Date / /
Page No
Shivalal

Data Structure :-

Data Structure is used to organize & store the data not only elements to be stored but also their relationship b/w them.

It is divided into two types :-

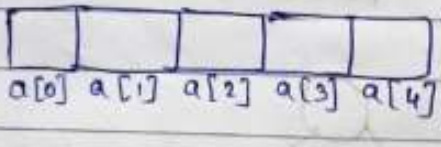
① **Primitive Data Structure ⇒**
These are just like int, char, float, string, Boolean etc.

② **Non-primitive Data Structure ⇒**
It is derived from Primitive Data Structure

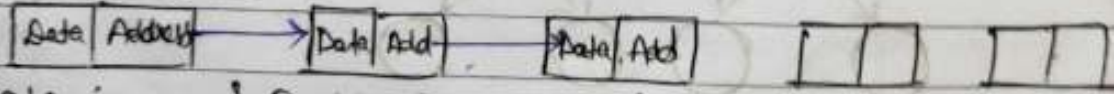
- (i) Linear Data Structure — { Data Stored in Sequencial form }
- (ii) Non-Linear Data Structure — { Data Stored in Unsequencal form }

(i) Linear Data Structure :-

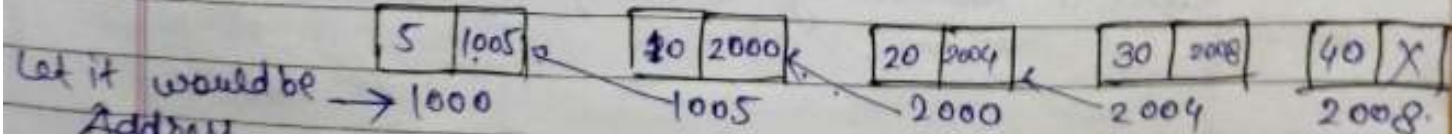
e.g :- ① Array =



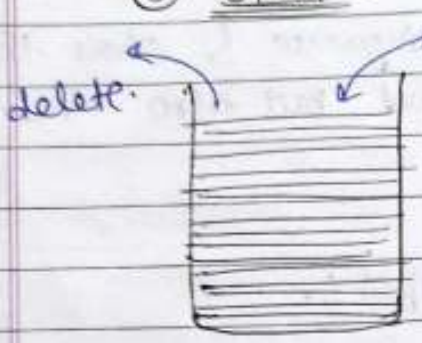
② Linked list =



if Data is {5, 10, 20, 30, 40}
then it will be stored as in linked list



③ Stack ≡



Insertion & Deletion possible only from Single side. ex.

} eg. plates }

④ Queues ≡

Deletion & Insertion in Sequencial Manner.

} eg. Reservation System }

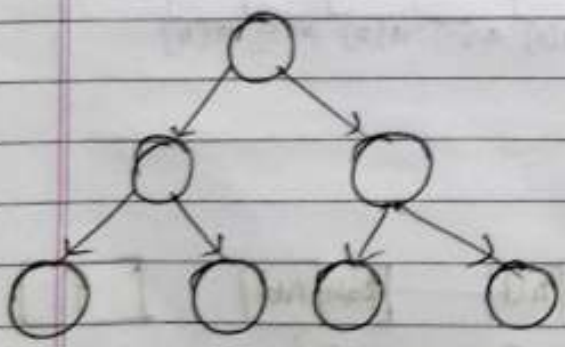
(ii) Non-Linear Data Structure :-

→ One data element attached with many data element

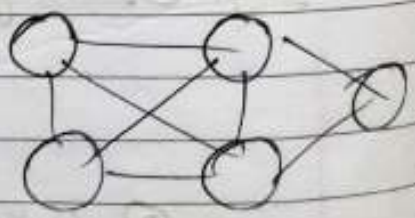
→ Data in Sequencial form.

eg:-

Tree , Graph.



} To Connect Different cities (like in Networking Problem) }



} for like Shortest Path }

Date / /
Page No.
Shivalal

Basic Terminology ⇒

* **Entity :-**
It is something that has certain attributes or properties which may be assigned values. Values can be either Numeric or Non-Numeric.

* **Entity Set :-**
It is a collection of similar entities like Student, employees, Data item..

* **Data :-** Set of values

* **Data element :-**
  - **Elementary data**
    which can't be divided like Roll. No., ID etc.
  - **Grouped data.**
    which can be divided into some sub elements like first Name - Middle Name - Last Name etc.

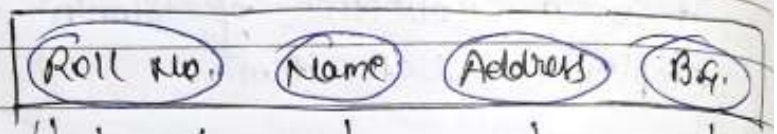
* **Field :-**
Single elementary unit of information representing the attribute of the entity.

* **Record :-**
It is a collection of field values of a given entity.

* **File :-**
It is a collection of records of the entities in a given entity set.

for eg^e - IMS

Entity Set = { Student , Faculty , Staff , ——— }
Entity = Student
Record =



File =

Operations on Data Structure ⇒

① **Traversing (Display) ⇒**
Accessing each data exactly once in the data structure so that each data item is traverse

② **Searching ⇒**
Finding the location of data within the Data Structure which satisfy the searching condition OR Criteria.

③ **Insertion ⇒**
Adding a New Data in the Data Structure.

④ **Deletion ⇒**
Removing the Data from Data Structure.

Date / /
Page No.
Shivalal

⑤ **Sorting :-**
Arranging the data in some logical order.

⑥ **Merging :-**
Combining the data of two different files into a single sorted file.

Why we use data Structure ?

① It gives different levels of Organisation of data.
② It tells us how data can be stored & accessed in its elementary level.
③ It provide operations on group of data.
④ It manage huge amount of data efficiently & provide fast accessing methods.

⇒ **Algorithm :-**
It is a well defined computational procedure that takes some values as a input & produce a set of values as output.

Properties of Algorithm :-

① **Input :-** Algo has Zero OR No Input.
② **Output :-** An Algo have One OR more output.
③ **Finiteness :-** Algo must be terminate after finite No. of Steps.
④ **Definiteness :-** Each Step in the Algo must be well defined.
⑤ **Effectiveness :-** All operations can be carried out exactly in finite time.

⇒ **Analysis of Algo :-**

* **Complexity of Algorithm ⇒**
The Complexity of Algo is a function which gives the running time OR space in terms of the input size.

There are two type of Complexity

① Time Complexity ⇒
The amount of time required by an algo to run upto its completion.
It is also divided into three

i) The Best Case ⇒
The input for which it will execute for a Minm number of time.

ii) Worst Case ⇒
The input for which it will execute for a Maxm No. of time.

iii) Average Case ⇒
It is in b/w Best and Worst Case.

② Space Complexity ⇒
The amount of space required by an algo to run upto its completion called space Complexity.

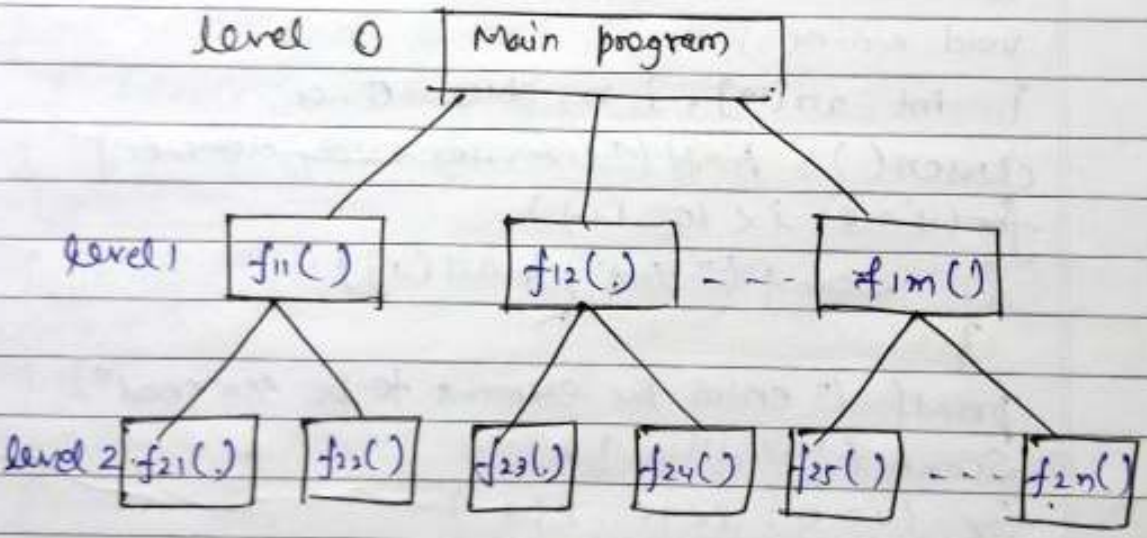
Time-space tradeoff :-
Time-space tradeoff is a situation where the memory used can be reduced at the cost of slower program execution OR vice versa.

The Computation time can be reduced at the cost of increased memory use. for eg :-
If the data is stored uncompressed it take more space but less time. but if the data is stored compressed using compression algorithm it reduce space but take more time.

23/08/16
Date / /
Page No.
Shivalal

Dynamic programming is another example where time complexity can be reduced by using more memory.

Abstract Data type (ADT's) ⇒
It allows various operations on the data to access & change it.
A big problem can not be written in single piece of program it is broken down in small modules called fn each module is developed ~~into~~ independently.



When the program is heirarical organized than the Main prog. utilise the services of the functions at level 1. Similarly level 1 utilize services of the function at level 2.
The main program uses the services at the next level of function without knowing implementation Details.
So the level of abstraction is created, when the abstraction is created at any level our concerned is "what it can be do" not "How to do"?

Array ⇒

Collection of all elements of the same type & All elements are stored using Continous Memory Allocation.

Syntax ⇒

```
data type v.name[size];
```

for eg ⇒ `int array[10];`

Memory used = 20 bytes

Q10. WAP in C for linear search.

```
#include <stdio.h>
#include <conio.h>
void main();
{ int arr[10], i, n, check=0, a;
clrscr(); printf("enter the array element");
for(i=0; i<10; i++)
  { scanf("%d", &arr[i]);
  }
printf("enter the element to be searched");
scanf("%d", &n);
for (i=0; i<10; i++)
  {
    if (arr[i]==n)
      { a=i;
        check=1; break;
  }
if (check==0)
{ printf("Not found");
}
else
{ printf("No. found at %d position", a+1);
} getch();
}
```

=> WAP in C for linear search using function.

```
# include <stdio.h>
# include <conio.h>
int linear_search (int [], int, int);
void main()
{
int ar[10], i, n, check, a;
printf ("enter the array elements");
for (i = 0; i < 10; i++)
  { scanf ("%d", &ar[i]); }
printf ("enter the element to be searched");
scanf ("%d", &n);
check = linear_search (ar, 10, n);
if (check == -1)
{ printf ("not found"); }
else
{ printf ("No found at %d position", check+1); }
getch();
}
int linear_search (int f[], int b, int s)
{
int p, check;
for (p=0; p<b; p++)
{
if (f[p] == s)
  {
    check = p;
  return (check);
  } }
return (-1); }
```

Linear Search by pointer:-

```
Linear_search (int*, int, int);
main()
{
int array[50], n, search, pos = -1, i;
printf("Enter size of array");
Scanf("%d", &n);
for(i=0; i<n; i++)
  {
    Scanf("%d", &array[i]);
  }
printf("Enter No. to be searched");
Scanf("%d", &search);
pos = Linear_search(array, n, search)
if(pos==-1)
    printf("Not found");
else
 printf("%d", pos+1);
getch();
Linear_search(int *a, int n, int search)
{
int i; pos;
for(i=0; i<n; i++)
 {
   if(*(a+i)==search)
    {
      pos = i;
      return(pos);
      break;
    }}
return -1;
}
```

05/Aug/2016.

2-D Array ⇒

Syntax ⇒

data type . V. name [row size] [col size]

for eg. int m, [2] [3]

total elements = 6.

representation of 2-D array in the memory: it can be stored either :-

i) Column by col, called column major order.
ii) Row by row, called Row major order.

By default :- Row by row

for eg ⇒

7 [0][0]	6 [0][1]	5 [0][2]
8 [1][0]	9 [1][1]	2 [1][2]
3 [2][0]	4 [2][1]	7 [2][2]

Address calculation in 2-D array ⇒

Length = UB + LB + 1

If data stored column by column then

$$Loc(A[J][K]) = Base(A) + w[M(K - L_2) + (J - L_1)]$$

Row major order :-

$$Loc(A[J][K]) = Base(A) + w[N(J - L_1) + (K - L_2)]$$

M → No. of Rows

N → No. of Columns

Date / /
Page No.
Shivalal

L_1 → Lower bound of Row
L_2 → Lower bound of Columns
Base → Base Address of matrix
W → No. of words per Memory location

Q- Consider the two dimensional array A[4---7, -1---3] requires 2-bytes of storage space for each element. if the the array is stored in row major form then Calculate the address of A[6,2]. the base address is 100.

Here J = 6 M → 4---7
K = 2. N → -1---3

$Loc(A[6][2]) = 100 + 2[5(6-4) + (2+1)]$
$Loc(A[6][2]) = 126$

for No. of Columns,
length = UB - LB + 1
3 - (-1) + 1
length = 5

Q Consider a 2-D array A[-20---20, 10---35] requires 1-byte of storage space. if Base address is 500 then Calculate the address of A[0][30] if it is stored in column major form.

J = 0, K = 30, Base = 500, W = 1.
M → -20---20
N → 10---35

$Loc[A[0][30]] = 500 + 1[41(30-10) + (0+20)]$
$= 500 + [820 + 20]$
$= 1340$

length = 20 + 20 + 1 = 41

Que. Consider the array A[4][5]. If the base Address is 1020. Then calculate the address of A[3][4]. when array stored in row major form.

I = 3 w = 1

k = 4 Base = 1020 $L_1 = 0$

Length = ~~UB − LB + 1~~ 5 $L_2 = 0$

$$Loc[3][4] = 1020 + 1[5(3-0)+(4-0)]$$
$$= 1020 + 19$$
$$= 1039$$

Que. The given 2-D array Z(2:9, 9:18) stored in column major form, Base address is 100. w → 4 byte. Calculate the Address of Z(4,12)

I = 4 w = 4

k = 12. leng = 9 − 2 + 1

Base → 100 = 8

$$Z[(4)(12)] = 100 + 4[8(12-2)+(4-2)]$$
$$= 100 + 4[24+2]$$
$$= 100 + 104$$
$$= 204 \quad \text{Ans}$$

8/8/2016

Date / /
Page No.
Shivalal

Address calculation in Linear array ⇒

$$Loc(A[k]) = Base + w(k - L)$$

↳ lower bound

Ques - Consider the array A(5:50), B(-5:10) & C(18). Calculate the No. of element in each array if the base address each. arry is 300 & w = 4. Calculate the value of A(15), A(35) & A(55)?

Length = UB − LB + 1

L(A) = 50 − 5 + 1 = 46
L(B) = 10 + 5 + 1 = 16
L(C) = 18
Base = 300
w = 4

A(15) = 300 + 4(15 − 5)
= 300 + 40
A(15) = 340

A(35) = 300 + 4(35 − 5)
= 300 + 120
= 420

~~A(55) = 300 + 4(55 − 5)~~
= 300 + 200
= 500

A(55) > length of A(5:50)
Hence it exceeds the limit of array

Address calculation in Multidimensional Array ⇒

An n-dimensional $m_1 \times m_2 \times m_3 \cdots m_n$ array is a collection of $m_1, m_2, \cdots m_n$ data elements in which each element is specified by a list of n integers like $k_1, k_2 \cdots k_n$ → Subscript.

$$1 \le k_1 \le m_1, \quad 1 \le k_2 \le m_2 \cdots 1 \le k_n \le m_n$$

The element of array with subscript is denoted by
Array $[k_1, k_2, \cdots k_n]$.

Address calculation in 3-D array ⇒

① Column major order

$$Loc(A[k_1, k_2, k_3]) = Base(A) + w((((E_3 L_2) + E_2) L_1 + E_1$$

② Row major order

$$Loc(A[k_1, k_2, k_3]) = Base(A) + w(((E_1 L_2) + E_2) L_3 + E_3)$$

L_i = UB − LB + 1
E_i = effective index of L_i & calculated by
E_1 = K_1 − Lower bound.
E_2 = K_2 − Lower bound
E_3 = K_3 − Lower bound.
⋮

Date / /
Page No.
Shivalal

Qu. Consider the 3-D array M[2:8, -4:1, 6:10] is stored in row major form. The base address is 200 & the element size is 4. (i) Calculate the total No. of elements in the matrix.

(ii) Calculate the address of M[5, -1, 8]
k_1 k_2 k_3

Total No. of element,

$L_1 = 8 - 2 + 1 = 7$

$L_2 = 1 + 4 + 1 = 6$

$L_3 = 10 - 6 + 1 = 5$

total No. of element $= 7 \times 6 \times 5 = 210$

$E_1 = 5 - 2 = 3$

$E_2 = -1 + 4 = 3$

$E_3 = 8 - 6 = 2$

$loc[A[5, -1, 8]] = 200 + 4(((3 \times 6) + 3)5 + 2)$

$= 200 + (107) \times 4$

$= 200 + 428$

$= 628$

B(1:8, -5:5, -10:5)

Qu. Consider a 3-D array the base address 400, element size 4. Calculate total No. of element & Address of B[3, 3, 3]

total No. of element

$L_1 = 8 - 1 + 1 = 8$

$L_2 = 5 + 5 + 1 = 11$

$L_3 = 5 + 10 + 1 = 16$

total element $= 8 \times 11 \times 16 = 1408$

$E_1 = 3 - 1 = 2$

$E_2 = 3 + 5 = 8$

$E_3 = 3 + 10 = 13$

Case-1 ⇒

In Column major

Case-2 ⇒

$$Loc(A[3,3,3]) = 400 + 4(((13 \times 11) + 8)8 + 2)$$

$$= 400 + 4840$$

$$= 5240$$

Case-3 ⇒

In Row major,

$$Loc(A[3,3,3]) = 400 + 4(((2 \times 11) + 8)16 + 3)$$

$$= 400 + 1972$$

$$= 2372$$

Q. Multiplication of Matrix.

I	II
m x n	p x q

m = q

n or p

```
for(i=0; i<m; i++)
  { for(j=0; j<q; j++)
      {
        for(k=0; k<n; k++)
      { Sum = Sum + A[i][j] * B[j][k]
        Sum = 0
      }
  }
}
```

Case-4 ⇒

Date / /
Page No.
Shivalal

Calculating Running time of Program :-

Case-1 ⇒ A program without loop

```
x = x+1 ;  → O(1)
y = y+3 ;  → O(1)
```
} O(1) → Constant

Case-2 ⇒ Program Consisting a loop :

```
for (i=1 ; i<=n ; i++)  → n+1
    x = x+1 ;           → n
```
} n+1+n = 2n+1 = O(n)

Neglect constant part

Case-3 ⇒ Nested loop.

```
for (i=1; i<=n ; i++)          → n+1
    for (j=1 ; j<=n ; j++)     → n(n+1) → n²
        x = x+1                → n²
```

= n+1+n²+n+n²

2n²+2n+1

= O(n²) Ans

Case (iv) Large program (Multiple loops)

```
x = 500 - O(1)
y = 600 - O(1)
for (i=1 ; i<=n ; i++)  { z = x+y ; }   → O(n+1) → O(n)
for (i=1 ; i<=n ; i++)  { z = z+1 ; }   → O(n)
for (i=1 ; i<=n ; i++)
    { for (j=1 ; j<=n ; j++)
        { x = x+1 ; } }                 → O(n²)
```

So O(n²) Ans

Case-2 When in the loop the variable is either multiplied or divided during each iteration of the loop.

```
for (i = 1; i < 100; i = i*2)
    Statement.
```
O(log n)

```
for (i = 1; i < 100; i = i/2)
    Statement.
```

Case-3

```
for (i = 1; i ≤ n; i++)
    for (j = 1; j ≤ n; j = j*2)
        Statement.
```
O(n+1) ⇒ O(n)

O(log n)

= O(n log n)

Ques - Calculate the worst case time Complexity of Program.

```
int f2(int n)
{
  Sum = 0
  for (i = 0; i < n; i++)
    for (j = 0; j < i; j++)
      Sum = Sum + j
}
```

i	No. of time j added to sum
0	0
1	1
2	2
⋮	⋮
n-1	n-1

$0 + 1 + 2 + \cdots + (n-1)$

$= \frac{n(n-1)}{2} = \frac{n^2 - n}{2}$

Hence $= O(n^2)$ Ans

Sparse matrix and its implementation.

$$4 \times 4 = \begin{bmatrix} 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 8 \\ 4 & 0 & 0 & 0 \end{bmatrix}$$

⇒ Sparse matrix

$16 \times 2 = 32$ bytes

if element which are zero $> \frac{n}{2}$ ⇒ Sparse.

