree, Construct a Threaded Binary Tree. Also, point out how the left threads and right threads are linked in the Threaded Binary Tree.  Threaded Binary Tree representation …….. **[4 Mark]** - To construct the threads, the following rules are followed: → 1. If Ptr → leftchild is null, replace Ptr → leftchild with a pointer to the node that would be visited before Ptr in inorder - traversal i.e; The null- link with a pointer to inorder - predecessor to ptr. → If Ptr → right child is null, replace Ptr → rightchild with a pointer to the node that would be visited after ptr in an inorder traversal. Explanation about the left thread and right thread ……. **[2 Marks]**	[6]	CO4	L3

3(b)	Write a Recursive C Function to search an element in Binary Search Tree and display appropriate messages.
	
	C Code for searching an element ……..[4 Marks]
	void search(struct Tree *root, int val){
	struct Tree *temp;
	temp = root;
	if(temp == NULL)
	printf("\n Element not found");
	else if(val<temp->data)
	search(temp->lchild,val);
	else if(val>temp->data)
	search(temp->rchild,val);
	else
	printf("\n Element Found!!");
	}
		[4]	CO4	L2

4(a)	Define Binary Search Tree. Construct a Binary Search tree for the following elements: 14, 15, 4, 9, 7, 18, 3, 5, 16, 14, 20, 17, 9. Definition: …… **[2 Marks]**	[06]	CO2	L3

Binary Search Trees :

A binary search tree is a binary tree. It may be empty. If it is not empty, then it satisfies the following properties:

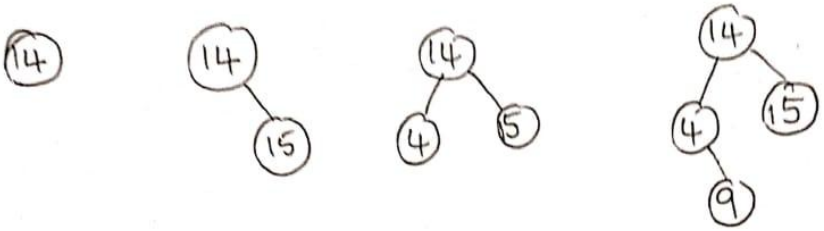
1. Each node has exactly one key, and the keys in the tree are distinct.
2. The keys (if any), in the left sub-tree are smaller than the key in the root.
3. The keys (if any), in the right sub-trees are larger than the key in the root.
4. The left and right subtrees are also Binary search trees.

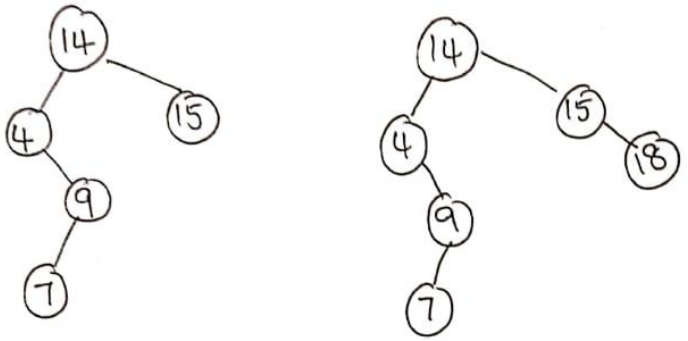
→ Properties 2, 3, 4 implies the keys must be distinct.

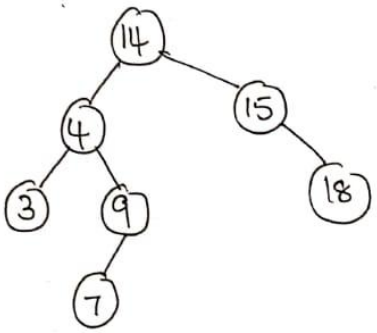
Construction of tree step by step …..**[4 Marks]**

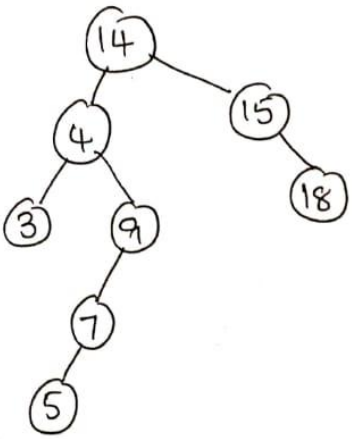
Given elements are:

14, 15, 4, 9, 7, 18, 3, 5, 16, 14, 20, 17, 9.

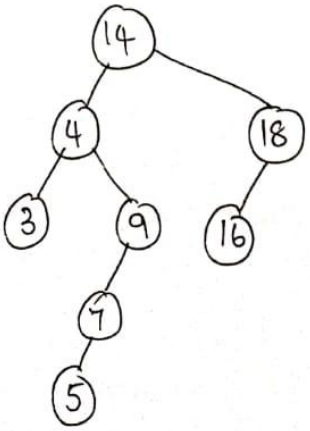


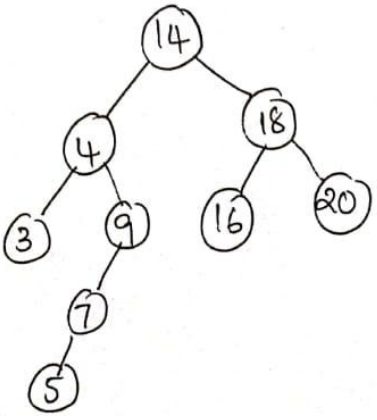


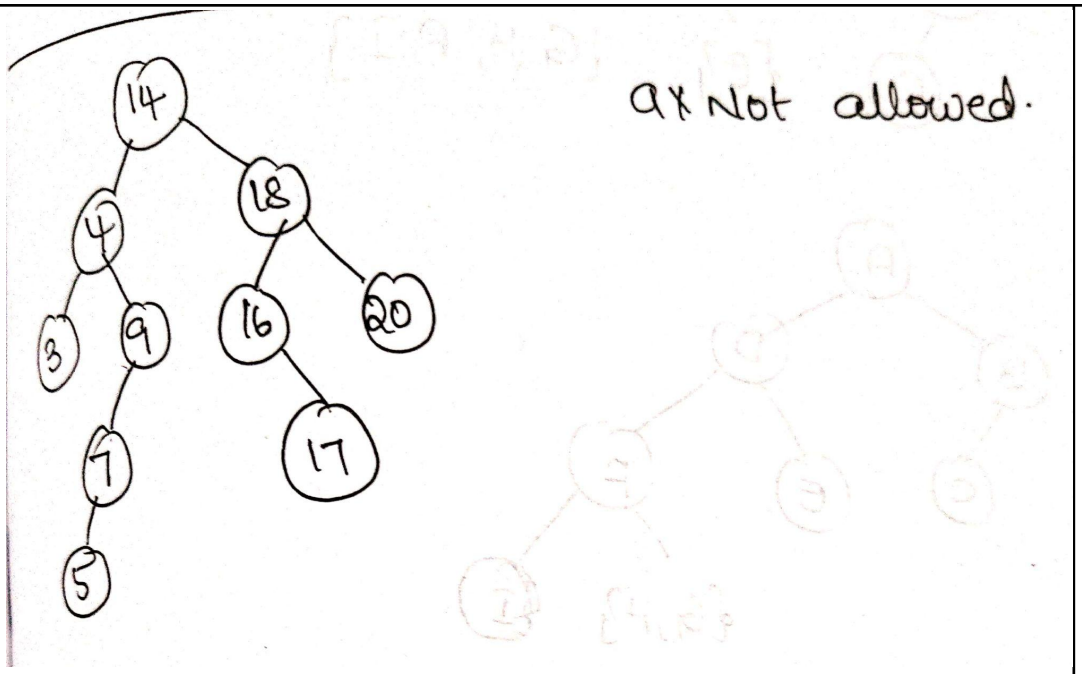




14 x Not allowed.