$< \frac{n}{2}$ ⇒ Dense.

for such type of matrix Sparse Matrix will be

	V	i	j
No. of Non zero element ←	4	4	4 → order of matrix
values ←	3	0	2
	5	1	3 → position
	8	2	3
	4	3	0

Sparse matrix is a matrix in which most of the element are zero.

The first Row Represent dimension of Matrix & No. of Non-Zero values.

& Second Row Onwards giving the value of Non-Zero number and its position.

Implementation of Sparse Matrix:

```
Main()
{
int A[10][10], B[10][3], m, n, S=0;
printf("Enter row and column");
Scanf("%d%d", &m, &n);
for (i=0; i<m; i++.)
   {
    for (j=0; j<n; j++.)
    {
     printf("Enter %d row and %d column", i, j);
     Scanf("%d", &A[i][j]);
   }}
printf("the given matrix is");
for (i=0; i<m; i++)
    {
     for (j=0; j<n; j++)
      {
       printf("%d", A[i][j]);
      }
     printf("\n");
{
for (i=0; i<m; i++.)
  }
   for (j=0; j<n; j++)
  {
    if (A[i][j] != 0)
   {
     B[S][0] = A[i][j];
      B[S][1] = i;
      B[S][2] = j;
     } S++;
} }
```

```
for (i=0; i<5; i++)
{for (j=0; j<3; j++)
 {
  printf ("%d", B[i][j]);
 }
  printf ("\n");
}
```

Memory Allocation in C

1. Compile time or Static Memory allocation (Array)
2. Runtime or Dynamic memory allocation. (Pointer)

Static Memory Allocation	Dynamic Memory Allocation
The memory allocated before run time	Memory allocated at run time
The programmer must know in advance the amount of memory needed.	The programmer is not expected to have knowledge about amount of Memory required.
Wastage of Memory in Compile time.	No wastage of Memory.
No function for Memory request is needed.	Predefined functions for Memory allocation request will be used such as malloc(), calloc().
Static allocation is fast.	Dynamic allocation is slow

The space is allocated once and is never released.

eg:. Array, Structure.

Programmer can free the block of memory if it is not needed.

eg. linked list, pointer

Predefined functions in Dynamic Memory allocation

<alloc.h> ⇒ Header file.

1. malloc( ) → default value = Garbage
2. Calloc( ) → default value = Zero.
3. realloc( )
4. free( )

1. malloc( ) ⇒

The malloc() function allocates a block of memory in byte, it request to the RAM for the memory if the request is granted, it returns a pointer to the first block of that memory, otherwise it returns 'NULL' pointer. it always returns a void pointer. The default value is Garbage.

Syntax ⇒

```
{ int *ptr.
ptr = (type cast *) malloc (Size); }
```

ptr is the name of pointer that holds the starting address of allocated memory block.
'typecast' is a datatype into which returned pointer (void) is to be converted.
Size = No. of elements x Size of each element

Date / /
Page No.
Shivalal

② calloc ( ) ⇒

It is also used for dynamic memory allocation, it takes two arguments, first is the No. of elements & second is the size of the element (,). It returns void pointer and the default value of variable is zero.

Syntax ⇒ ptr = (type cast*) calloc (10, 2)

2 byte → 10 slots allocated

③ realloc ( ) ⇒

It is used to reallocate some memory at run time.

Syntax ⇒

ptr = realloc (ptr, new size) ← always in bytes

ptr = pointer holding the starting address of memory block.

new size = size in byte, it may be smaller or greater according to requirement.

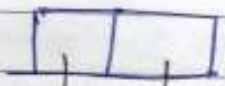
④ free ( ) ⇒

The free( ) is used to deallocate the previously allocated memory.

Syntax ⇒ free (ptr)

Linked List ⇒

It is a Linear data Structure, In this each element pointing to the Next element and each element is called Node.
The Node is divided into two parts.



* next → contain the address of Next element.
info - store information.

Advantages :-

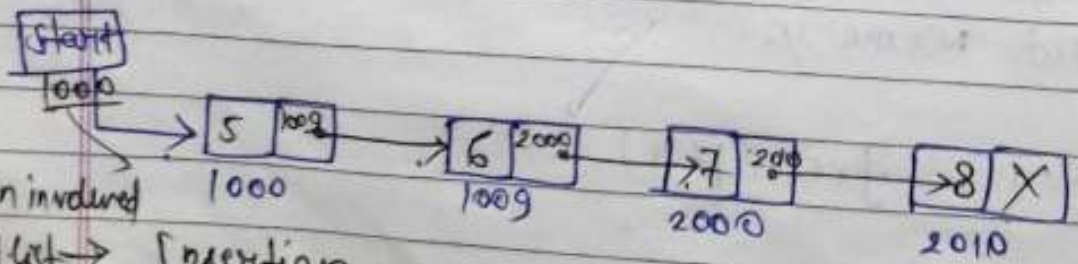
① We can grow OR shrink the memory.
② Memory is not pre allocated.
③ It is allocated whenever it is necessary.
④ Insertion & Deletion are easy.

Disadvantage :-

Because of More fields, more Memory Space is required.

Types of Linked List ⇒

① Singly Linked List ⇒



Operation involved in Linked List → Insertion → Deletion

Insertion: first, last, Anywhere
Deletion: first, last, Anywhere

Start ⇒ Address
Start → info ⇒ value
Start → next ⇒ Next Address

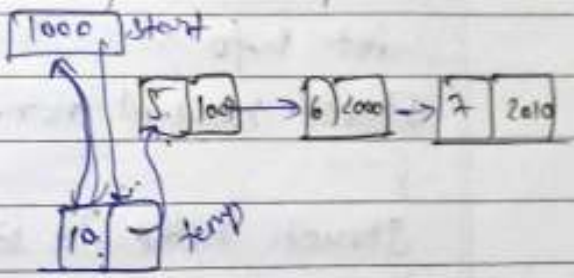
Date / /
Page No.
Shivalal

→ Traversing ⊕ Reverse

→ Create.

For Insert First

Diagramatically,




```
Struct node
{
 int info;
Struct node *next;
}
Struct node *start;
insert_first(.)
{
Struct node *temp=(Struct node *) malloc (Size of (Struct node*));
printf ("Enter element");
Scanf ("%d", temp → info);
if (Start == NULL)
   {
      temp → next = NULL;
       Start = temp;
   }
else
 {
   temp → next = Start;
       Start = temp;
 }
}
```

Date / /
Page No.
Shivalal

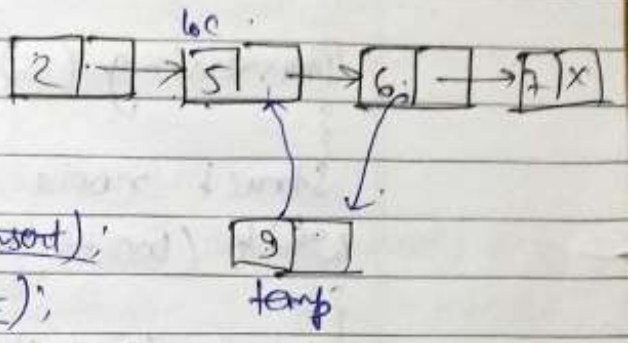
for Insert last

Diagrammatically,


```
Struct Node.
{
 int info;
Struct node * next;
}
Struct node * start.
insert_last ( ) .
{    Struct node * loc;
Struct node *temp = (struct node *) malloc (Size of
                                  (struct node*));
printf ("Enter element");
scanf ("%d", temp → info);
  temp → next = NULL;
  if (start == NULL)
    {
      start = temp;
    }
else.
{
  loc = start;
while (loc → next != NULL)
  {
    loc = loc → next;
  }
 loc → next = temp;
}}
```

Date / /
Page No.
Shivalal

Insert at any position ⇒



~~Printf ("Enter element to be insert");~~
~~Scanf ("%d", temp → info);~~
After the value inserted = y
while (loc → info != y)
{
loc = loc → next
}
temp → next = loc → next
loc → next = temp.

Before the value inserted = y.
while (loc → next → info != y)
{
loc = loc → next
}
temp → next = loc → next.
loc → next = temp

Program ⇒

```
insert_loc( )
{
 int x;
 Struct node * loc;
Struct node * temp = (Struct node*) malloc (Size of (Struct Node*))
 printf ("Enter element");
 Scanf ("%d", & temp → info)
 print ("Enter element after which you want to insert");
 Scanf ("%d" & y);
```

For Display.

```
Traversing ( )          //display
{
Struct node * loc = start;
while (loc -> next != NULL)   OR (loc != NULL)
{
 printf (" %d", loc -> info);
   loc = loc -> next;
}
printf (" %d", loc -> info);
}
```

Algorithm to delete first Node,

```
delete_first ( )
{
Struct node * temp = start;
 if (start == NULL)
   { printf ("List empty");
   }
  else.
  {
     start = start -> next.
      free (temp);
  }
}
```

Algorithm to delete last node:

```
delete last ( )
{
Struct node *temp = start ;
if ( temp -> next != NULL) Struct node *q ;
   { temp = temp -> next ;   if (start == NULL)
   }                         {
else                          printf ("List empty");
                            }
                            else.
                          { while (temp -> next != NULL)
                            {
                              q = temp;
                             temp = temp -> next
                            }
                          q -> next = NULL
                             free (temp);
                          }
```

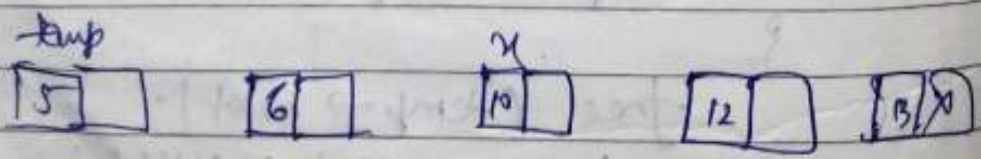
Without Using Another pointer

```
while ( temp -> next -> next != NULL )
{
    temp = temp -> next);
}
    free ( temp -> next );
      temp -> next = NULL;
```

Algorithm for deleting specific Node.

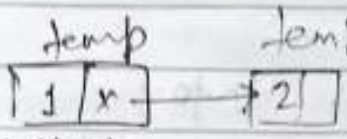
~~delete_loc()~~ delete_specific node()
{
int y; struct node *temp1;
Struct node * ~~loc~~ temp = start;
~~Struct node *temp =~~ int x;
printf ("Enter node which you want to delete");
Scanf ("%d", &x);

~~while (loc -> info. ! = y)~~
~~{~~
~~loc = loc -> next~~
while (temp -> next ! = NULL)
{
if (temp -> next -> info == x)
{
temp 1 = temp -> next;
temp -> next = temp -> next -> next;
free (temp 1);
}
temp = temp -> next;
}
}



Algorithm to create Singly linked list -

```
Struct node
{
    int info;
    Struct node * next ;
{
    Struct node * start;
Insert
Create ( )
{ char ch;
 Struct node *temp = (struct node * ) malloc (size of(Struct node*)
printf (" Enter first element");
Scanf (" %d", & temp -> info );
   temp -> next. = NULL ;
   start = temp ;
do.
{.
Struct node * temp 1 = (Struct node *) malloc ( Size of (Stuct node*)
 Printf (" Enter next node");
Scanf (" %d", & temp1 -> info);
    temp 1 -> next = NULL ;
        temp -> next = temp 1 ;
            temp = temp 1 ;
printf (" wants to insert more node (Y/N)");
 Scanf (" %c" , & ch);
}
while ( ch == 'Y' || ch == 'y');
}
 temp -> next = NULL ;
}
```



Implementation of linked list.

```
# include <stdio.h>
Struct node.
{
 int info;
struct node * next;
};
Struct node * start;
void main ( )
{ int a; char ch;
do {. printf (" Welcome to linked list windows");
printf (" press '1' for Insert at first position");
printf (" press 2 for Insert at last position");
printf (" press '3' for Insert at specific position");
print (" press 4. for delete first node");
printf (" press 5 for delete last node");
print (" press 6 for delete specific node");
print (" press 07 for Display");
printf (" Enter your choice.");
scanf ("%d" & a);
Switch (a)
 {
 Case 0: create( );
         break;
 Case 1: Insert_first ( );
         break;
 Case 2: Insert_last ( );
         break;
 Case 3; Insert_loc( );
         break;
 Case 4; delete_first( );
         break;
```

printf(" press '0' for Creat a li

Date / /
Page No.
Shivalal

```
case 5 : delete_last();
         break;
case 6 : delete_specific node();
         break;
case 7 : Traversing();
         break;
default : printf("Invalid choice"); break;
}
printf("want to continues");
scanf("%c", &ch);
} while(ch=='y' || ch=='Y');
getch(); }
```

C function to create . create, display.
insert node. insert, display
implement Create, insert
delete, display;

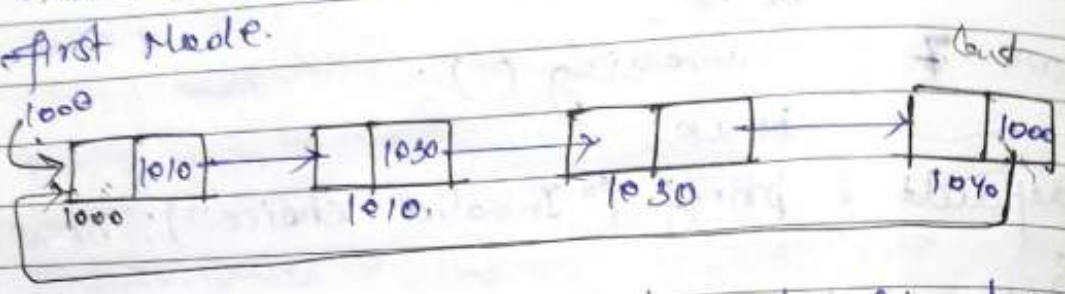
Assignment-1.

Q.1 ⇒ Implement a data structure that support the following operation insert(), findmin(), findmax(), deletemin(), deletemax(), isempty(), make empty : You must use the following algo.

→ Maintain the sorted Array.
→ Insert new item into the correct position in the array, sliding the element over one with position at right.
→ And for deletion remove the item in position zero and sliding over other element to.

(2) Circular linked list ⇒

In Circular linked list there is No Null pointer the last Node will contain the address of first Node.



Disadvantage ⇒ There is No backtraversing in Circular linked list

Algorithm to create a Circular linked list ⇒

```
Crete( )
{ char ch; struct node * last ', struct node * start;
Struct node * temp = (Struct node * ) malloc (sizeof(Struct
printf (" Enter first element");
Scanf ("%d" , & temp → info);
   temp → NEXT = NULL temp.
    start = last = temp;
do
{
Struct node * temp1 = (Struct node*) malloc(Sizeof(Struct
 printf ( Enter next node")
 Scanf ("%d" &temp1 → info);
    last → next = temp1
          last = temp1
printf ("want to continue ");
 Scanf ("%c" &ch);
} while (ch == 'y');
} last → next = start ;
```

Date / /
Page No.
Shivalal

Algorithm to insert at first position in Circular linked list:

```
Struct node
{
 int info
Struct node * next ;
}
 Struct node * start ;
insert_first( )
{
Struct node * temp =(struct node *) malloc( size of (Struct node));
 Printf ("Enter element");
Scanf ("%d", &temp → info );
if (start == NULL)
     {
          temp → next = temp
          start = last = temp ;
     }
else
  {
     temp → next = start ;
          start = temp ;    last → next = start ;
     }
}
```

Algorithm to insert at last position in Circular linked list.

```
insert_last()
Struct node *temp = (Struc
printf ("Enter node");
Scanf ("%d", &temp->info);
if (start == NULL)
 {
  temp -> next = temp
   start = last = temp;
 }
 else
 {
last->temp -> next = start temp ;
last->start = temp;
    last -> next = start;
 }
```

Algorithm to insert at specific position in Circular linked list.
Same as in Singly linked list.

to delete specific Node,
Same as in Singly linked list.

2/10/2016

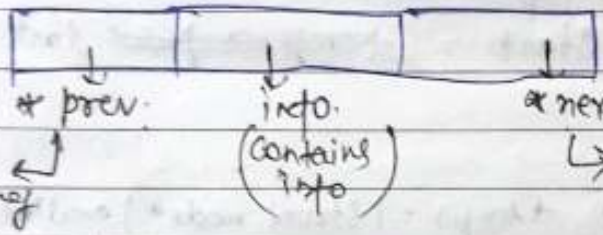
Page No.

Shivalal

Doubly linked list ⇒

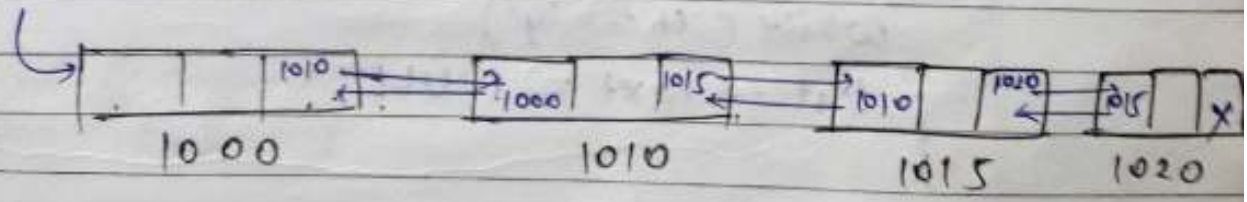
In this list all nodes are linked together by a multiple number of links which help in accessing previous and next node from the given node position. it provides by-directional traversing.

The node is divided in three parts.



Node Creation –

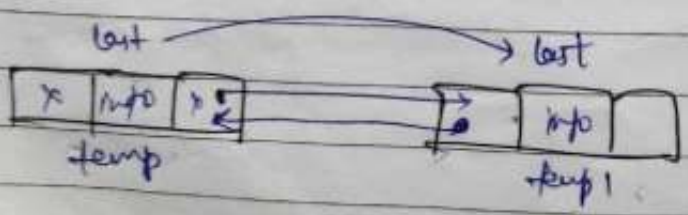
```
{
  int info;
  Struct node * prev;
  Struct node * next;
};
```



Page No.

Algorithm to create doubly linked list;

```
create ( )
{ char ch; struct node *prev; struct node *start;
struct node *temp = (struct node*) malloc( size of (struct node*
  printf ("Enter first element");
  scanf ("%d", &temp->info);
temp->prev = temp->next = NULL;
        start = temp  last = temp;
do
}
struct node *temp1 = (struct node*) malloc (size of (struct node*
printf ("Enter next node");
scanf ("%d", &temp1->info);
last->next = temp1
        temp1->prev = last;
            last = temp1;
printf ("Do you want insert more");
 scanf ("%c", &ch;)
 }
  while (ch == 'y')
   last->next = NULL;
}
```



Algorithm to traverse doubly linked list

```
traversing( )
{
    struct node * loc = start;
    while ( loc -> next ! NULL ) OR (loc ! =NULL)
    {
        printf ("%d", loc -> info);
        loc = loc -> next;
    }
    printf ("%d", loc -> info)
}
```

Algorithm to insert at first position

```
struct node
{ int info;
 struct node * next;
}
 Struct node * start;
insert_first( )
{
struct node * temp = (struct node *) malloc(size of (struct node));
printf (" Enter element");
Scanf ("%d", & temp -> info);   temp -> prev = NULL;
if ( start == NULL )
    { temp -> next = NULL
      start = temp; last = temp.
    }
else
    { temp -> next = start;
      start = temp;
    }
```

Algorithm to insert at last position

```
insert_last( )
{
struct node* temp = (struct node*) malloc (size of (struct node));
printf ("Enter element");
Scanf ("%d", & temp → info)
  temp → next = NULL ;
if (start == NULL)
  {
    temp → prev = NULL
    start = last = temp;
  }
else
 while (loc → next != NULL)
 {
    last → next = temp;
    temp → prev = last;
      last = temp;
} }
```

Insert at specific position

```
insert_specific ( )
{
int y
struct node * loc ;
Struct node * temp ;
printf ("Enter element");
Scanf ("%d", & temp → info)
printf ("Enter element after which you want to insert
Scanf ("%d", & y)
```

```
while (loc -> info != y)
    { loc = loc -> next
    }
temp -> prev = loc -> next -> previous
temp -> next = loc -> next
    loc -> next = temp
    temp -> prev = loc
    temp -> next -> prev = temp;
}
```

Delete specific Node.

```
delete-specific node ( )
{
int y ; struct node * temp 1';
Struct node * temp = start ;
int x.
print f("Enter node which you wants to delete")
scanf ("%d", & x )
while ( temp -> next != NULL)
    {
      if ( temp -> next -> info == x )
      {
       loc -> prev -> next = loc -> next ;
        loc -> next prev = loc -> prev ;
         free ( temp );
      }
```

May or may not work. depend upon Compiler

OR

```
if (start == NULL)
printf ("List empty");
else
 {
 while (temp -> info != x)
 {
   temp1 = temp
   temp = temp -> next;
 }
   temp2 = temp -> next;
    temp1 -> next = temp2;
   temp2 -> prev = temp1;
    free (temp);
}
```

Delete first node.

```
delete_first( )
{
struct node * temp = start;
if (start == NULL)
   {
     printf ("List empty");
   }
      start = start -> next;
      start -> prev = NULL
        free (temp);
    }
}
```

Date / /
Page No.
Shivalal

Delete last Node.