				
4(b)	Given the following traversal, Draw a Binary tree. 1. Inorder: BCAEDGHFI Preorder: ABCDEFGHI **Drawing the tree step by step : …….. [2.5 Marks]**	[05]	CO2	L3

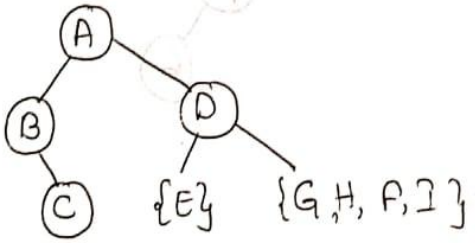
Given traversals

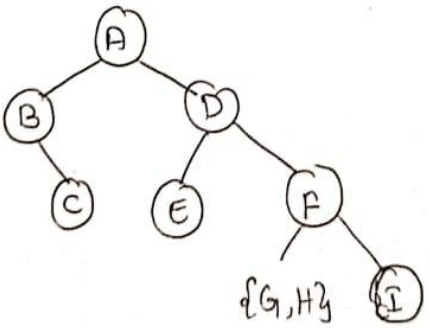
Inorder : B C A E D G H F I

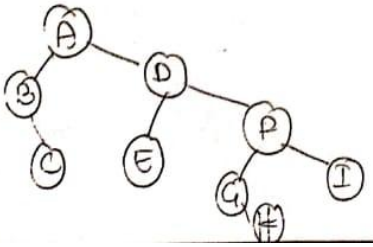
Preorder : A B C D E F G H I.

→ The root of the tree is considered from pre order and left child and right child are considered from Inorder.