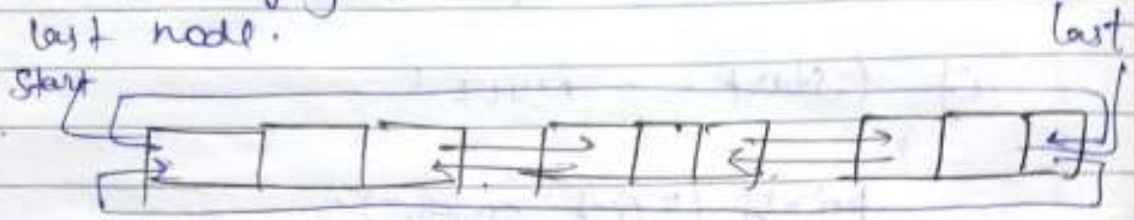
```
delete_last ( )
{
struct node *temp = start;

if (start == NULL)
  {
    printf ("List empty");
  }
else
{
  while ( temp -> next != NULL )
    {
      q = temp;
      temp = temp -> next
    }
    q -> next = NULL;
      free (temp);
}
```

OR.

```
else.
last = last -> prev;
last -> next = NULL;
 free (temp);
 }
}
```

Circular doubly linked list ⇒

Last Node will Contain the Address of first Node. & prev of first will Contain the address of last node.



Insert at first position ⇒

~~Insert_first( )~~

```
Struct node
{
int info;
Struct node * next;
};
Struct node * start;
insert_first ( )
{
Struct node *temp = (Struct node*) malloc ( Size of ( Struct node*))
printf ("Enter element");
scanf ("%d", & temp → info);
if (start == NULL)
    ;
    temp → next = start temp → prev = temp
    temp → prev = last start = last = temp;
}
else
    temp → prev = last;
    last → next = temp.
    start → prev = temp
    temp → next = start.
    start = temp;
}
```

Date / /
Page No.
Shivalal

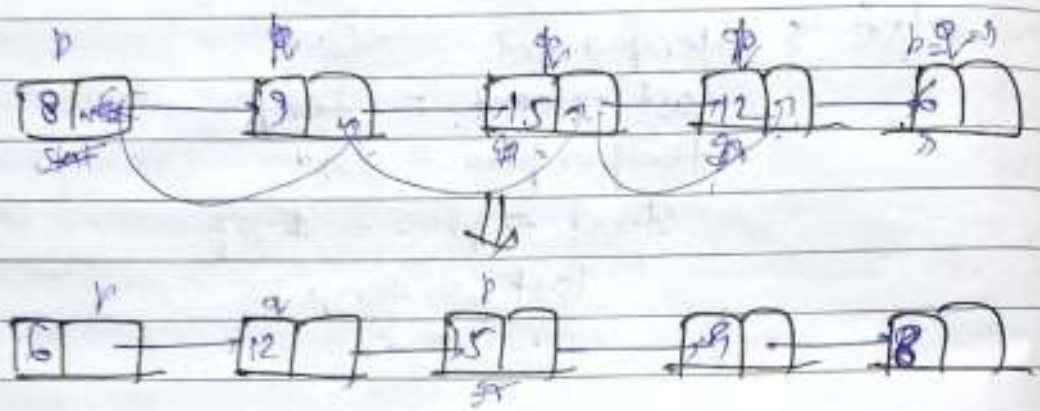
Insert last

```
else {  temp -> next = start;
        last -> next = temp.
        temp -> prev = last
        start -> prev = temp
        last = temp;
     }
```

Delete first

```
else {
        start -> next = start
        start -> prev = last -> next.
        free(temp);
     }
```

Reverse a Singly linked list ⇒




```
reverse( )
{
Struct node *p, *q, *r;
   p = start;
   q = p→next;
 p→next = NULL;
while (q != NULL)
{
   r = q→next;
   q→next = p
     p = q
     q = r
   }
start = p
}
```

Applications of Link list ⇒

→ Linked list are used in polynomial addition and multiplication.

→ Use in time sharing problem solved by the operating system. (Circular link used to solve such type of problem).

circular →
- Used for implementing Queue. (Circular linked)
- Multiplayer board games
- Used for implement advance data structure. like fibonaccitheap.
- [redacted] Escalator.

doubly →
- The browse cache which allow to hit the back button (link list of URL's)
- Undo functionality in photoshop or word.
- Stack, hash table and binary tree can be implemented using doubly linked list.

Polynomial Addition ⇒

Link list are widely used to represent and manipulate polynomials. Polynomials are the expression containing number of terms with non-zero coefficients and exponents

$$P(x) = a_n x_n^e + a_{n-1} x_{n-1}^e + \cdots - a_1 x_1^e + a.$$

ex. $4x^3 + x^2 + 10x + 6$

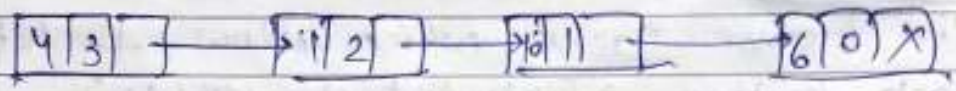
here a_i → non-zero coeffi.
e_i → Exponents.

In linklist representation of polynomial each term is considered as a Node. and the Node

containing three fields. as given below:

the coeff. field hold the coefficient value., expo contain the expo and next will contain the address of next node.

for eg ⇒ $4x^3 + x^2 + 10x + 6$



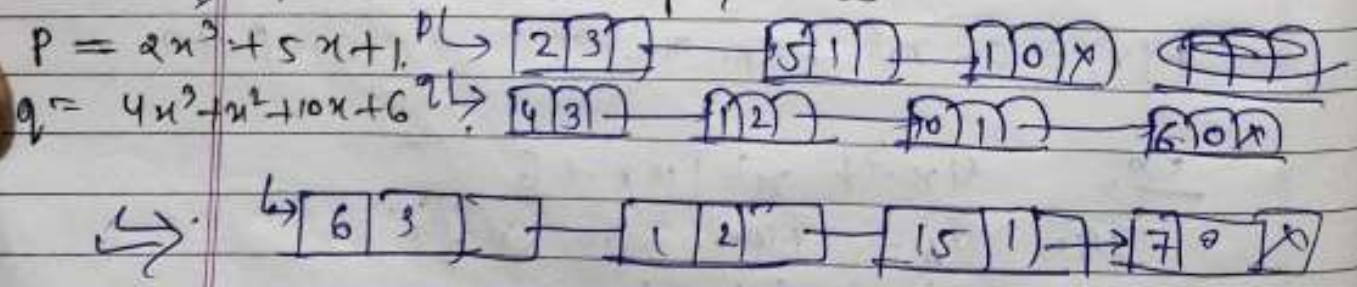
Node Creation.

```
Struct polynode.
{
 int coeff;
 int expo
 Struct polynode * next;
};
```

⇒ Add these two polynomials

$P = 2x^3 + 5x + 1$

$q = 4x^3 + x^2 + 10x + 6$



Date / /
Page No.
Shivalal

Algorithm

Step-① Read the number of terms in the first polynomial p.

② Read the exponent & Coeff of first polynomial p.

③ Read the Number of terms in second poly. q.

④ Read the exponent & Coeff. of second poly. q.

⑤ Set the temp pointers p and q to traverse the polynomials respectively.

⑥ Compare the exponents of polynomials starting from the first Node

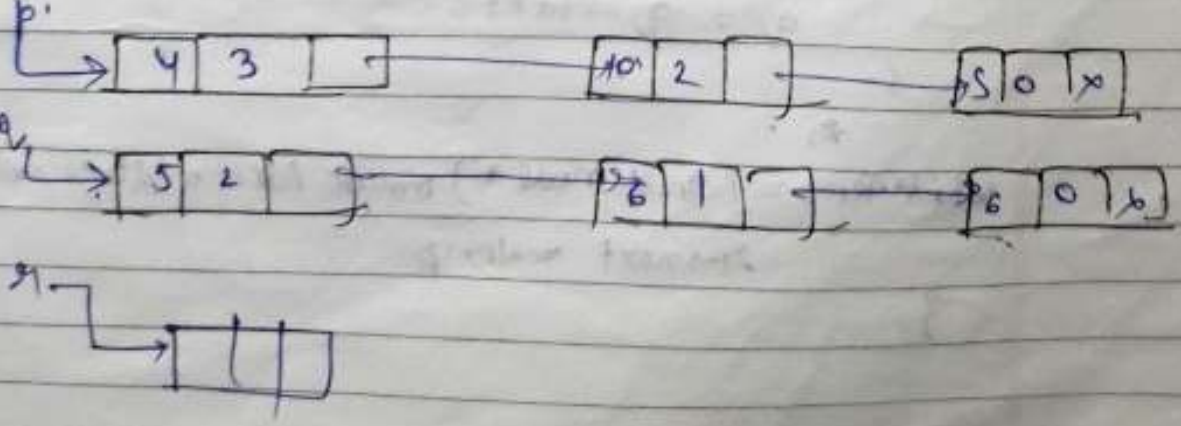
ⓐ if both the exponents are equal, add the coeff. and store it in the resultant linked list.

ⓑ if the exponent of p is less than the exponent of q than q is added to the resultant linked list and move to the next node.

ⓒ if the exponent of q is less than the exponent of p than p is added to the resultant linked list and move to the next node.

ⓓ Append the remaining node either p or q to the resultant linked list.

Q. $4x^3 + 10x^2 + 5$ $\quad 5x^2 + 6x + 6$



```
polynomial - addition( . )
{
Struct node *temp = (Struct node *) malloc (Size of (Struct node
while ( p != NULL && q != NULL)
{
  r = temp ;
  if ( p -> expo == q -> expo)
  {
    temp -> coeff = p -> coeff + q -> coeff
     temp -> expo = p -> expo;
      p = p -> next ;
      q = q -> next ;
  }
else if ( p -> expo > q -> exp)
  {
     temp -> coef = p -> coeff.
     temp -> expo = p -> expo
         p = p -> next
  }
else.
  {
      temp -> coff = q -> coff :
      temp -> expo = q -> expo:
        q = q -> next.
  }
      .
~~while~~ temp = (Struct node *) malloc (Size of (Struct node));
          r -> next = temp
}
```

Case II:

```
while ( p
if ( p == NULL)
{
   while ( q != NULL)
    {
     temp -> coff = q -> coff
     temp -> expo = q -> expo
        q = q -> next.
  struct node *temp = (struct node *) malloc (sizeof (struct node*))
          r -> next = temp.
    }
}
```

Case - II

```
if (q == NULL)
 {
    while (p != NULL)
     {
        temp -> coff = p -> coeff.
        temp -> expo = p -> expo
           p = p -> next.
Struct node *temp = (Struct node*) malloc (size of (Struct node*))
          r -> next = temp.
     }
}
    temp = NULL.
     r -> next = temp
 }
```

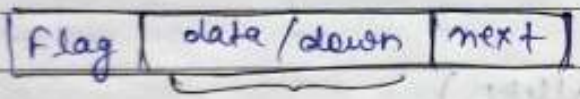
Drawback of polynomial

In link list polynomial addition the only drawback is it can handle only single variable polynomial only.

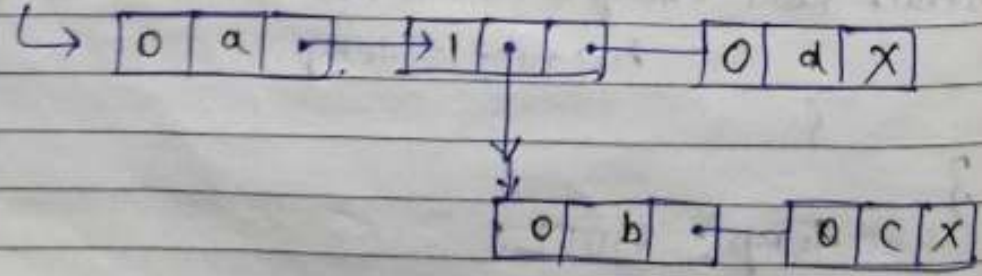
If we have multivariable polynomial than it become complicated. To overcome this problem we use Generalised link list.

Generalized link list ⇒

A Generalized link list A is defined as a finite sequence of n elements $(n \geq 0)$ $a_1, a_2, \ldots a_n$ where a_j – a to m or list of atom, where n is the total number of nodes in the list. The structure of a node is like,

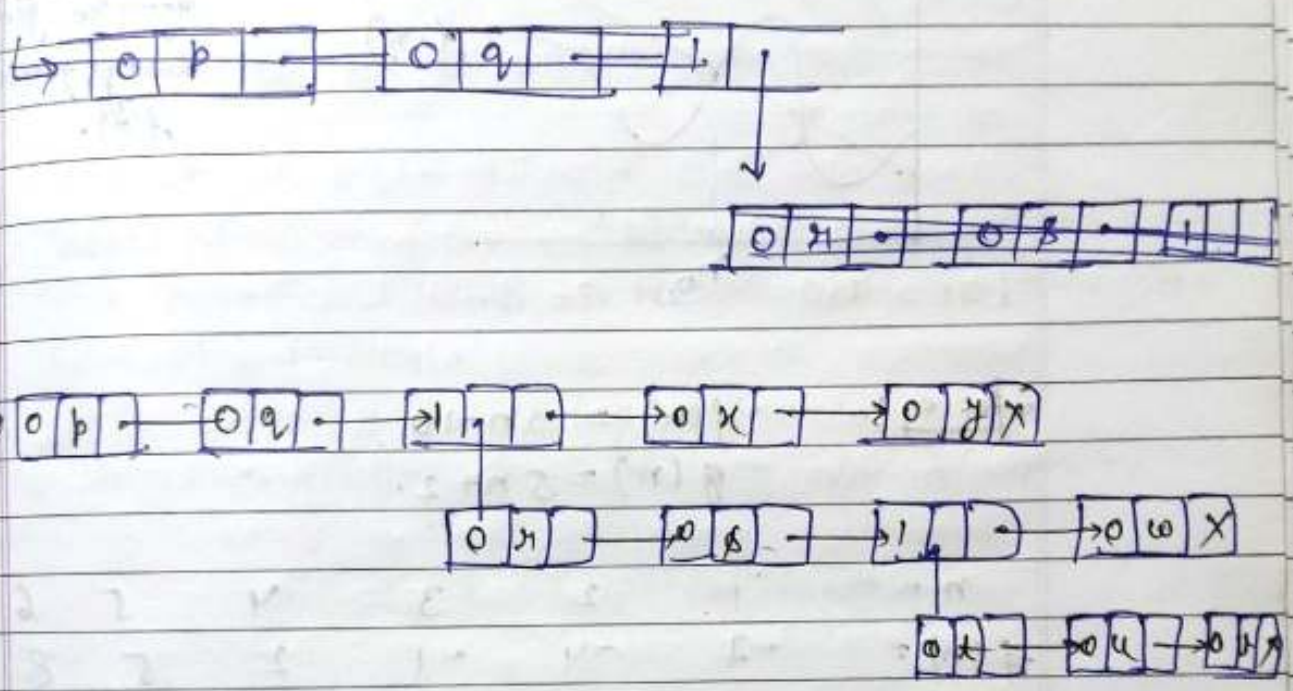


Flag is either 0 or 1, 1 means down pointer exist and 0 means next pointer exist. Means if the first field is zero, it indicates that the second field is a variable; if the first field is one than the second field is down pointer. for eg ① (a, (b, c), d)



Date / / Page No. Shivalal

(2) $g = (p, q, (r, s, (t, u, v), w), x, y)$



Asymptotic Notation ⇒

The Notations we use to describe the Asymptotic running time of an algorithm, are defined in terms of functions whose domains are the set of Natural Numbers, These notations are used in performance analysis and used to categorise the Complexity of algo.

(1) Big 'oh' (O) (Upper bound) →

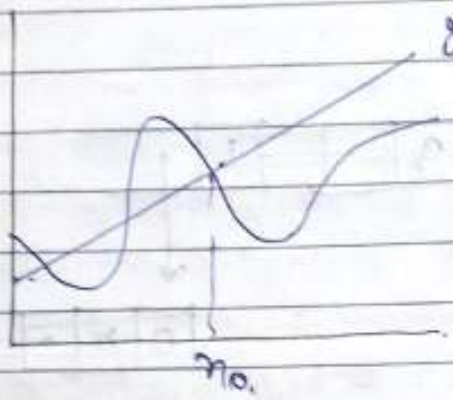
It gives the worst Case Complexity, this Notation gives an upper bound to the function to given a constant factor.

$$f(n) = O(g(n))$$

$$f(n) \leq C\,(g(n)) \qquad n \geq n_0$$

C → constant

Graphically :



after no. g(n) will always greater than f(n).

for eg :- $f(n) = 3n-10$
$g(n) = 5n+2$.

n =	1	2	3	4	5	6
f(n) =	-7	-4	-1	2	5	8
g(n) =	7	12	17	22	27	32

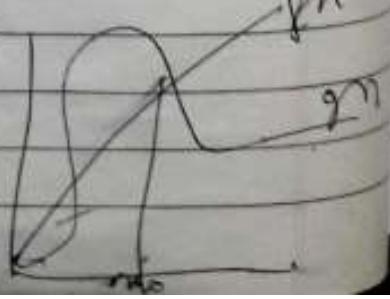
$\therefore$ $\boxed{f(n) \leq g(n) \quad n \geq 1}$

② Big omega (Ω) (lower Bound)
It gives best case complexity this notation gives the lower bound for a function within a constant factor.

$f(n) = \Omega(g(n))$

$\boxed{f(n) \geq C(g(n)) \quad \forall\ n \geq n_0}$

for eg $f(n) = 3n^2+5$
$g(n) = 5n+10$



Date / /
Page No.
Shivalal

$n=$	1	2	3	4	5
$f(n)=$	8	17	32	53	80
$g(n)=$	15	20	25	30	35

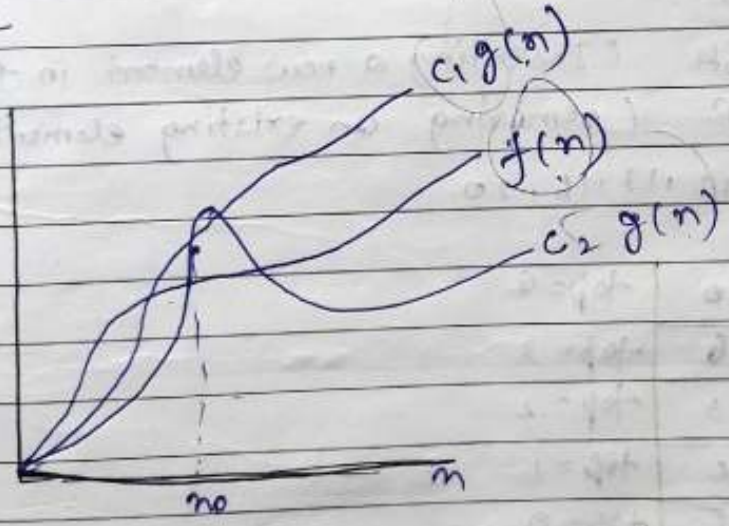
$$f(n) \geq g(n) \quad \forall \quad n_0 \geq 3$$

(3) Theta Notation (θ) (~~Extra~~ Exact bound) ⇒
This notation bound to a function within a constant factor.

$$f(n) = \theta(g(n))$$
$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall \quad n > n_0$$

c_1 & c_2 → constants

Graphically



Q. $f(n) = 5n^2 + 10$
$g(n) = n^2$

$n=$	1	2	3	4	5
$f(n)=$	15	30	55	90	135
$g(n)=$	1	4	9	16	25

$$g(n) \leq f(n) \leq 6\, g(n) \quad \forall \quad n \geq 4$$
$$c_1 = 1 \qquad c_2 = 6$$

Unit - 1 Stack.

Stack is a Linear data structure in which items are added or removed only at one end i.e. known as top of the stack:
Means the last item to be added to the stack, is the first item to be removed. It is also called LIFO list (Last In First Out).
whenever a stack is created stack base is fixed. when the new item is added to the stack from the top, the top goes on increasing conversely when the item is removed from the top the stack top is decrementing.
There are two basic operations associated with stack:-

1) PUSH (Inserting a new element in the stack)
2) POP (removing an existing element from the stack)

for eg 5, 12, 13, 16, 20

Insertion →

20	top=4
16	top=3
13	top=2
12	top=1
5	top=0

Initial top = -1 (stack is empty)

Top = size - 1

Stack Underflow ⇒
It is a situation when the stack contains no element at this point top of the stack is present at the bottom of the stack.

Date
Page No.

Stack Overflow ⇒

This is the situation when the stack become full and no more element can be pushed to the stack. At this this point top of the stack is present at the highest position.

Static implementation of Stack. (Using array)
Dynamic implementation of stack (Using linked list).

Static implementation of Stack ⇒

Push ⇒

```
Push( )
{
 int item, top; s[5];
 if (top == SIZE - 1)
 {
   printf ("Stack is full");
   }
 else
 {
   printf ("Inser element");
  Scanf ("%d", &item);
   top = top + 1;
     s[top] = item;
   }
 }
```

Pop =>

```
Pop ( )
{
int itemp, top, s[5];
if (top == -1)
  {
  printf ("Stack is empty");
  }
else
{
 top = top-1;
}
}
```

Display =>

```
{
Display ( )
{
int i;
for (i= top ; i> -1 ; i--)
  {
    printf ("%d", s[i])
    top = top-1;
  }

void main( )
{
int item, top, s[5], i
```

Date / /
Page No.
Shivalal

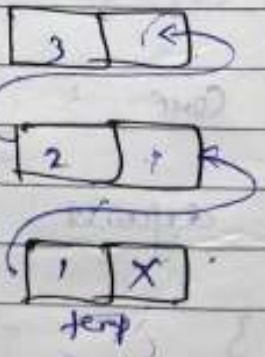
```
# include <stdio.h>
# include <conio.h>
# define SIZE 5
int S[SIZE], top=-1, i, item;
void main ( )
{ int a; char ch;
do {
printf ("press 1 for inserting a New element");
printf ("press 2 for deleting an element");
printf ("press 3 for display of list");
printf ("Enter your choice:");
scanf ("%d", &a);
switch (a)
{
   Case 1 : push ();
            break;
   Case 2 : pop ();
            break;
   Case 3 : Display ();
            break;
   default : printf ("incorrect input");
             break;
}
   printf ("wants to continue");
   Scanf ("%c", &ch);
} while (ch=='y' || ch=='Y');
}
getch ();
}
```

Dynamic implementation of Stack ⇒

Push →

```
/* for Main body.
Struct node
{
 int info;
struct node *next;
};
```

```
Push (.)
{
struct node *temp = (struct node *) malloc (sizeof (struct node));
printf ("Enter element");
Scanf ("%d", &temp → info);
  temp → next = top;
    top = temp;
}
```



pop →

```
pop ()
{
Struct node * temp;
if (top == NULL)
    printf ("List empty");
else.
  temp = top;
  top = top → next;
  free (temp);
```

display =>

```
display ( )
{
struct node * temp = top.
while (temp -> next != NULL)
 {
    printf ("%d", temp -> info)
        temp = temp -> next
 }
 printf ("%d", temp -> info);
}
```

Application =>

1. Used to reverse the data.
2. Evaluating Arithmatic expressions.
3. for expression conversion like infix to postfix OR infix to prefix (Polish notations)
4. Parsing
5. Simulation of Recursion
6. function calling
7. Hardware Support for page replacement in Operating System.

Expression Conversion =>

* priority of operators

↑ Highest (↑ or ^) exponent
Next highest *, /
Lowest +, -

Polish Notations

Infix	Prefix	Postfix
A+B	+AB	AB+
(A+B)/D	+AB/D	AB+/D
	/+ABD	~~AB+D/~~ AB+D/
A+B/D	·A+/BD	A+BD/
	+A/BD	ABD/+

Convert the given element into postfix

Q. A+((B+C)+(D+E)*F)/G

⇒ A+(BC+ + DE+ · F*)/G

= A + (BC+ DE+ F*+)/G

= A + BC+ DE+ F*+G/

= ABC+ DE+ F*+G/+ Postfix

Prefix

A + (+BC + ·+DE * F)/G

~~A + (+ + BC + DE * F~~

A + (+BC + ·* + DEF)/G

= A + (+ + BC * + DEF)/G

= A + /++BC*+DEFG

= +A/++BC*+DEFG

Date / / Page No. Shivalal

Q. (A + B↑D) / (F − F) + G

postfix

(A + BD↑) / EF − + G
ABD↑+ / EF − G +
= ABD↑+EF−/G+

prefix

(A + ↑BD) / − EF + G
= + A↑BD / + − EFG

+ / + A↑BD − EFG

Q. Algorithm for transforming infix expression to postfix expression.
Suppose Q is an arithmetic expression written in infix notation, find the equivalent postfix expression P.

Step 1: Push left parenthesis "(" into the stack and add ")" to the end of Q.
Step 2: Scan Q from left to right and repeat step 3 to 6 for each element of 'Q' until the stack is empty.
Step 3: If an operand is encountered add it to P.
Step 4: If a left parenthesis "(" is encountered push it into the stack.
Step 5: If an operator is encountered then
a) repeatedly pop from stack and add to P each operator which has the same priority or higher priority than the operator.
b) Add operator to the stack.
Step 6: If a right parenthesis is encountered then

a) repeatedly pop from stack and add to P until a "(" is encountered.

b) remove the left parenthesis ("do not add to P) ~~then~~ end of step 2.

Step-7 → end.

Let Q : (A + ((B + C) + (D + E) * F) / G)

Symbol Scanned	Stack	P
(	(	
A	(	A
+	(+	A
(	(+ (	
(	(+ ((	
B		AB
+	(+ ((+	AB
C	(+ ((+	ABC
)	(+ (	ABC +
+	(+ (+	
(	(+ (+ (	ABC + D
D	(+ (+ (+	ABC + D
+	(+ (+ (+	ABC + D
E	(+ (+ (+	ABC + D E
)	(+ (+	ABC + D E +
*	(+ (+ *	ABC + D E + F
)	(+	ABC + D E + F * +
/	(+ /	ABC + D E + F * +
G	(+ /	ABC + D E + F * + G

15/09/16

Date / /
Page No.
Shivalal

Q. Convert the given infix expression into Postfix expression

(i) (A * (B+D)/E - F * (G+H/K))

(ii) ((A+B) * D) ↑ (E-F)

(iii) 2 ↑ 3+5 * 2 ↑ 2 - 12/6

(i)

Symbol scanned	Stack	Postfix expression
A	(	A
*	(*	A
(	(*(	
B	(*(	AB
+	(*(+	AB
D	(*(+	ABD
)	(*	ABD+
/	(*/	ABD+
E	(*/	ABD+E
-	(*/-	
F		ABD+E/F
*	(-	ABD+E/F*
/	(-/	ABD+E/F*
-	(-	ABD+E/F*
*	(-*	ABD+E/F*
(	(-*(	ABD+E/F*
+	(-*(+	ABD+E/F*GH
/	(-*(+/	ABD+E/F*GHK
)	(-*	ABD+E/F*GHK/+
)		ABD+E/F*GHK/+*-

(i) (((A+B) * D) ↑ (E - F))

Symbol Scanned	Stack	Postfix expression
(((	(((	
A	(((	A
+	(((+	"
B	"	AB
)	((	AB +
*	((*	AB + D
)	(	AB + D *
↑	(↑	"
(	(↑ (	AB + D * E
-	(↑ (-	"
F		AB + D * E F
)	(↑	AB + D * E F -
)		AB + D * E F - ↑

(ii) (2 ↑ 3 + 5 * 2 ↑ 2 - 12/6)

Symbol Scanned	Stack.	Postfix expression.
(	(	
2		2
↑	(↑	
3		2 3
+	(+	2 3 ↑ 5
*	(+ *	2 3 ↑ 5
2	(+ * ↑	2 3 ↑ 5 2
		2 3 ↑ 5 2 2
-	(-	2 3 ↑ 5 2 2 ↑ * +
12		2 3 ↑ 5 2 2 ↑ * + 12
	(* - /	2 3 ↑ 5 2 2 ↑ * + 12 *
)		2 3 ↑ 5 2 2 ↑ * + 12 6 / -

Algorithm for evaluating postfix expression :-

This algorithm finds the value of arithmetic expression written in postfix notation P.

Step 1:- Add the ")" at the end of P.

Step 2 → Scan P from left to right and repeat step 3 & 4 for each element of P until the ")" is encountered

Step 3 → if an operand is encountered put it on stack.

Step 4 → if an operator is encountered remove the top-2 elements of stack where A is the top element and B is the next top, evaluate B operator A and place the result back to the stack (end of Step-2)

Step 5 → set the value equal to the top element on stack

Step 6 → exit.

Q. (2 ↑ 3 + 5 * 2 ↑ 2 - 12/6)

P: 2, 3, ↑, 5, 2, 2, ↑, *, +, 12, 6, /, −,)

Symbol Scanned	Stack	value.
2	2	
3	2, 3	
↑		A B 3, 2 B↑A = 2↑3 = 8
5	8, 5	
2	8, 5, 2	
2.	8, 5, 2, 2.	
↑	8, 5, 4	$2^2 = 4$.
*	8, 20	5*4 = 20
+	28	8+20 = 28.
12.	28, 12.	
6	28, 12, 6	
/	28, 2	12/6 = 3.
−	26	

Date / /
Page No.
Shivalik

Q. Evaluate the given Arithmetic expression

5, 6, 2, +, *, 12, 4, /, -)

Symbol scanned	Stack	value
5	5	
6	5, 6	
2	5, 6, 2	
+	5, 8	2+6=8
*	40	8*5=40
12	40, 12	
4	40, 12, 4	
/	40, 3	12/4=3
-	37	40-3=37

& find out its infix expression.

= 5 * (6+2) - 12/4 Ans

Symbol

NOTE: $ → treated as powers

Q. Consider the given parenthesis free expression & find out its value.

6+2,↑,3,↑,2,-,4,*,5

⇒ 6+2↑3↑2-4*5

Symbol Scanned	Stack	Postfix
6	(	6
6		6
+	(+	6,2,
↑	(+↑	6,2,3
3	(+↑	6,2,3,↑,2↑
↑		
2		
-	(-*	= 6,2,3,↑,*,2,↑,4,5

Date / /
Page No.
Shivalal

6, 2, 3, ↑, *, 2, ↑, 4, 5, *, −

Symbol Scanned	Stack	value
6	6	
2	6, 2	
3	6, 2, 3	
↑	6, 8	$2^3 = 8$
*	14	$6 + 8 = 14$
2	14, 2	
↑	196	$14^2 = 196$
4	196, 4, 5	
5	196, 20	$4 * 5 = 20$
−	176	$196 - 20$

Ans

2	6, 8, 2	
↑	6, 64	$8^2 = 64$
+	70, 4, 5	$6 + 64 = 70$
*	70, 20	$4 * 5 = 20$
−	$70 - 20 = 50$ Ans	

Tutorial - 4

15/09/2016

Qus. (a):
Give the Postfix Notation for the following.

(1) NOT A OR NOT B AND NOT C.
(2) a && b || c || ! (e > f)
(3) (A − B) * X + Y / (F − (C * E)) + D

Ans (1) (!A || !B && !C.)

priority
NOT ↑
AND
OR

Symbol Scanned	Stack	value
(	(	
!	(!	A
||	(||	A ! B
&&	(||&&	A ! B ! C ! && ||
)	= A NOT B NOT C NOT AND OR. Ans	

(ii) (a && b || c || ! (e > f))

Symbol Scanned	Stack	Postfix expression
(	(	
&&	(&&	a b
||	(||	ab && c.
||	(||	ab && c || e f >
!	(|| !	
(	(|| ! (	
>	(|| ! (>	
)	(|| !	

= a, b, &&, c, ||, e, f, >, !, || Ans

(iii) ((A - B) * X + Y / (F - (* E) + D))

Symbol Scanned	Stack	Postfix Notation
(	(	
(	((	A B
-	((-	
)	(	AB - X
*	(*	
+	(+/(-	AB - X * Y F
C	(+/(-	AB - X * Y F C
*	(+/(-*	AB - X * Y F C * E
)	(+/	AB - X * Y F C * E * -
+	(+	AB - X * Y F C * E * - / +
)		

Ans

Ques. Evaluate the given expression. Also transform the expression into postfix
12 / (7 - 3) + 2 * (1 + 5) ⇒ expression into postfix

Symbol Scanned	Stack	Postfix
(	(	
12	(	12
/	(/	
(	(/(	12,7,3
)	(/	12,7,3,-
+	(+	12,7,3,-,/
2	(+	12,7,3,-,/,2
*	(+*	"
(	(+*(	"
1	"	12,7,3,-,/,2,1,5
+	(+*(+	
)	(+*	12,7,3,-,/,2,1,5,+
)		12,7,3,-,/,2,1,5,+*+

Step-2 Now calculate the value of postfix expression

Symbol Scanned	Stack	value
12	12	
7	12, 7	
3	12, 7, 3	
-	12, 4	7-3 = 4
/	~~12,4,1,2,1,5~~ 3,2,1,5	12/4 = 3
+	~~12,4,1,2,5~~ 3,2,1,6	1+5 = 6
*	~~12,4,1,12~~ 3,12	2*6 = 12
+	3+12 = 15 Ans	

16/09/2016

Date / /

Page No.

Shivalal

Algorithm for Converting infix to Prefix ⇒

Step-1 Reverse the input string.

Step-2- Scan string from left to right

Step-3- if it is an operand add it to the output string.

Step-4- if it is ")" Push it on stack.

Step-5- if it is an operator then

a) if stack is empty push operation on stack.

b) if top of the stack is ")" push operator on stack.

c) if it has same or higher priority than the top of the stack push operator on stack.

d) else pop the operator from the stack and add it to the output string.

Step-6- If it is "(", pop operator from stack and add them to the output string untill the ")" is encountered.

Step-7- if there is no more input unstack the remaining operators and add them otherwise go to step 2.

Step-8 → Reverse the output string.

Step-9 → exit.

Date / /
Page No.
Shivalai

Ques. Convert the given expression into prefix using Stack.

(A-B) * X+Y / (F-(C*E)+D

Reverse → D+)E*C-F(/Y+X*)B-A(

Scanned Symbol	Stack.	o/p String
D		D
+	+	
)	+)	
E	+)	DE
*	+)*	
C	+)*	DEC
-	+)-	DEC*
F	+)-	DEC*F
(	+	DEC*F-
/	+/	DEC*F-Y
+	++	DEC*F-Y/
X	++	DEC*F-Y/X
*	++*	
)	++*)	
B	++*)-	DEC*F-Y/XB
A	++*)-	DEC*F-Y/XBA
(		DEC*F-Y/XBA-*++

↓ Reverse.

++*-AB X/Y-F*CE D Ans

Date / /
Page No.
Shivalal

Ex. Convert the given expression into prefix

(A * (B+D) / F - F * (G + H / K))

)) K / H + G (* F - E /) D + B (* A (.

Symbol Scanned.	Stack.	of String
))	))	
K.	))	KH.
/	)) /	
+	)) +	KH/*G
(.	)	KH/*G+F
*	)*.	
-	)-/)+	KH/*G+F*EDB
(	)-/.	KH/*G+F*EDB+A
*.	)-/*.	
(		KH/*G+F*EDB+A*/-

↓ Reverse.

-/*A+BDE*F+G*/HK.

Ex. Convert the given infix into prefix

① (A + B ↑ D) / (E - F) + G

② 5 + 3 ↑ 2 - 8 / 4 * 3 + 6

Sol^n - G +) F - E (/) D ↑ B + A (

Symbol scanned	Stack	value
G		G
+	+	G
)	+)-	GFE
(	+/)↑	GFE-DB
+	+/)+	GFE-DB↑A
A		GFE-DB↑A+/+
(	+/	

Date / /
Page No.
Shivalal

⇒ ·+/+A↑BD−EFS

Ans (2) 6+3*4/8−2↑3+5

6	+	6,3
3	+*/	6,3,4,8
−	+−	6,3,4,8,/,*
2	+−↑	6,3,4,8,/,*,2,3
+	+−+	6,3,4,8,/,*,2,3,↑
5		6,3,4,8,/,*,2,3,↑,5

6,3,4,8,/,*,2,3,↑,5,+,3,+

Ans :- +−+5↑3,2,*/,8,4,3,6.

Recursion ⇒
When a function call itself called Recursion.
There are two condition in a Recursion.

(1) Base Case
(2) Recursive Case.

~~for eg :-~~

factorial

```
fact()
{
  if n = 0 ;return(1)
};
else
{
  return (n * fact(n-1))
}
}
```

fibonacci

```
fib()
{
if(n==0||n==1)
return(n);
else
{return (fib(n-2)+fib(n-1))}
}
```

Base Case

Recursive Case.

Date / /
Page No.

Difference b/w iteration & Recursion :-

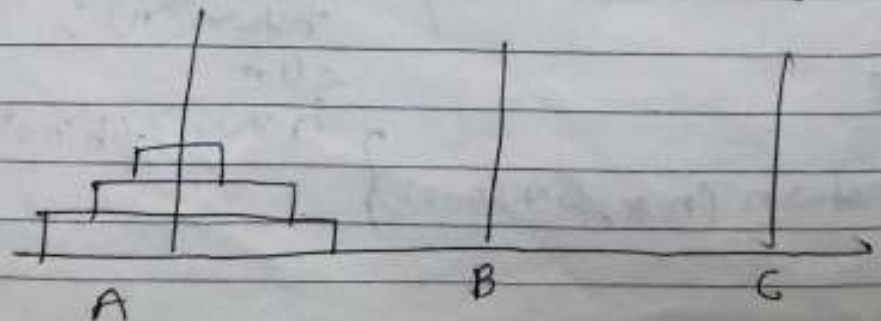
1. Recursion may Scare Smaller & iteration may Scare longer.
2. Recursion use more memory than iteration.
3. Recursion is slower than iteration due to overhead of maintaining stack. whereas iteration doesn't use stack so it is faster than Recursion.
4. Recursion uses selection structure, whereas iteration use repetition structure.
5. iteration terminate when loop Condition fails whereas recursion terminates when a base case is recognised.

⇒ Give example for factorial using recursion and by for loop using iteration also.

Tower of Hanoi ⇒

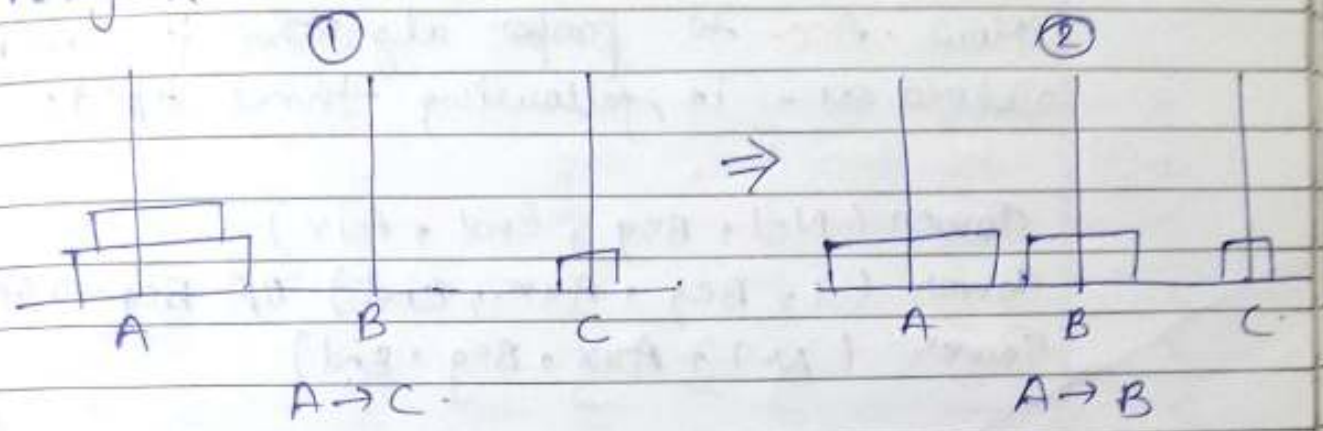
Problem ⇒ Move one disk from page A to page C using page B as an auxiliary only one disk can be moved at a time No larger disk can be placed on a smaller disk.