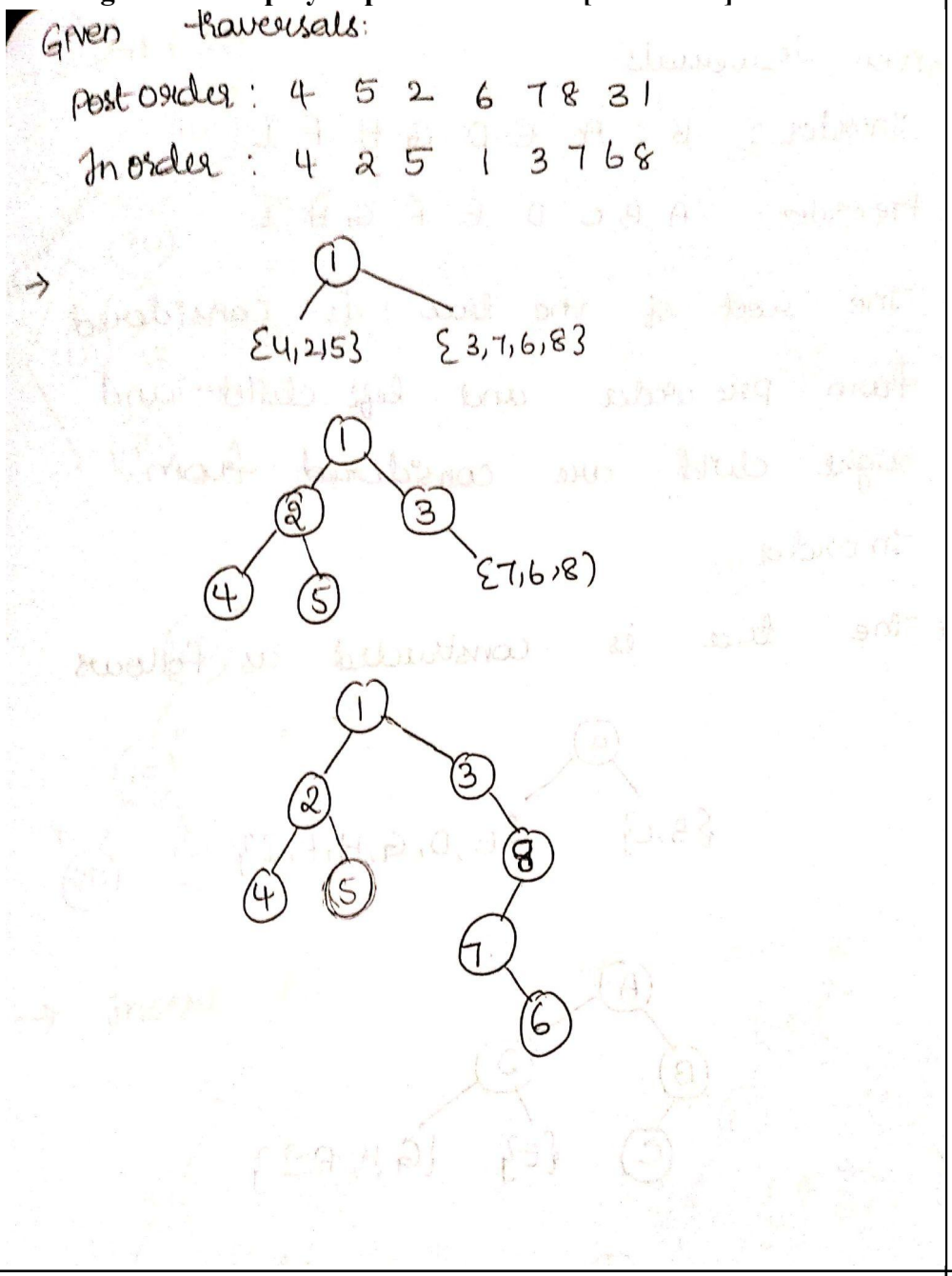
→ The tree is constructed as follows







2. Postorder: 4 5 2 6 7 8 3 1

	Inorder: 4 2 5 1 3 7 6 8 **Drawing the tree step by step : …….. [2.5 Marks]** 			
5(a)	Give the importance of the Balance Factor in the AVL tree. For the following elements 40, 50, 70, 30, 42, 15, 20, 25, 27. Construct an AVL tree and update the balance factor for every node insertion. Importance of Balance factor …… **[1 Mark]** Construction of tree step by step ……**[5 Mark]** **SOLUTION** Balance factor of a node in an AVL tree is the difference between the height of the left subtree and that of the right subtree of that node. Balance Factor = (Height of Left Subtree - Height of Right Subtree) or (Height of Right Subtree - Height of Left Subtree)	[06]	CO5	L3

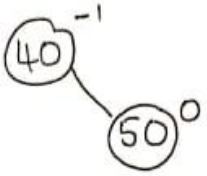
	The self-balancing property of an AVL tree is maintained by the balance factor. The value of the balance factor should always be -1, 0, or +1			

Given elements are 40, 50, 70, 30, 42, 15, 20, 25, 27.

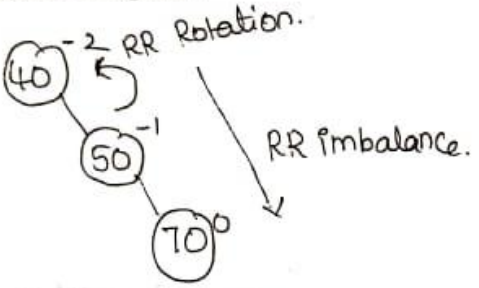
→ The construction of AVL trees is as follows:

→ Insert (40)

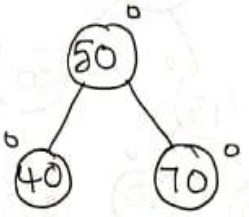
→ Insert (50)



→ Insert (70)

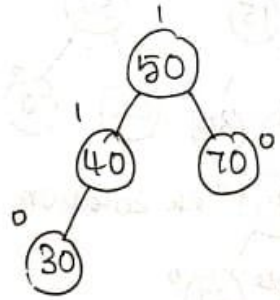


[Unbalanced Tree]