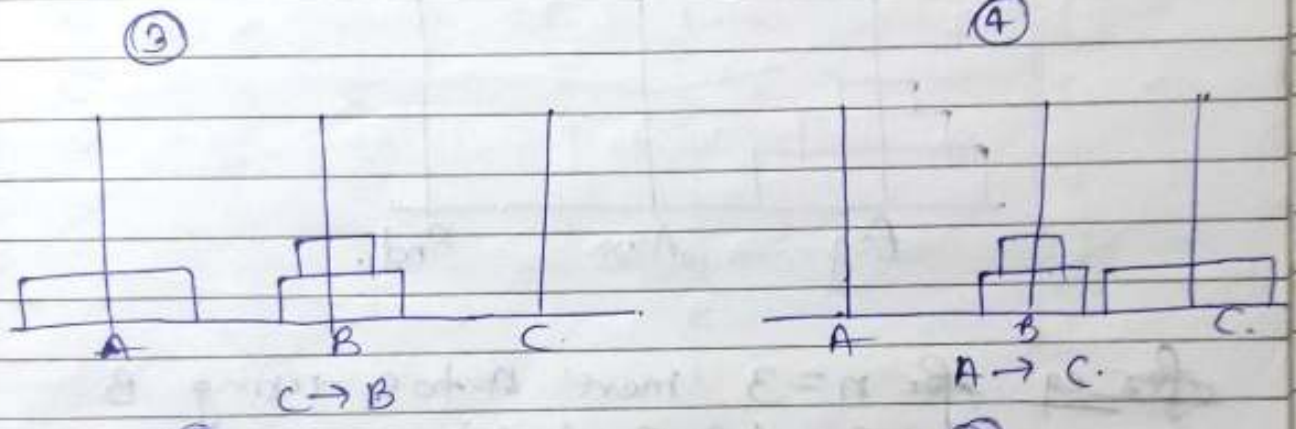
Note - Disk solution requires $2^n - 1$ moves for n-disks.

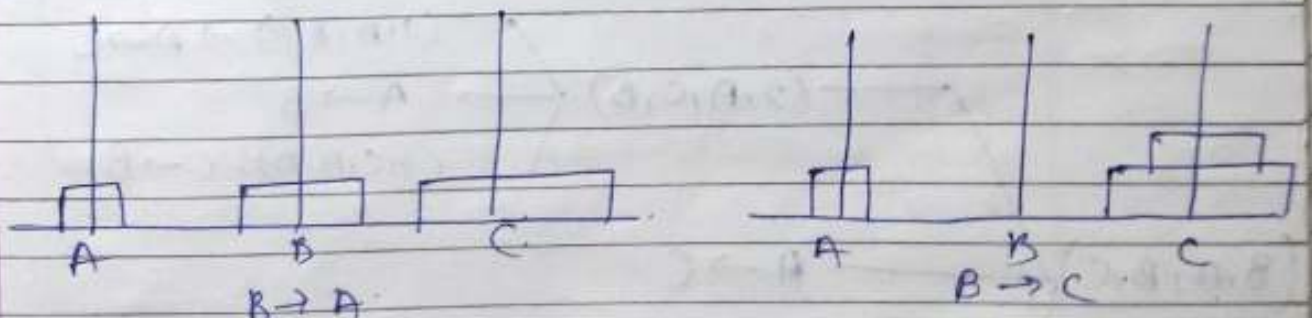


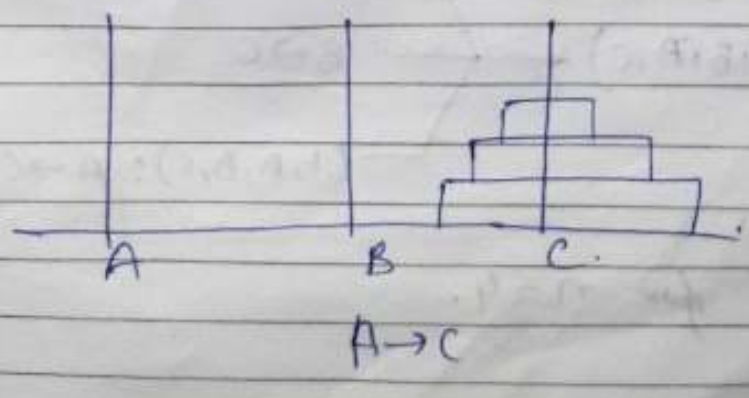
Date / /
Page No.
Shivalal

Verbly it can be solved as.





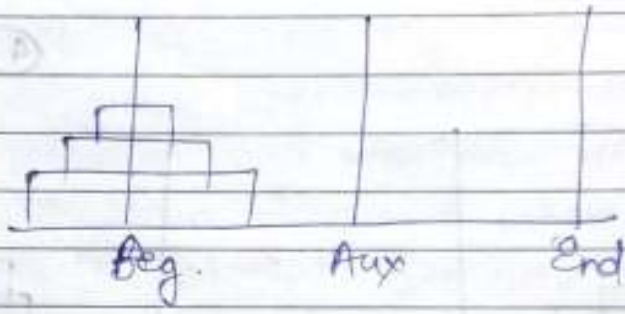




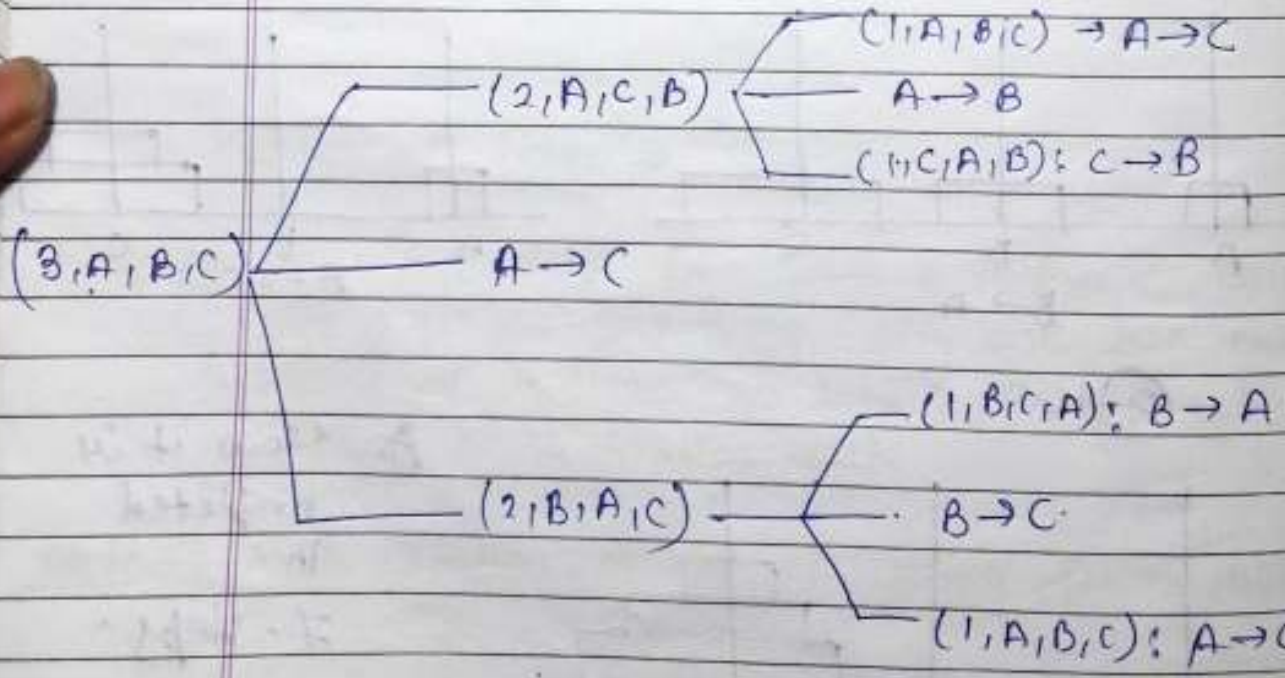
As now it is completed in 7-steps.

Now · Acc. to proper algorithm it can be solved as - in following three steps }:-

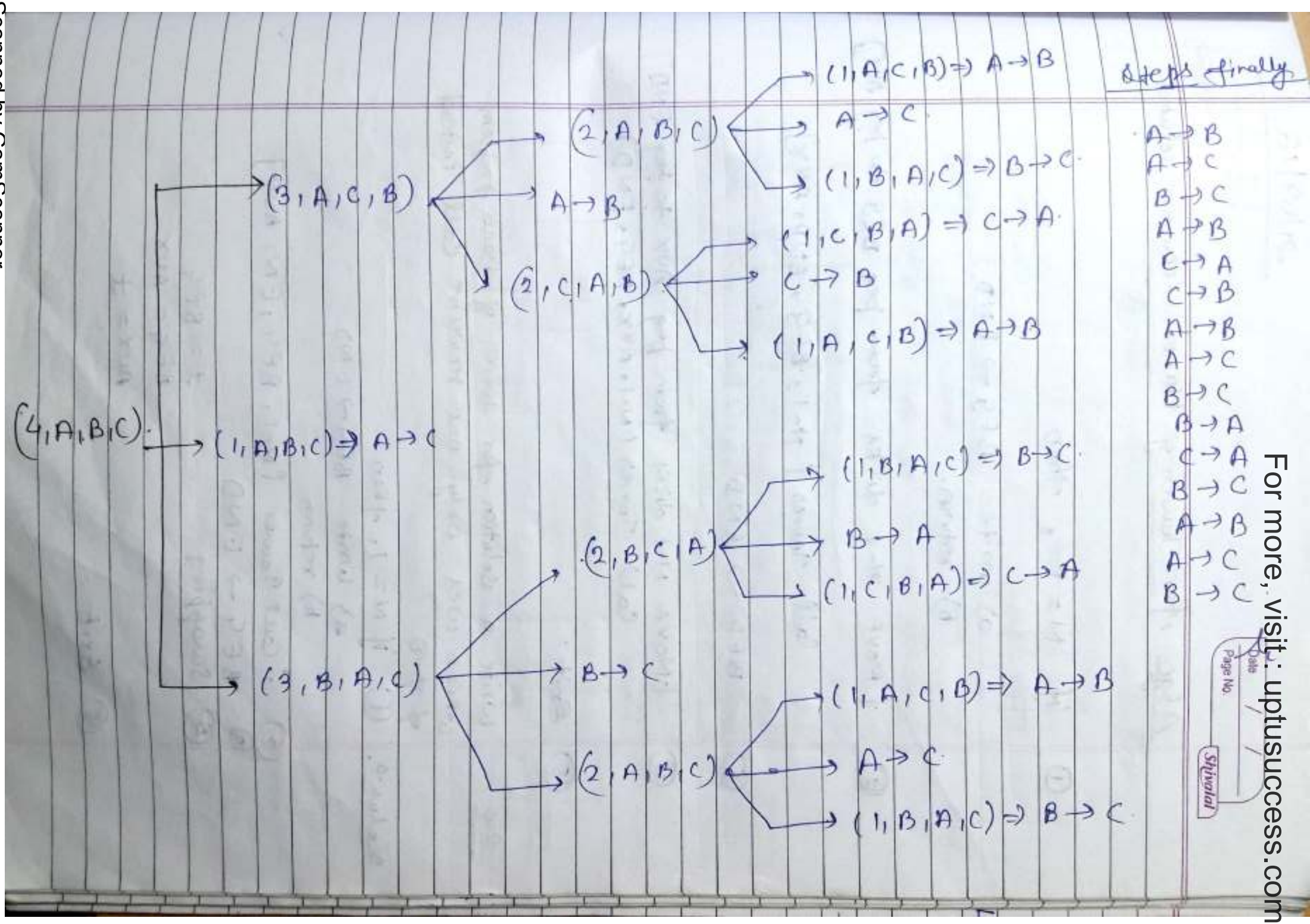
Tower (N-1, Beg, End, Aux)
Tower (1, Beg, Aux, End) OR Beg → End
Tower (N-1, Aux, Beg, End)



~~for eg~~ ~~for~~ n=3 move A to C using B.
i.e. (3, A, B, C)



Ex. Now solve for n=4.



Algo for Tower of Hanoi (Recursive solution)

① If N = 1, then
   a) write BEG → END
   b) return.

② [MOVE N-1 disks from peg BEG to peg AUX]
   Call Tower [N-1, BEG, END, AUX].

③ BEG → END

④ [Move N-1 disks from peg AUX to peg END]
   Call Tower (N-1, AUX, BEG, END)

⑤ Exit.

Que. Write a solution for tower of Hanoi problem which uses only one recursive call instead of two

Solution. ① If N = 1, then
   a) write BEG → END
   b) return

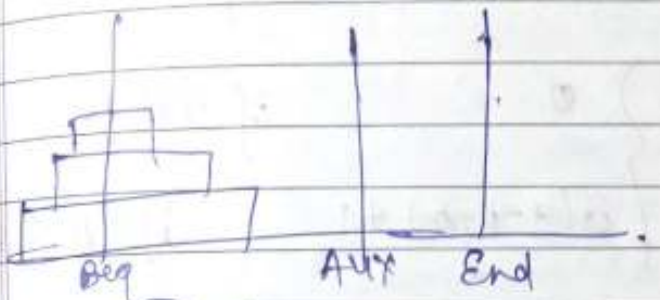
② Call Tower [N-1, BEG, EN, AUX]
③ BEG → END
④ Swapping
   t = BEG
   BEG = AUX
   AUX = t

⑤ Exit.

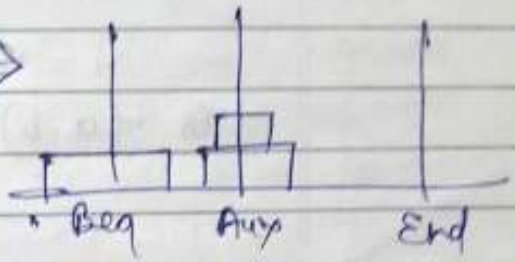
Date / /
Page No.
Shivalal

for eg for n = 3

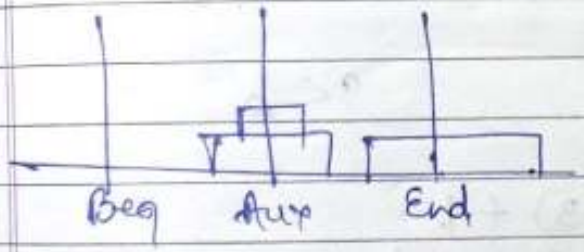
① Step



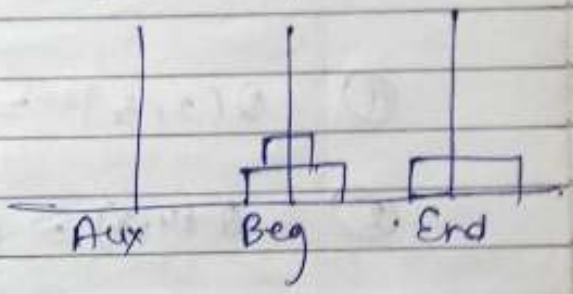
② Step



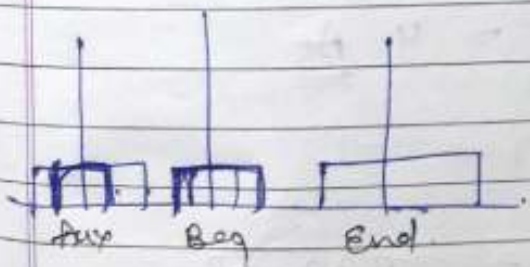
③ Step



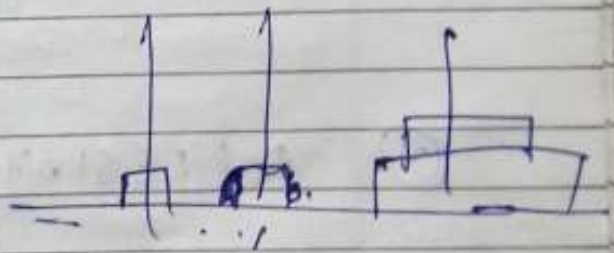
④ Step



again ①



Step - ②



Step - 3

Q6p Let A & B denotes positive integer & Q is a function defined recursively.

$$Q(a,b) = \begin{cases} 0 & \text{if } a < b \\ Q(a-b,b) + 1 & \text{if } b \le a \end{cases}$$

Find the value of (1) $Q(2,3)$ (2) $Q(14,3)$ (3) $Q(5861,7)$

(1) $Q(2,3) = 0$ $\quad a < b$

(2) $Q(14,3) = Q(11,3) + 1$
$= Q(8,3) + 1 + 1$
$= Q(5,3) + 1 + 1 + 1$
$= Q(2,3) + 1 + 1 + 1 + 1$
$= 0 + 4 = 4$. Ans

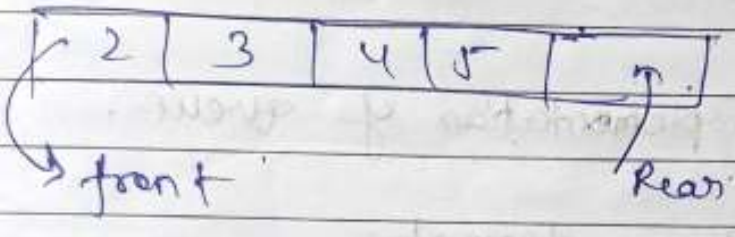
(3) $Q(5861,7)$

$5861 - 1 \cdot 7 = 837 \cdot 2 \cdot 8$
$= 837$ Ans

Date / /
Page No.
Shivalal

Queue Data Structure :-

Queue is a linear data structure in which deletions takes place only at one end called front & insertions can take place only at the other end is called rear. it is also known as FIFO (First In First Out) list.



Notes :-

1. If the front and rear both are -1, it means our queue is empty.
2. When we insert the first Node in a queue both front are rear are incremented.
   front = rear = 0.
3. When we insert a Node in a queue only rear is incremented by one
   rear = rear + 1.
4. when we delete a node from a queue the front is incremented by one.
   front = front + 1.

if it is only a Single Node which we want to delete, both front & rear becomes -1.

Overflow Condition → Rear = Size - 1
Underflow condition → front !! Rear = -1

Operations on queue. =>

1. En-queue( ) (Insert)
2. de(queue)( ) (Delete)
3. display( )
4. full( ) (Overflow)
5. Empty( ) (Underflow)

Static implementation of queue.

To insert an element:

```
# define size 5
int q[size], front=-1, rear=-1
insert()
printf("Enter a node");
scanf("%d", &item);
if (rear == size-1)
  { printf("it is case of overflow"); }
else if (front = -1 || rear = -1)
   { front = rear = 0
     q(rear) = item;
   }
else
    {
      rear = rear+1;
      q(rear) = item;
    }
}
```

Date / /
Page No.
Shivalal

Dynamic ~~implementation~~ of Queue :-

To delete an element

```
delete( )
{
if ( front == -1 || Rear == -1)
  printf ("list is empty");
    }
elseif (front == Rear)
   {
     (front = Rear)
     front = -1 ; Rear = -1   }
else
  {
    front = front + 1;
   }
}
```

Display ⇒

```
Display( )
{
  int i;
 if ( front == -1 || Rear == -1)
     printf ("List is empty")
else
     for (i = front i<= Rear; i++)
        printf ("%d", q[i]);
     }
```

for Main body ⇒

```
#include<stdio.h>
#define SIZE 5 ;
int q[SIZE] ,front=-1, Rear=-1;
void main( )
{
int a; char ch;
do { printf("press 1 to delete an element");
printf(" press 2 to insert an element");
printf(" press 3 to display the list");
printf(" Enter your choice");
Scanf("%d", &a);
Switch (a)
{
Case 1 ; delete( );
        break( );
Case 2; insert( );
        break( );
Case 3 : Display( );
        break;
default : printf("incorrect input");
        break;
}
printf("U want to Continue");
Scanf("%c", &ch);
}while(ch=='y' || ch=='Y');
}
getch( );
}
```

Date / /
Page No.
Shivalal

Dynamic Implementation of Queue =>

Insert =>

```
Insert()
{ Struct node *rear = NULL;
  Struct node * front = NULL;
insert()
{
Struct node * temp = (Struct node *) malloc( Size of(Struct Node))
printf ("Enter node");
Scanf ("%d", & temp → info);
temp → next = NULL
if (front == NULL
{
   front = rear = temp;
}
else
{
   rear → next = temp
   rear = temp
}
```

Delete =>

```
delete( )
{
printf ("Enter node you want to delete");
Scanf ("%d",
Struct Node * temp = front;
if ( front == NULL)
        print("List empty");
```

```
    else
    {
      front = front → next
      free (temp);
    }
}
```

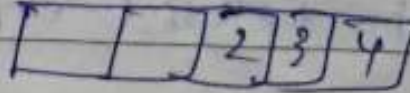
Display

```
Display ( )
struct node *temp = front;
if (front == NULL)
  {
    printf ("List empty");
  }
else
  {
    while (temp != NULL)
    {
    print ("%d", temp → info);
    temp = temp → next;
    }
}
```

Q Draw back of linear queue →

A If a queue is filled upto its last limit and then we delete some element.
After that it will show queue is full although we have space for some elements.

for eg

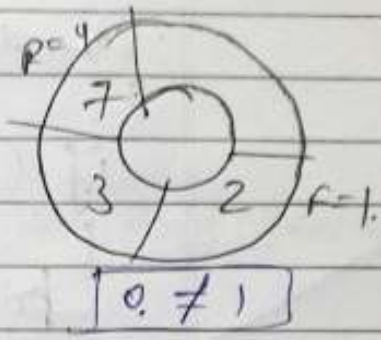


⇒ queue is full although space

Date / /
Page No.
Shivalal

Circular queue ⇒

Condition for Overflow:



$0 \neq 1$

as rear − front = size − 1

rear = size − 1

(rear + 1) % size = 1

(rear + 1) % size = 0

(rear + 1) % size = front → if it satisfies then queue is full.