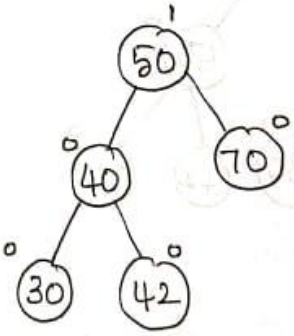


[Balanced Tree]

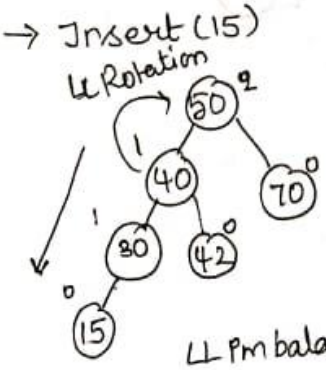
→ Insert (30)



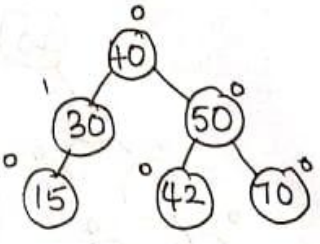
→ Insert (42)



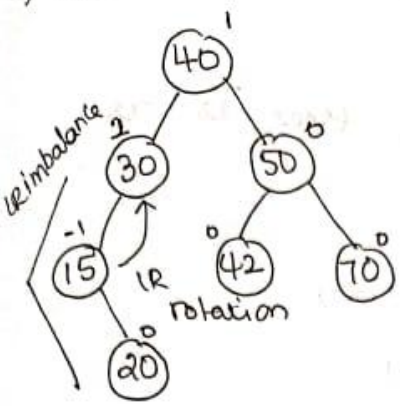
→ Insert (15)



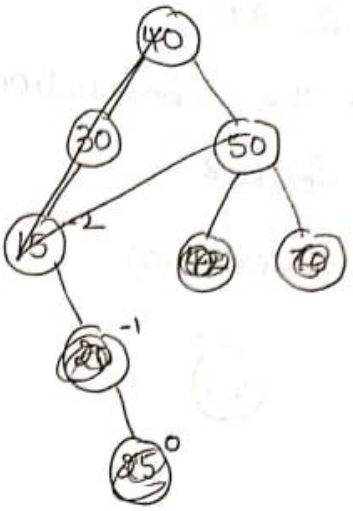
LL Imbalance

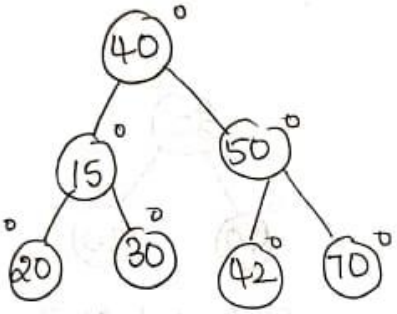


→ Insert (20)