Static Implementation of Queue ⇒

⇒ Insertion of an element.

```
cq - insert( )
 {
  if ( front == (rear+1) % size )
    {
      printf ( " queue full");
    }
  else if (front == -1)
    { front = rear = 0;
      cq[rear] = item;
    }
  else
   {
     rear = (rear +1) % size;
      cq [rear] = item;
   }
}
```

Deletion of an element

```
Cq - delete()
{
 if (front == -1)
  {
   printf("queue empty");
  }
 else if (front == rear)
  {
   front = rear = -1
  }
 else
 {
  front = (front+1) % size;
 }
}
```

For display of queue =>

```
Cq - display.
{
 if (front == -1)
  {
   printf("List empty");
  }
 else
 for (i = front; i != rear; i = (i+1) % size)
  {
   printf("%d", cq[i]);
  }
 printf("%d", cq[i]);
}
```

Date / /
Page No.
Shivalal

Dynamic implementation of Circular Queue :-

Insertion

```
Cq-insert()
{
Struct node *temp = (Struct node*) malloc (Size of (Struct node*))
printf ("Enter node");
scanf ("%d", &temp->info);
if (front == NULL)
{
    front = rear = temp
    rear->next = front;
}
else
{
    rear->next = temp
    rear = temp
    rear->next = front;
}
}
```

Deletion

```
Cq-delete()
Struct node *temp = front;
}
    if (front == NULL)
    {
        printf ("List empty");
    }
else
{   front = front->next;
    free (temp); rear->next = front;
}
```

Display

```
cq - display ( )
{
 Struct node * temp = front;
if ( front = = NULL)
  {
     printf ("List Empty");
  }
else. while ( temp → next != = front )
   { printf ("%d", temp → info;)
        temp = temp → next ;
   }
   printf ("%d", temp → info);
  }
```

Double ended Queue (deque) ⇒

It is a homogeneous list of elements in which insertion & deletion operations are performed from both the ends but not in the middle. means we can insert an element front end and ~~rear~~ rear end & can delete the element from front end and ~~front~~ rear end.

There are two types of deque :-

1. input Restricted deque.
2. Output Restricted deque.

① **Input Restricted deque :-**
We restrict deque only for insertion and deletion it can be done from both the ends front or Rear.

② **Output Restricted deque :-**
It allows deletion at only one end of list but insertion at both the ends of the list.

Q. Consider a deque of characters which is a circular array having 6 memory cell

—	A	C	D	—	—

Left = 2
Right = 4

Perform the following operation
a) F is added to the right of deque
b) two letters deleted from the right.
c) K, L and M are added to the left of the deque
d) One letter on the left is deleted.
e) R is added to the left
f) S is added to the right
g) T is added to the right.

(a)

—	A	C	D	F	—

Left = 2
Right = 5

(b)

—	A	C	—	—	—

Left = 2, Right = 3

(c)

K	L	M	A	C	D

K	A	C	—	M	L

Left = 5
Right = 3

d⇒ K A C _ _ L

left = 6
Right = 3

e⇒ K A C _ R L

left = 5
Right = 3

(f) K A C S R L.

left = 5
Light = 4

(g) Queue is full. (Condition ⇒ if left = Right+1)

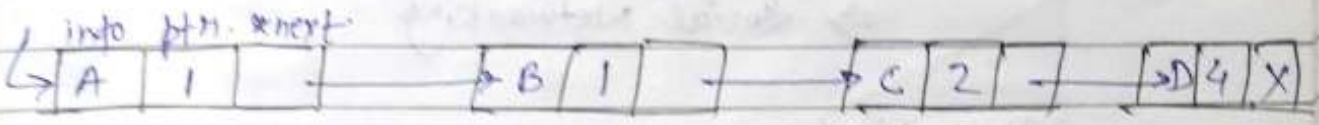
Priority Queue:-
It is a collection of elements such that each element has been assigned a priority and the order in which elements are deleted & processed comes from the following rules.

Rules:- ① an element of higher priority is processed before an element of lower priority.
② Two elements with the same priority are processed acc. to the order in which they were added to the queue.
③ There are two types of implementation of priority queue.

a) Ascending priority queue:-
It is a collection of items into which items can be inserted arbitarily and from which only

smallest item can be removed is called ascending priority queque.

b) Descending priority queue
Only the largest item is deleted the elements of priority queque need not be numbers or characters that can be compared directly.



Application:- Time sharing system.

Applications of Queue:-
1. All type of Customer Services like Reservation System.
2. Printer server routines are designed using queue.
3. Round Robin technique for processor Scheduling is implemented using queue.

Unit-III

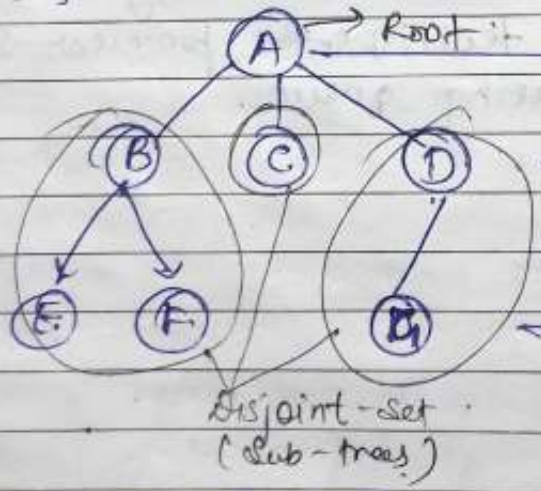
Tree

Tree is a Non-Linear data structure. it is used in ⇒ Network file System
⇒ Routing Algorithms
⇒ Social Networking.

Definition:-
Tree is a finite set of one or more nodes having following properties -

i) there is a node called Root.
ii) The remaining nodes are partitioned into $n > 0$ disjoint set $T_1, T_2 \dots T_n$. ~~xy~~ called Subtrees.

eg - A



~~Depth = 3~~

A → Root of the tree
Here tree contain 7 nodes

→ **Degree of a node** -
Degree of a node is the No. of Subtrees of a Node.

for A → 3 D → 1
B → 2 E, F, G → 0
C → 0

Date / /
Page No.
Shivalal

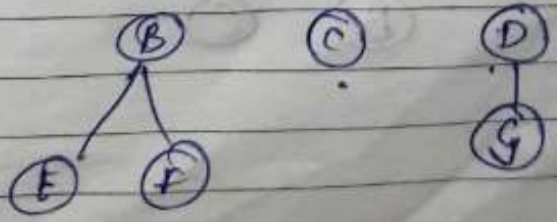
→ **Degree of a tree** –
Degree of a tree is the maximum degree of the node in the tree.
Hence for eq-(A) → Degree of a tree = 3

→ **Leaf Node/terminal node** :-
A node with degree zero is called leaf Node or terminal Node.
for In eq. (A) ⇒ E, F, C, G.

→ **Level** :-
The Root node is always at level 0. Then its immidiate children are at level 1. & their immidiate children are at level 2 and so on.
for eg in (A) ⇒ level 0 → A
level 1 – B, C, D
level – 2. E, F, G.

→ **Depth** :- (height)
Depth is the maximum Level of any node
Depth = Max-level + 1
Depth of tree = 3
Depth of node B = 1
$h = \log_2(n+1) - 1$

→ **forest** :-
It is the set disjoint trees. In a tree if you remove its root node then it become a forest.
for eg

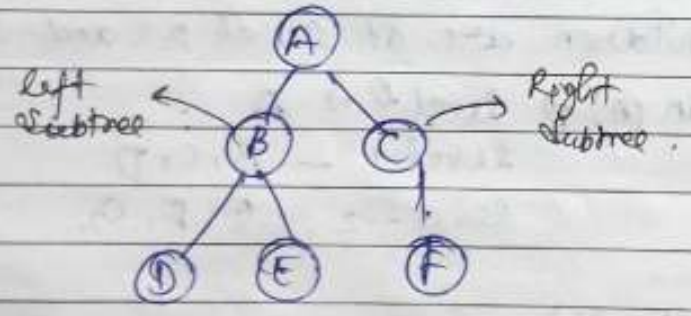


Binary tree :-

A Binary tree T is defined as a finite set of elements called nodes such that.

a) T is empty called Null tree OR empty tree.
b) T contains a node R called the root of the tree, and the remaining nodes of the tree form an ordered pair of disjoint binary trees called T_1 and T_2. that means each node has 0, 1 or 2 children (degree).

T_1 and T_2 are called left and Right Subtrees of R.

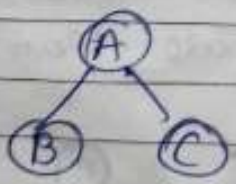


Traversing in Binary Tree :-

Visiting the node of the binary tree.

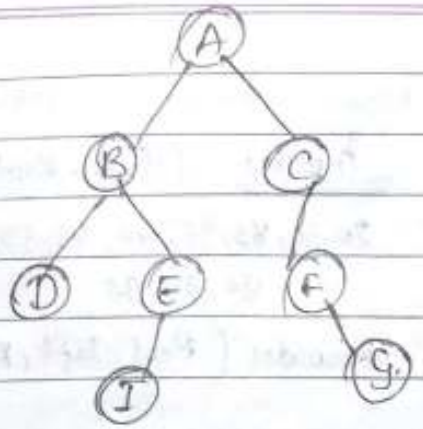
1. Inorder Traversing (left, root right) BAC
2. Preorder Traversing (Root, left, right) ABC
3. Postorder Traversing (left, right, Root) BCA

Eg -



Date
Page No.

~~for eg~~.



Find the all traversing tree ?

① Inorder.

D B I E A F G C.

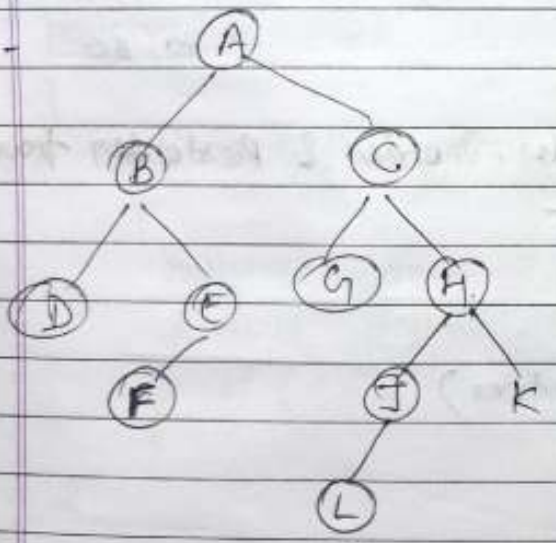
② Preorder.

~~A D E I C F G~~ A B D E I C F G

③ Postorder.

D I E B G F C A.

Qu.



Inorder

D B F E A G C L J H K

Preorder

~~A D E F C G H J L K~~

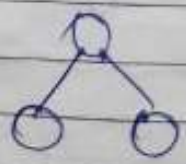
A B D E F C G H J L K

Postorder

D F E B G L J K H C A

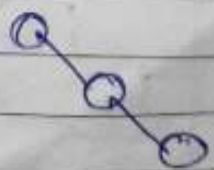
Qu Draw all the possible Non-similar trees with 3-nodes ~~that are~~ (binary tree)

i)

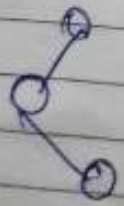


ii)

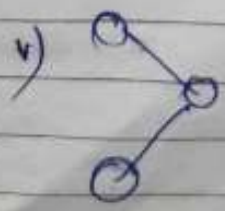
iii)



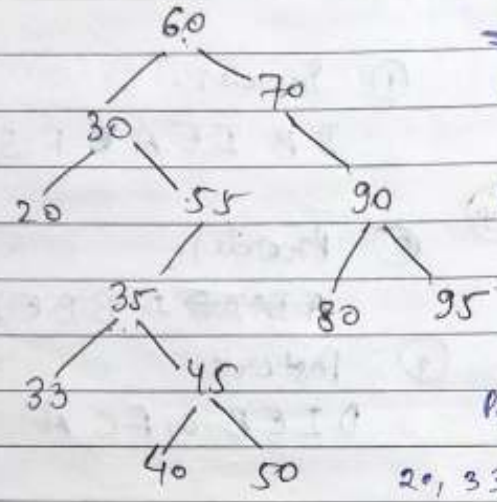
iv)



v)



Q1 Find all traversing



Inorder (Left, Root, right)

20, 30, 33, 35, 40, 45, 50, 55, 60, 70, 80, 90, 95

Preorder (Root, left, Right)

60, 30, 20, 55, 35, 33, 45, 40, 50, 70, 90, 80, 95

Postorder (Left Right, Root)

20, 33, 40, 50, 45, 35, 55, 30, 80, 95, 90, 70, 60

Algorithms for preorder, Inorder & Postorder traversing in a binary tree :-

```
Inorder (struct node *tree)
{
  if (tree != NULL)
  {
   Inorder (tree -> left);    // Visit left subtree
   printf ("%d", tree -> info);
   Inorder (tree -> right);   // Visit right subtree
  }
}
```

```
preorder ( struct node * tree)
{
  if ( tree != NULL)
  {
   printf (" %d", tree -> info);
   preorder (tree -> left);
   preorder (tree -> right);
  }
}

postorder ( struct  node * tree)
{
  if (tree != NULL)
    {
     postorder (tree -> left);
     postorder (tree -> right);
     printf ("%d", tree -> info);
    }
}
```

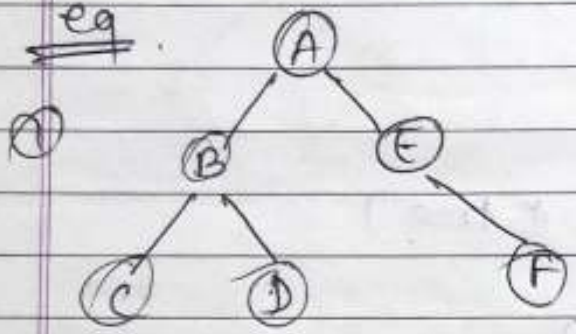
Representation of binary tree :-

There are two types of representation of binary tree

1. Array Representation
2. Linked list

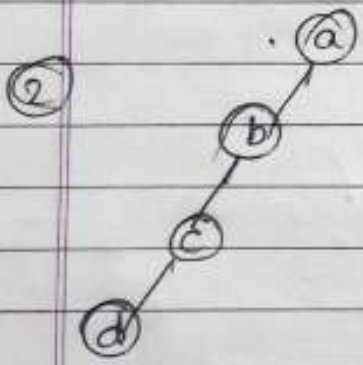
(1) Array Representation.

eg.



A	B	E	C	D		F

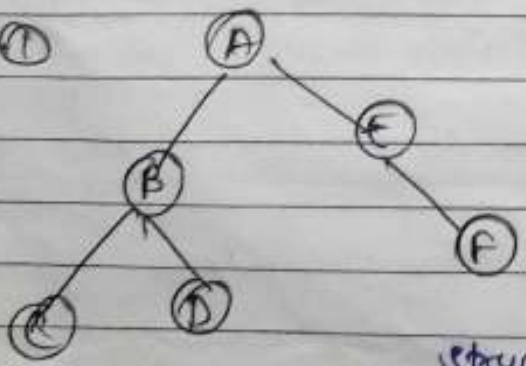
(2)

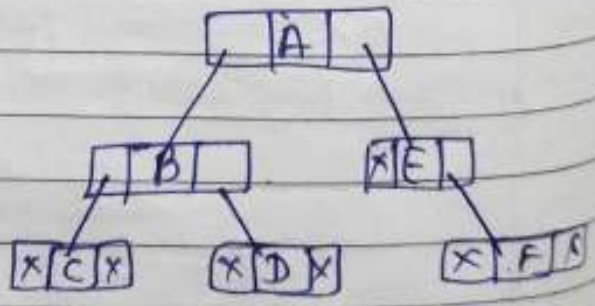


a	b	c				d

(2) Linked Representation :-

eg (1)






```
struct node
{ int info;
  struct node * left;
  struct node * right;
};
```

Algorithm to count no. of nodes in a binary tree :-

```
count (struct node *p)
{
  if (p == NULL)
    return 0;
  else
    return (count (p->left) + count (p->right) + 1)
}
```

OR

```
count (struct node *tree)
{
  if (tree != NULL)
    {
      if (tree->left != NULL
        {
          node++;
          count (tree->left);
        }
      if (tree->right != NULL)
        {
          node++;
          count (tree->right);
        }
    }
}
```

Date / /
Page No.
Shivalal

Algorithm to find the height of binary tree ⇒

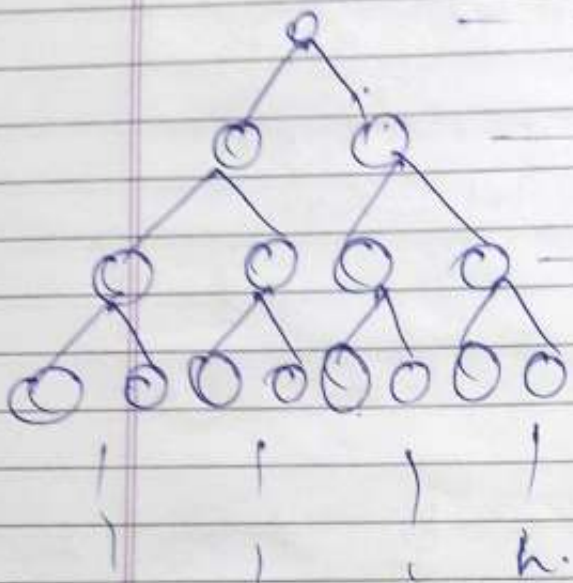
```
height (struct node * p)
{
  if (p == NULL)
  return 0;
  else
    return (1 + max (height(p→left), height(p→right)));
}
```

Algorithm to count the no. of leaf nodes in a tr

```
Count_leaf (struct node * p)
{
  if (p != NULL)
  {
    if (p→left == NULL && p→right == NULL)
      {
        c++;
      }
  else
    count_leaf (p→left);
    count_leaf (p→right);
  }
}
```

Prove that ⇒

Note ⇒ The maximum no. of nodes in complete binary tree of height 'h' is $2^{h+1} - 1$.



Hence.

total Maximum no. of Nodes $= 2^0 + 2^1 + 2^2 + \cdots 2^h$

$= \frac{2(2^n - 1)}{n-1}$

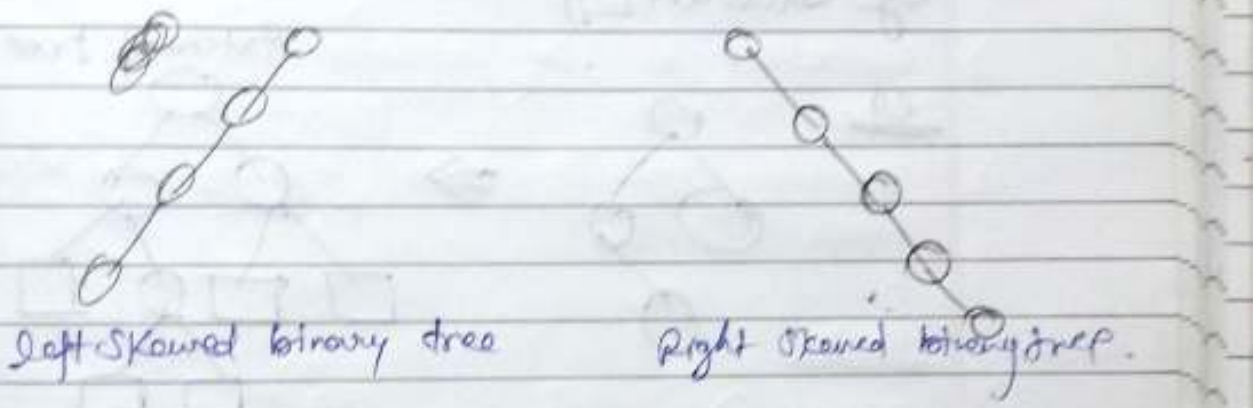
$= \frac{1(2^{n+1} - 1)}{2-1} = \boxed{2^{n+1} - 1}$

Note. 2 ⇒ The minimum no. of nodes in Complete binary tree of height h is 2^{h-1}

& $\boxed{h = \log_2 (n+1) - 1}$

Skewed Binary tree ⇒

A skewed binary tree to be skewed to the left or to the right that means in left skewed binary tree most of the nodes have the left child without corresponding to right child, similarly in a right skewed binary tree most of the nodes have right child not left child.



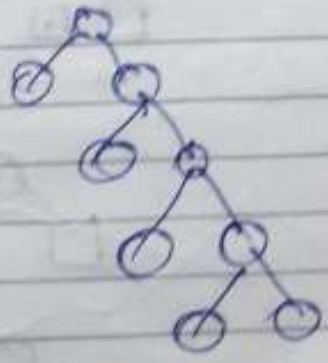
Left skewed binary tree

Right skewed binary tree.

Strictly binary tree ⇒

If every non-terminal node in a binary tree consist non-empty left sub-tree and right sub tree than tree is called strictly binary tree means the nodes have only 0 child or 2 child.

for eg



Extended binary tree ⇒

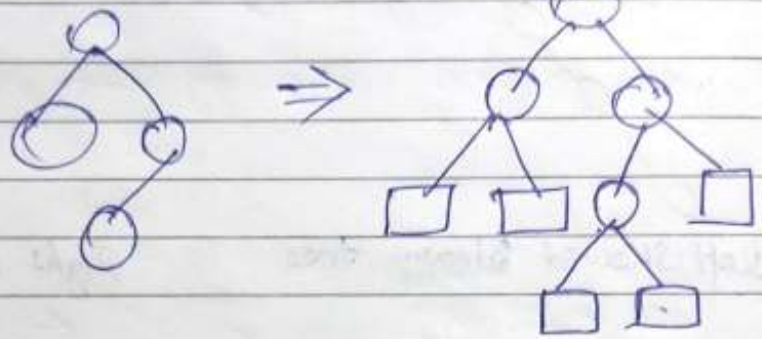
It is also called 2-tree.

The node with two child are called internal node & the node with zero child are called external node.

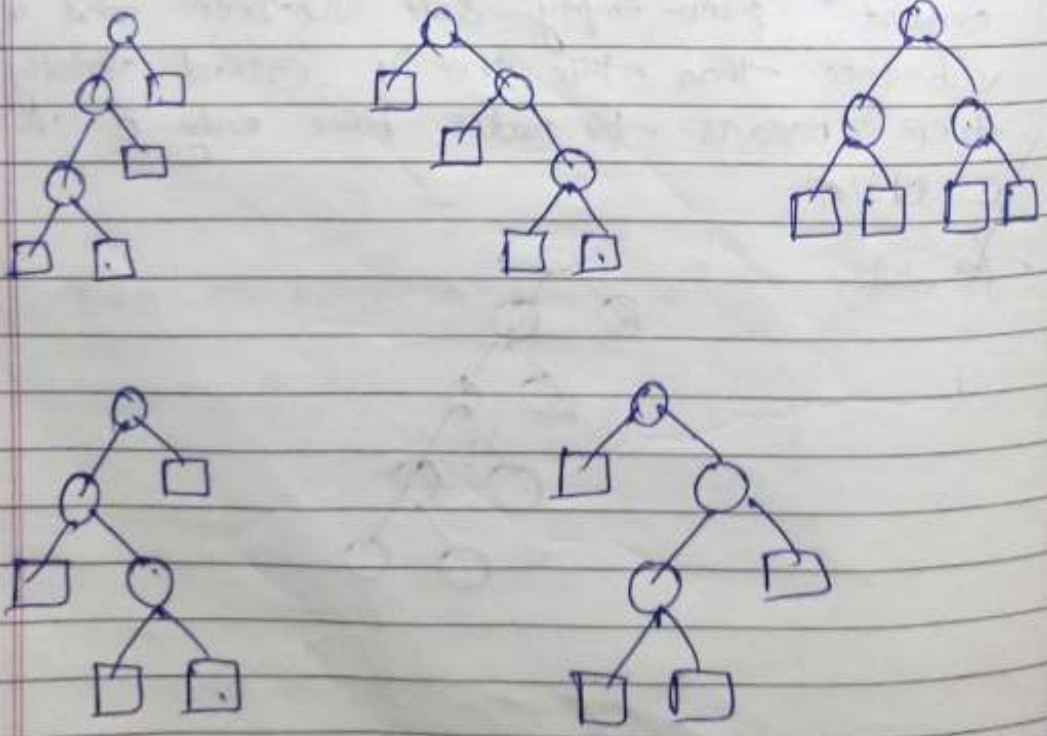
In an extended tree each empty subtree is replaced by external node. it is denoted by square. (□)

Extended tree.

eg.



Draw all possible two-tree of Node-3 .⇒



• Notp⇒ In a binary tree of n nodes having n+1 external nodes

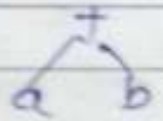
Expression tree →

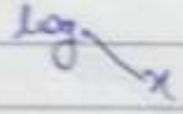
Expression tree is an application of binary tree. If we make binary tree of an expression and traverse the binary tree inorder we will get inorder sequence of the expression.

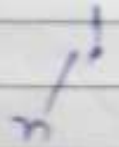
Rules for Constructing expression tree! -

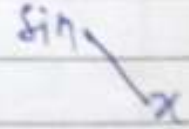
1) Operators will be always the root of the expression tree.
2) Operands will be either leaf or non-terminal node

for eg

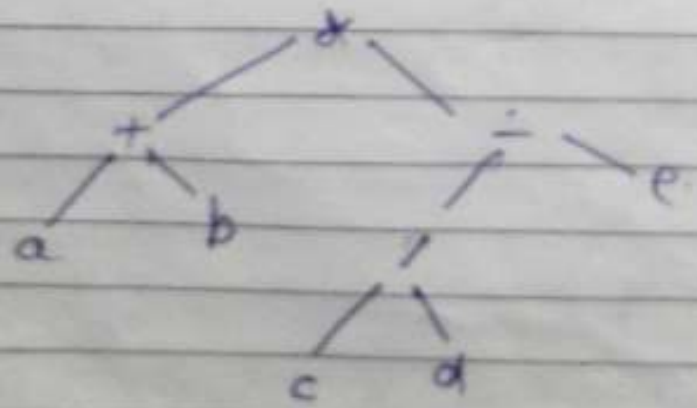
i) $a+b \Rightarrow$ 

ii) $\log x \Rightarrow$ 

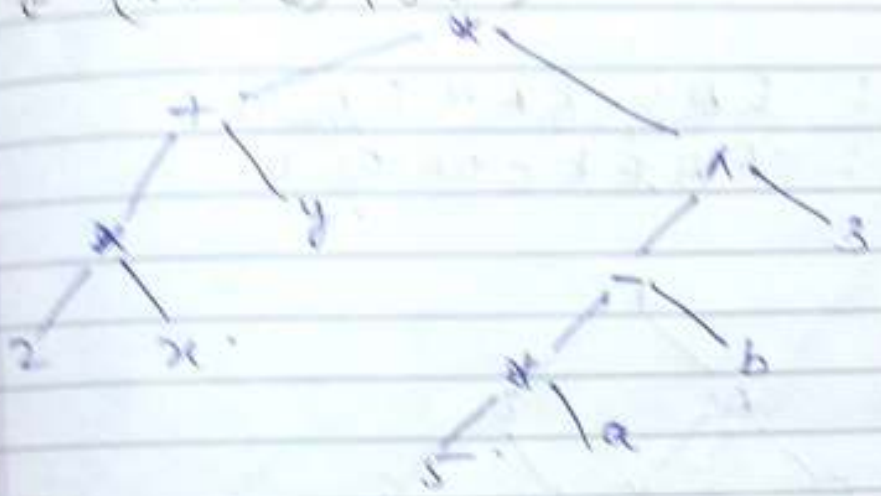
iii) $n! =$ 

iv) $\sin x \Rightarrow$ 

v) $(a+b) * ((c/d) - e)$



Q) $E = (2x + y)(5a - b)^3$



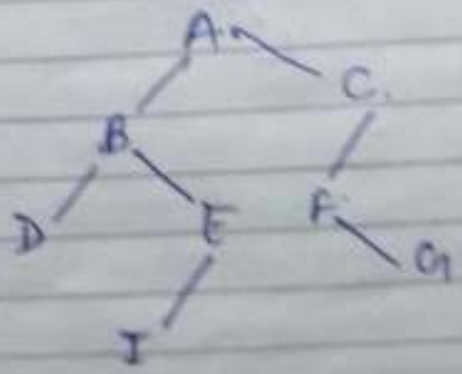
Construction of binary tree from the given expression

To reconstruct a binary tree we will need two order. first preorder and inorder. Second is postorder and inorder.

Inorder → Check Left & Right
Pre/postorder → Check its Root

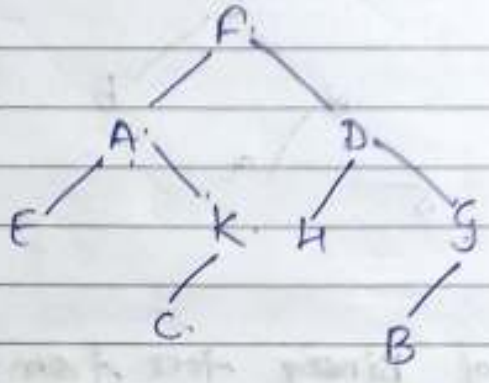
Q) Preorder : A B D E I C F G

Inorder : D B I E A F G C
Left Right

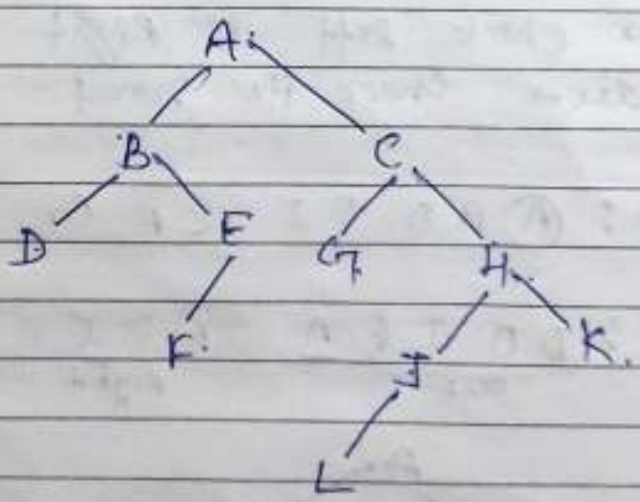


Q. Construct the binary tree that has 9 nodes

Inorder : E A C K F H D B G
Preorder : F A E K C D H G B



Q. Inorder : D B F E A G C L J H K
Postorder : D F E B G L J K H C A

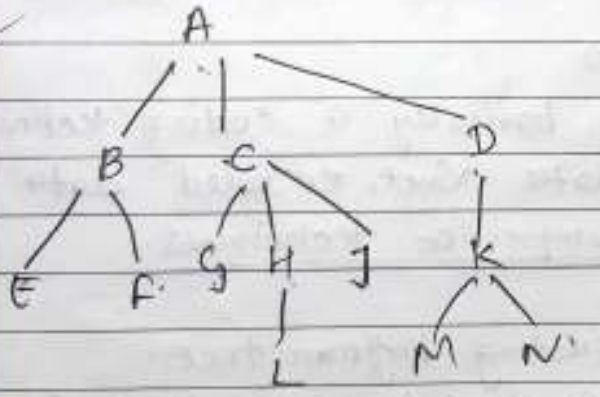


General tree to binary tree :-

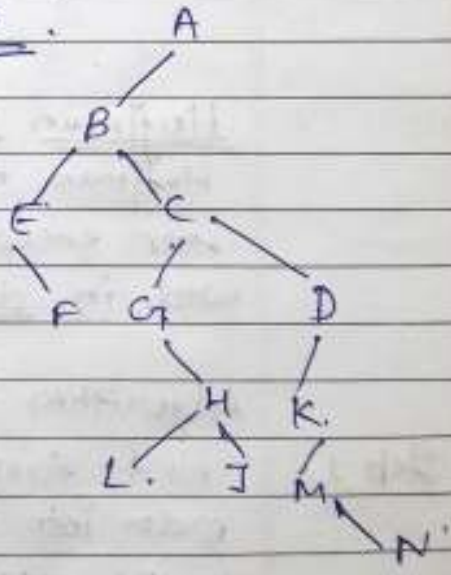
Suppose T is a general tree & T' is binary tree.

1) First of all the nodes of the binary tree (T') will be same as the nodes of the general tree (T) & root of T' will be root of T.
2) Let n be a node in binary tree T' then the left child of n in T will be the first child of the node n in the general tree T & the right child of n in T' will be the next sibling of n in the general tree T.

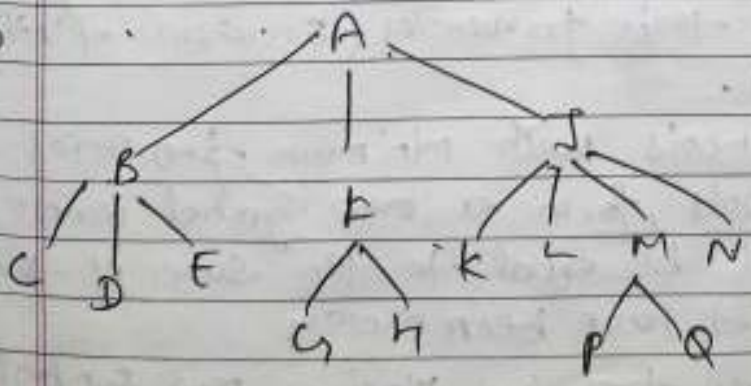
Q. Construct the binary tree of the given tree



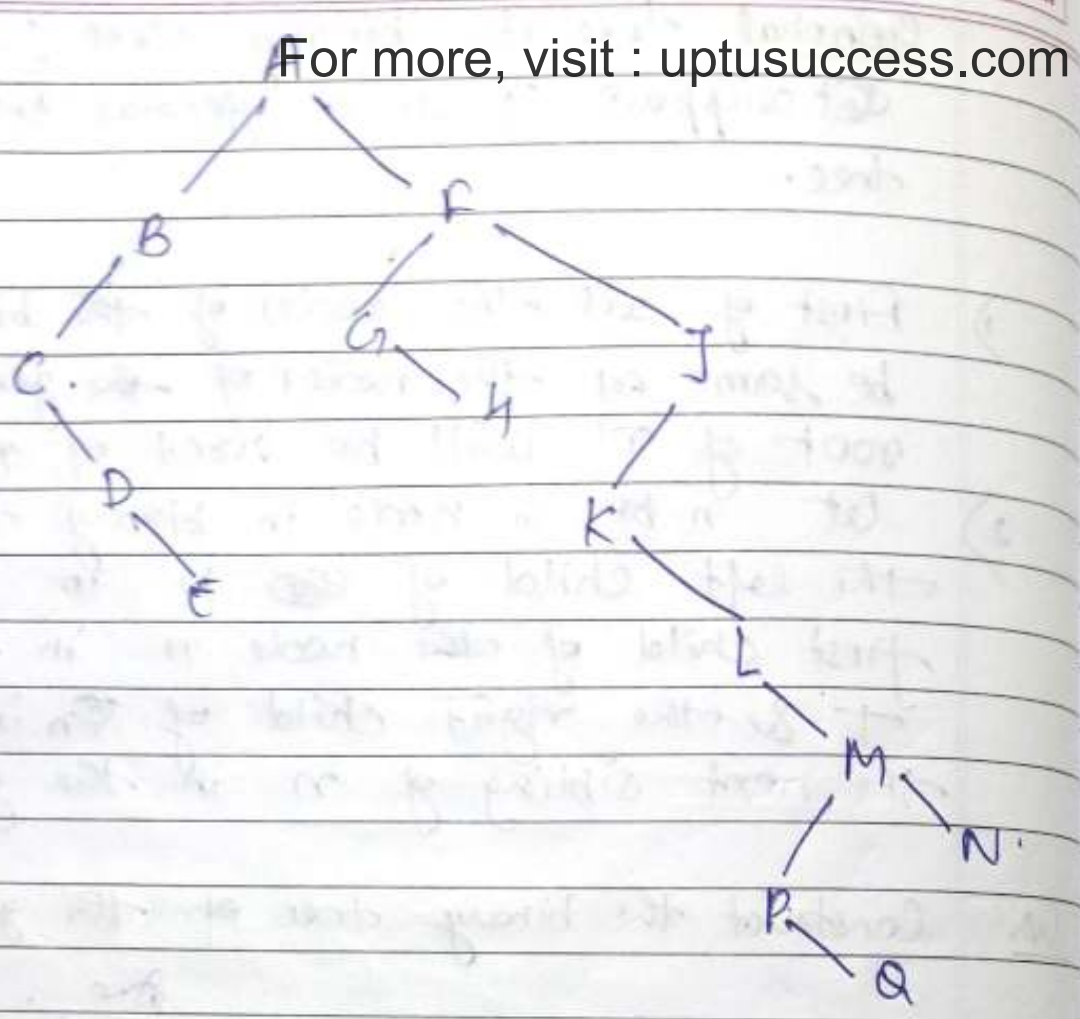
Ans.



Q.



Ans



Huffman tree =>

Huffman tree is basically a coding technique for encoding data Such encoded data is used in data Compression techniques.

Algorithm for creating Huffman tree -

Step 1 - First Arrange the symbols into decreating order of their frequencies (Occurence of letter in the msg.).

Step 2 Pick two symbols with minimum frequencies.

Step 3 Pick the symbols form a new symbol whose frequency will be equal to the sum of the frequencies that have been chosen

Step 4 Create a binary tree by placing new symbol at the root & make the symbol left child which has low frequency & make the symbol right child that has high frequency

Step (5) Assign Zero to the left & One to the right.

Step-(6) Delete the chosen symbols from the list & place the new symbols to the list.

Step-(7) Re-sort the list again into decreasing order.

Step-(8) Repeat the above steps untill all the symbols have been covered.

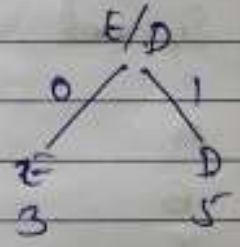
Qe. Create the Huffman tree of the given data & decode the message 10110 1110010

Step-1

Symbols	frequencies
A	50
B	45
C	40
D	5 } Minimum freq.
E	3 } Minimum freq.

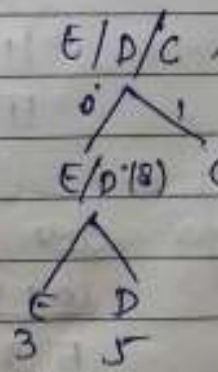
⇒

A	50
B	45
C	40
E/D	8 (5+3)



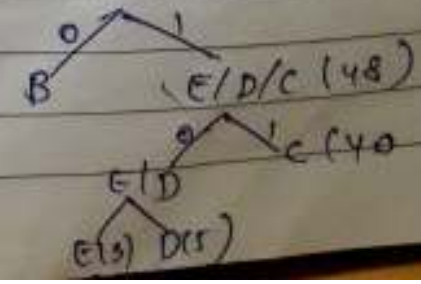
Step-(2)

A	50
B	45
E/D/C.	48.