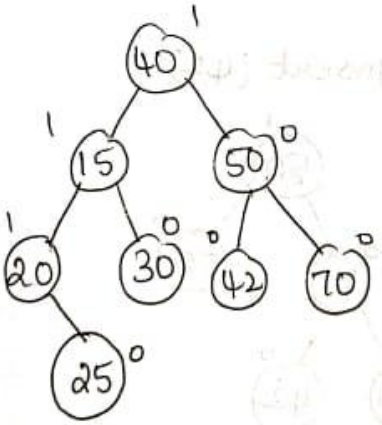


→ ~~Insert (25)~~

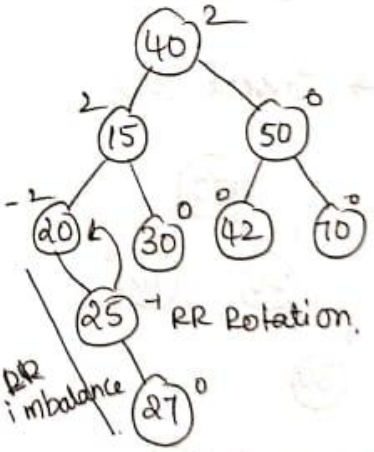


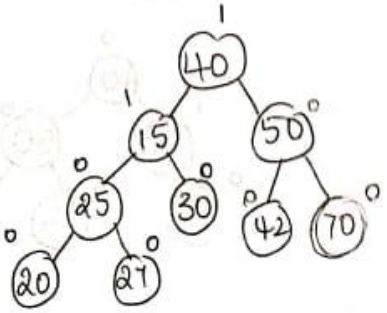


→ Insert 25



→ Insert 27





5 (b)	Consider the hash table of size 10. Using the Linear Probing technique insert the keys 72,27,36,24,63,81,92,101 into the hash table. Table representation, and hashing ……. [4 Marks]	[04]	CO5	L2

Sol:

Given keys are:

72, 27, 36, 24, 63, 81, 92 and 101

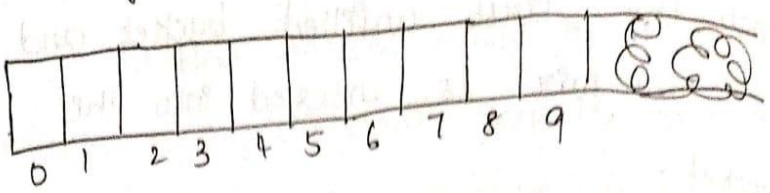
Size of hash table = 10.

hash function $h(k) = k \% D = k \% 10$

or

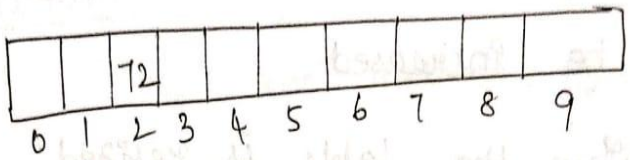
$h(k) = k \bmod 10$.

Initially, the hash table is as follows.



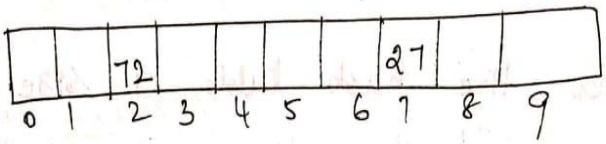
→ Insert Key 72

$h(72) = 72 \% 10 = 2 \rightarrow$ Index.



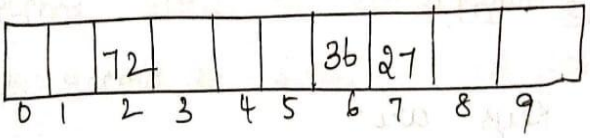
→ Insert Key 27

$h(27) = 27 \% 10 = 7$



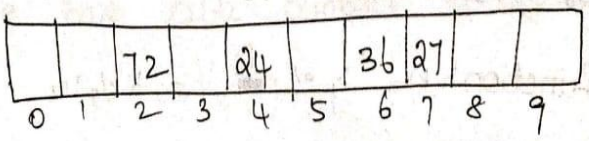
→ Insert Key 36

$h(36) = 36 \% 10 = 6$



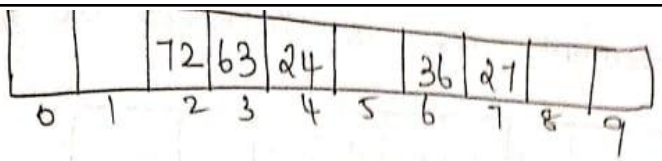
→ Insert Key 24.

$h(24) = 24 \% 10 = 4$



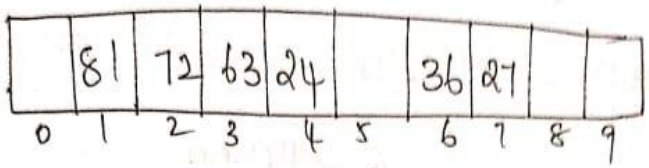
→ Insert Key 63

$h(63) = 63 \% 10 = 3.$



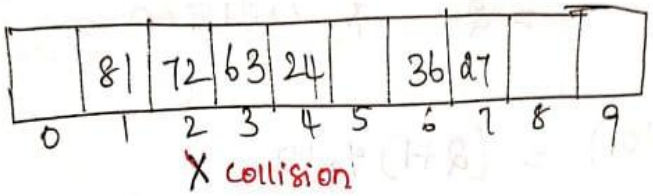
→ Insert key 81.

$h(81) = 81 \% 10 = 1$



→ Insert key 92.

$h(92) = 92 \% 10 = 2.$



→ In index 2, already an element exists. so collision happens.

→ The next position is searched using linear probing as follows:

$h'(K) = (h(K) + i) \% D$

$i = 1 \quad = (2+1) \% 10 = 3$ X collision

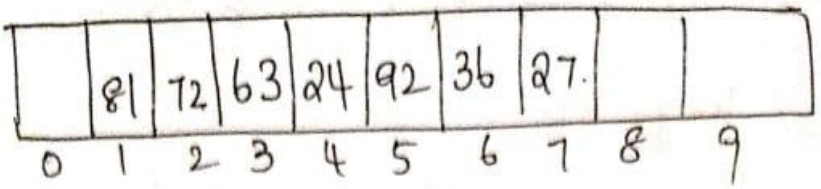
$i = 2 \quad h'(K) = (h(K) + i)$

$= (2+2) \% 10 = 4$ X collision

$i = 3 \quad h'(K) = (h(K) + i)$

$(2+3) \% 10 = 5 \rightarrow$ available

→ The element 92 is inserted at Ind



→ Insert key (101)

$h(101) = 101 \% 10$

$= 1$ X collision.

$i=0$ $h'(101) = (0+1)\%10$

$= 2$ X collision

$i=2$ $h'(101) = (2+1)\%10$

$= 3$ X collision

$i=3$ $h'(101) = (3+1)\%10$

$= 4$ X collision

$i=4$ $h'(101) = (4+1)\%10$

$= 5$ X collision

$i=5$ $h'(101) = (5+1)\%10$

$= 6$ X collision

$i=6$ $h'(101) = (6+1)\%10$

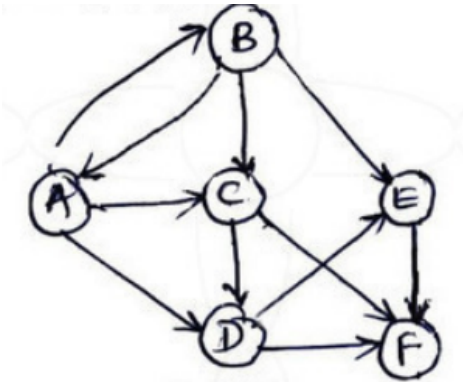
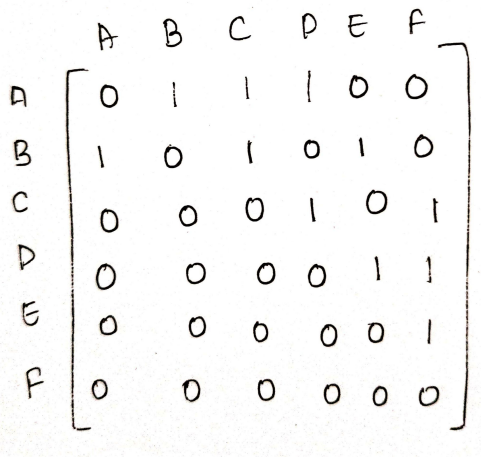
$= 7$ X collision

$i=7$ $h'(101) = (7+1)\%10$

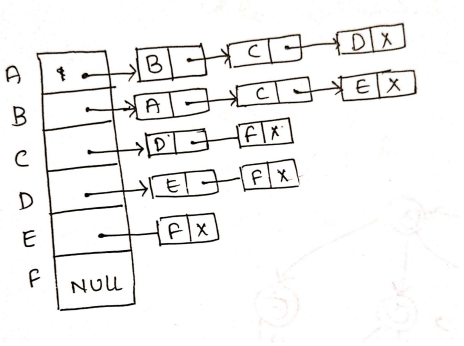
$= 8$

→ The element is inserted at the index 8.

	81	72	63	24	92	36	27	101	
0	1	2	3	4	5	6	7	8	9

6 (a)	 For the given Graph, 1. Represent it using an adjacency Matrix and adjacency List 2. Find the Indegree and Outdegree of every Node. Adjacency Matrix …… **[1 Mark]**  adjacency List ……. [**1 Mark]** A: B, C, D B: A, C, E C: D, F D: E, F E: F F: NIL	[04]	CO5	L2

	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	1	0	1	0
C	0	0	0	1	0	1
D	0	0	0	0	1	1
E	0	0	0	0	0	1
F	0	0	0	0	0	0



Indegree ……**[1 Mark]**

For the given Directed graph the Indegree of nodes are as follows:

A	1
B	1
C	2
D	2
E	2
F	3

Outdegree ……..**[1 Mark]**

For the given Directed graph the Out degree of nodes are as follows:

A	3
B	3
C	2
D	2
E	1
F	0

(b)	Write a C program to perform the following: 1. Depth First Search (DFS) ……**. [3 Marks]** 2. Breadth First Search (BFS)..... **[3 Marks]** **Solution:**	[06]	CO4	L2

```
#include<stdio.h>
# define MAX 10
int adj[][MAX]={{0,1,0,1,0},{1,0,1,1,0},{0,1,0,0,1},{1,1,0,0,1},{0,0,1,1,0}};
void bfs()
{
   int visited[MAX]={0};
   int start=0;
   int r,f,i;
   r=f=-1;
   int q[MAX];
   q[++r]=start;
   visited[start]=1;
   while(r!=f)
   {
      start=q[++f];
      printf("%d-",start);
      for(i=0;i<MAX;i++)
      {
         if(adj[start][i]==1 && visited[i]==0)
         {
            q[++r]=i;
            visited[i]=1;
         }
      }
   }
}
void dfs(int start)
{
   int visited[MAX]={0};
   int stack[MAX];
   int top=-1,i;
   printf("%d->",start);
   visited[start]=1;
   stack[++top]=start;
   while(top!=-1)
   {
      start=stack[top];
      for(i=0;i<MAX;i++)
      {
         if(adj[start][i] && visited[i]==0)
         {
            stack[++top]=i;
            printf("%d->",i);
            visited[i]=1;
            break;
         }
      }
      if(i==MAX)
      top--;
   }
}

int main()
```

	{
	printf("BFS:");
	bfs();
	printf("\nDFS:");
	dfs(0);
	return 0;
	}			

CI **CCI** **HOD**

PO Mapping

Course Outcomes		Modules covered	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1																		
CO2																		
CO3																		
CO4																		
CO5																		

COGNITIVE LEVEL	REVISED BLOOMS TAXONOMY KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.

L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PROGRAM OUTCOMES (PO), PROGRAM SPECIFIC OUTCOMES (PSO)				CORRELATION LEVELS	
PO1	Engineering knowledge	PO7	Environment and sustainability	0	No Correlation
PO2	Problem analysis	PO8	Ethics	1	Slight/Low
PO3	Design/development of solutions	PO9	Individual and team work	2	Moderate/ Medium
PO4	Conduct investigations of complex problems	PO10	Communication	3	Substantial/ High
PO5	Modern tool usage	PO11	Project management and finance		
PO6	The Engineer and society	PO12	Life-long learning		
PSO1	Develop applications using different stacks of web and programming technologies				
PSO2	Design and develop secure, parallel, distributed, networked, and digital systems				
PSO3	Apply software engineering methods to design, develop, test and manage software systems.				
PSO4	Develop intelligent applications for business and industry				