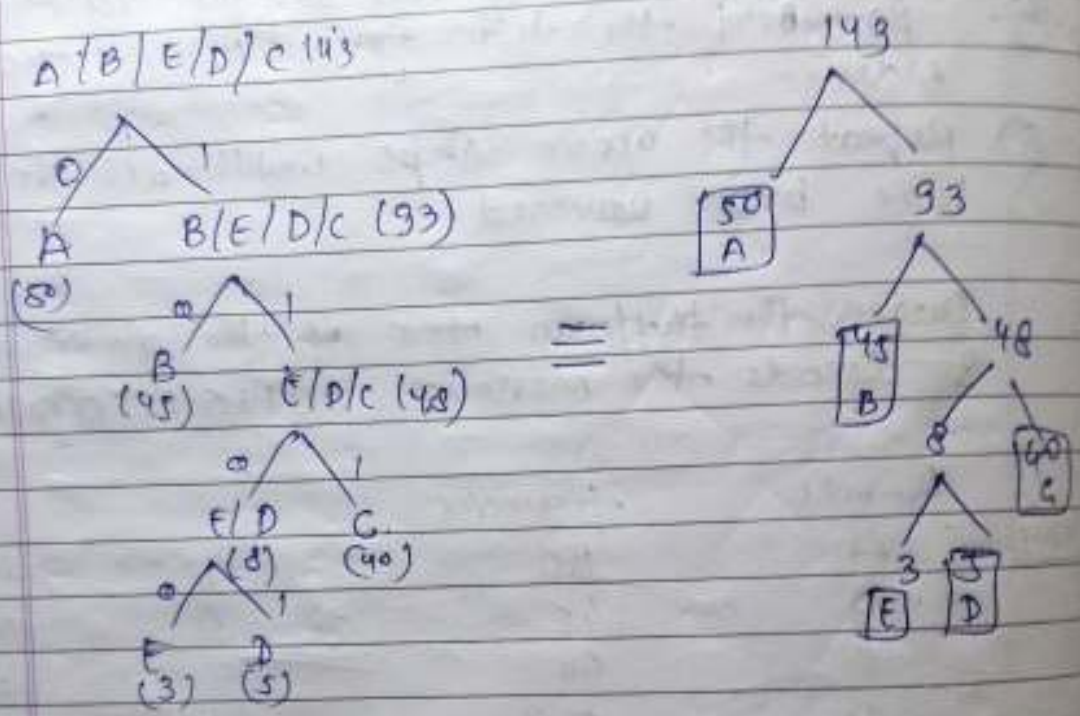


(3)

A	50
B/E/D/C	93



(4) A/B/E/D/C. (143)



Symbol	codes	Code length
A	0	1
B	10	2
C	111	3
D	1101	4
E	1100	4

Read for Code.

10110111 0010

B D F B

Move in tree until leaf node is not occured then read

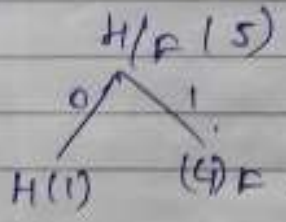
Page No.
Shivalat

Q. Draw a Huffman tree for the following symbols where frequencies are & decode the msg 1110100010111011

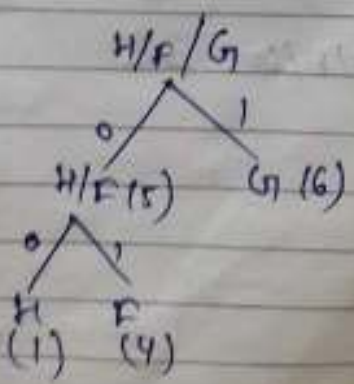
A	15
B	6
C	7
D	12
E	25
F	4
G	6
H	1
I	15

E	25
A	15
I	15
D	12
C	7
B	6
G	6
F	4
H	1

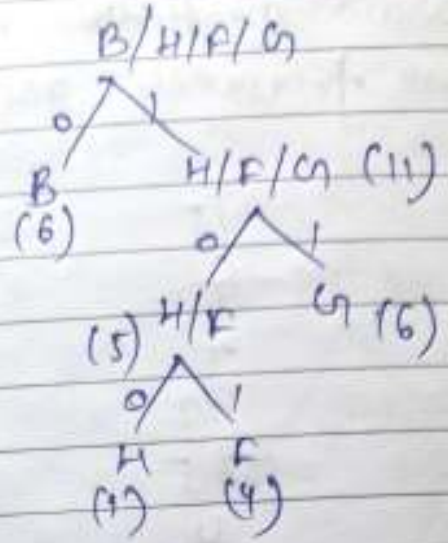
E	25
A	15
I	15
D	12
C	7
B	6
G	6
H/F	5



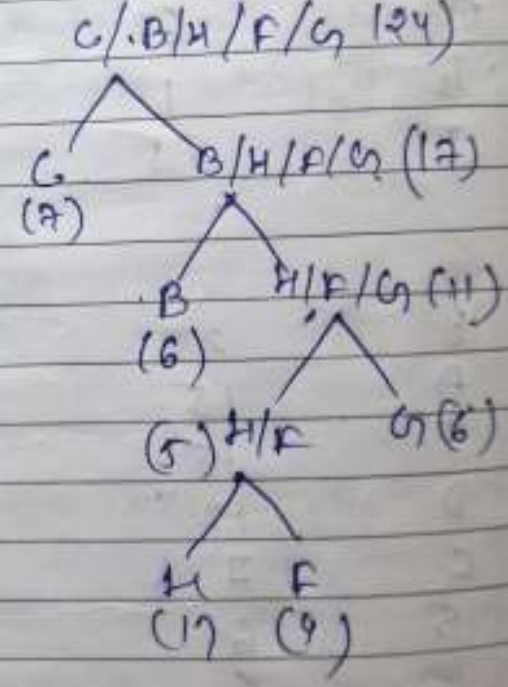
E	25
A	15
I	15
D	12
C	7
B	6
H/F/G	11



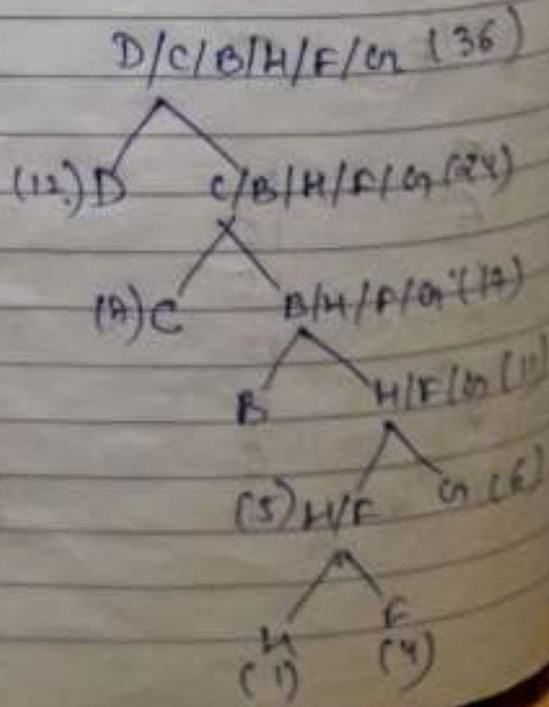
E	25
A	15
I	15
D	12
C	7
B/H/F/G	17



E	25
A	15
I	15
D	12
C/ B/H/F/G	24



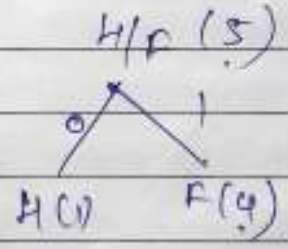
E	25
A	15
I	15
D/ C/B/H/F/G	36



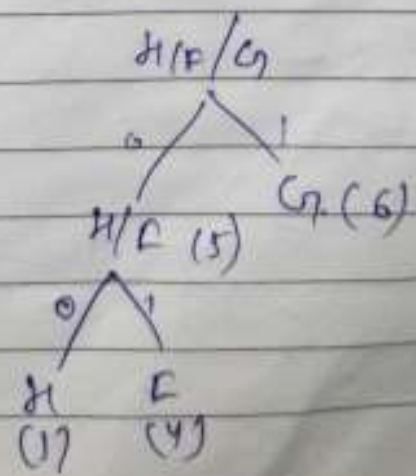
Q. Draw a Huffman tree for the following symbols whose frequencies are.

A	15	E	25
B	6	A	15
C	7	I	15
D	12	D	12
E	25	C	7
F	4	B	6
G	6	G	6
H	1	F	4
I	15	H	1

E	25
A	15
I	15
D	12
C	7
B	6
G	6
H/F	5



E	25
A	15
I	15
D	12
C	7
B	6
H/F/G	11

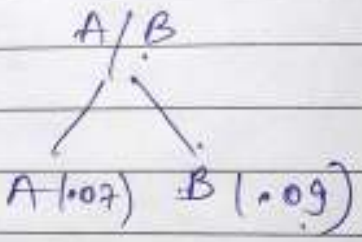


Q4. Suppose Characters A, B, C, D, E, F have probabilities .07, .09, .12, .22, .23, .27 respectively. Find the Huffman code and Huffman tree and what is the Average Code length.

A	.07	F	.27
B	.09	E	.23
C	.12 ⇒	D	.22
D	.22	C	.12
E	.23	B	.09
F	.27	A	.07

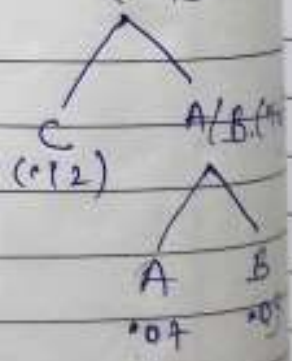
F	.27
E	.23
D	.22
C	.12
A/B	.16

⇒



⇒

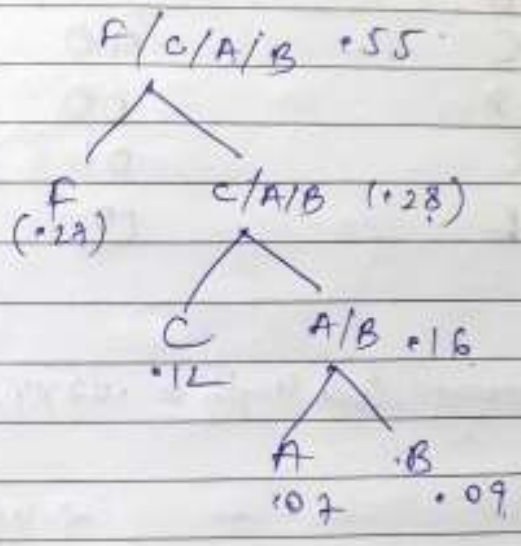
F	.27	F	.27
E	.23	E	.23
D	.22 ⇒	D	.22
A/B	.16	C/A/B	.28
C	.12		



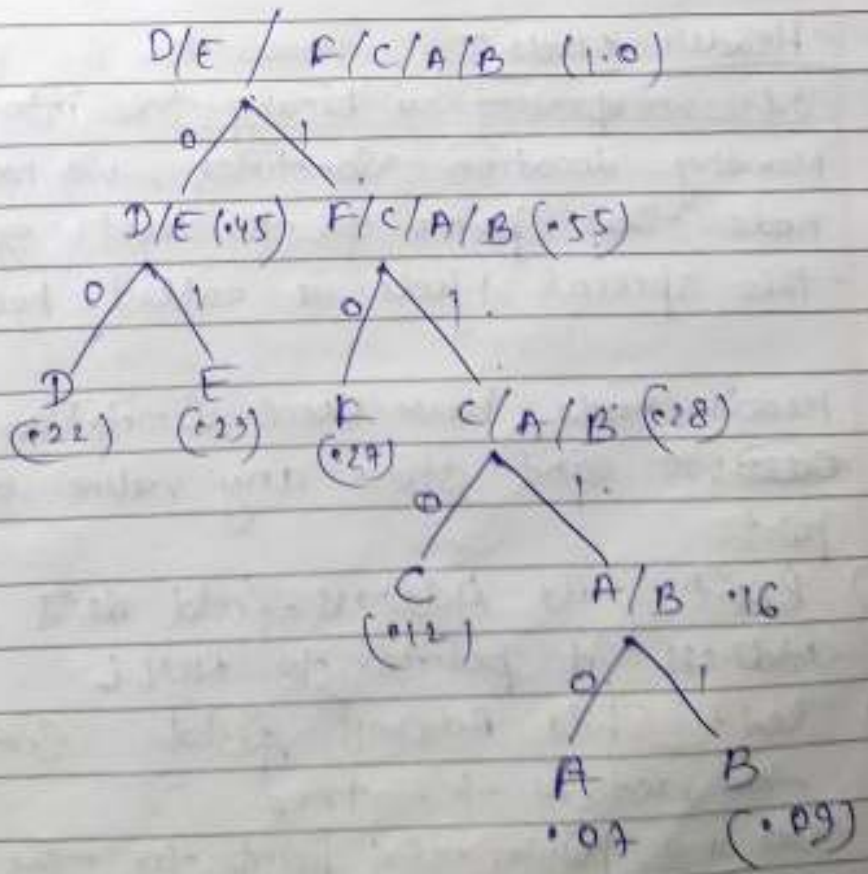
C/A/B	.28	C/A/B	.28
F	.27 ⇒	F	.27
E	.23	D/E	.45
D	.22		

⇒

D/E	.45		D/E	.45
C/A/B	.28	⇒	F/C/A/B	.55
F	.27			



F/C/A/B	.55	⇒	D/E / F/C/A/B	1.0
D/E	.45			



Symbol	Code	Code length
A	1110	4
B	1111	4
C	110	3
D	00	2
E	01	2
F	10	2

Average Code length = .07 x 4 + .09 x 4 + .12 x 3 + .22 x 2 + .
+ .27 .

= 2.64 bits/AP

Threaded Binary Tree :

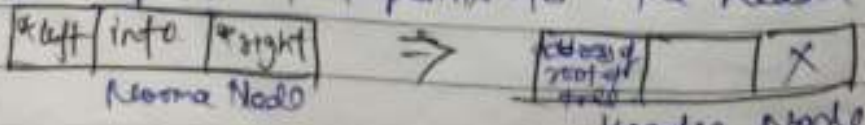
Header Node -

If we represent a binary tree into distributed memory location sometimes, we maintain a special node that points to the root of the tree and this special node is called header node.

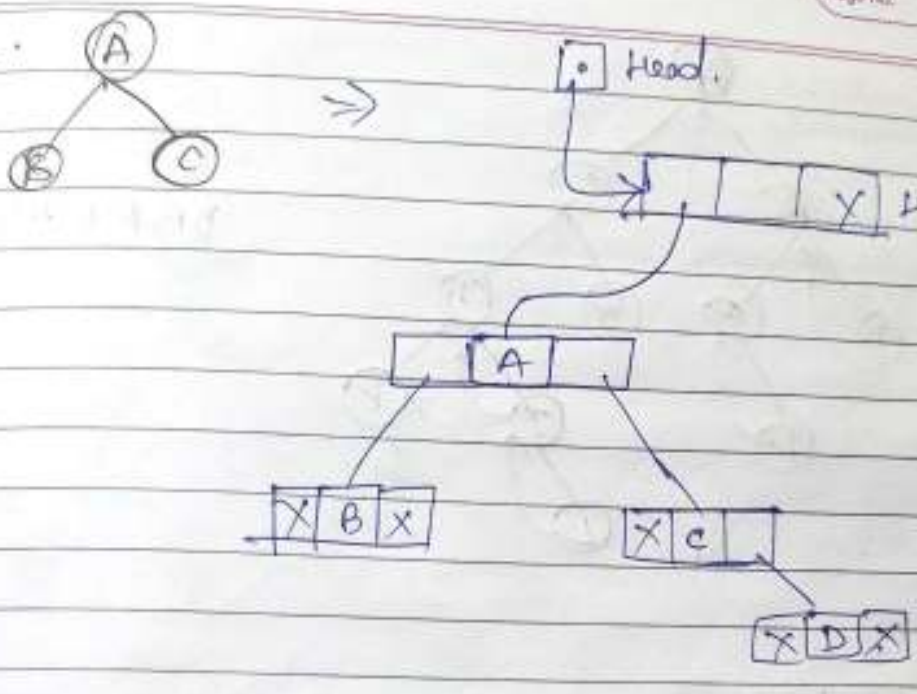
Header node have some limitation & limitation

1. we can't store any value at the info part
2. Right Child Address field does not contain an address it points to NULL.
3. Left Child Address field Contain address of the root of the tree.

Head is a pointer that points to the Header node



Eg.

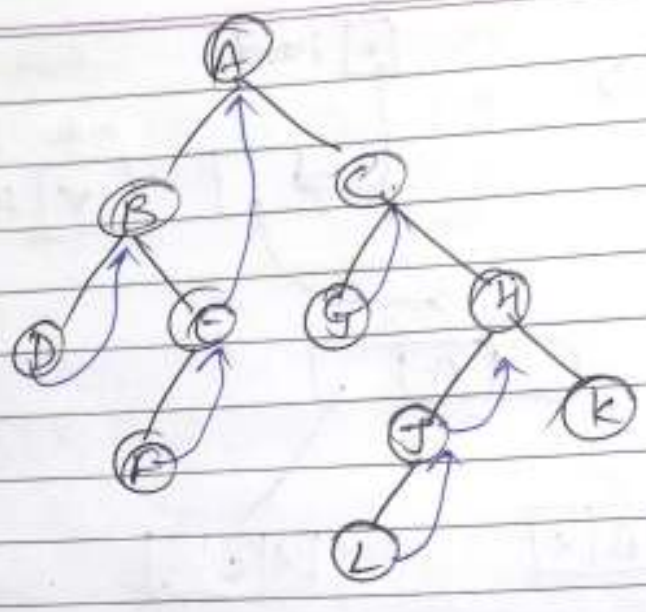


Header node is used to avoid illegal addresses stored in the NULL field.

Thread is a Mechanism of removal of NULL entry of the tree Node & this approach is called threading.
There are 3 types of threading :-

1. ~~Inorder~~ One way Inorder threading
2. two way Inorder threading.
3. two way Inorder threading with header node

1. One way Inorder threading.
In One way inorder threading a node having an empty right subtree points to its immediate inorder successive.



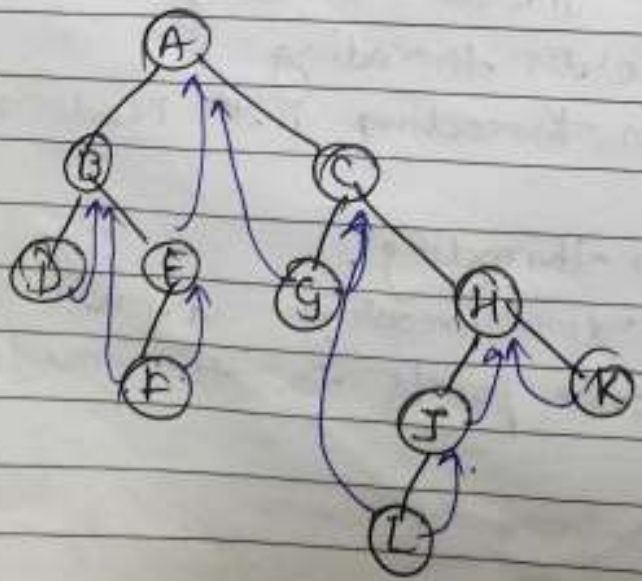
DBFEAGCLJHK

Drawbacks

It removes only from right subtree & can't remove the NULL entry for the node that comes in the last while traversing the tree inorder.

(2) two - way Inorder threading ⇒

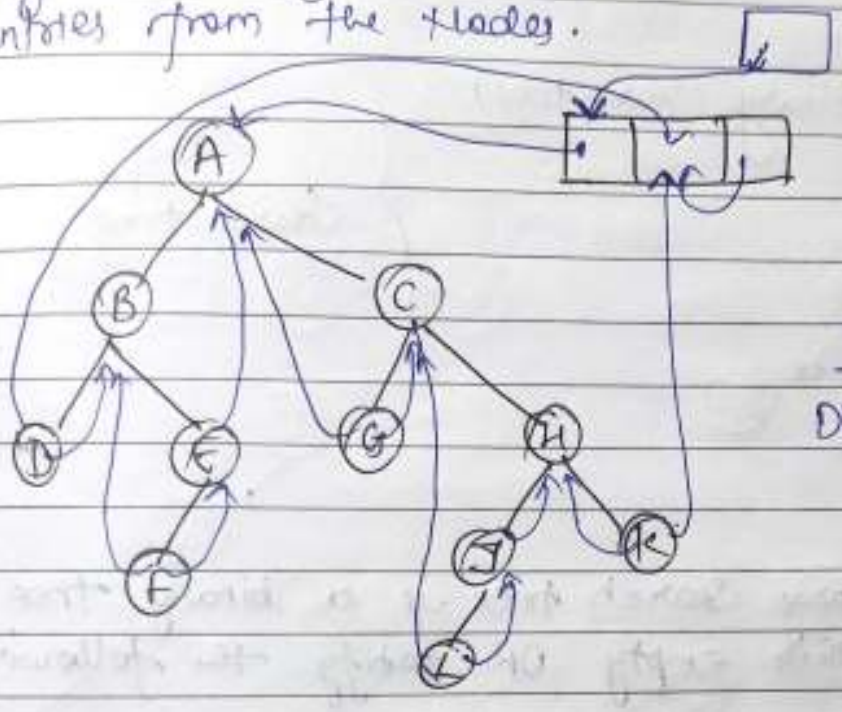
In two-way Inorder threading we can remove both the Remove both the NULL entries left and right but except the first & last node



Inorder -
DBFEAGCLJHK

(3) Two-way Inorder threading with header node ⇒

By Introducing Header we eliminate all the NULL entries from the nodes.



Inorder:-

DBFEAGCLJHK

14/10/2016

Date / /

Page No.

Shivalal

Unit - V

BST (Binary Search tree)
AVL
m-way
B-Tree
B+ Tree.
} Search tree.

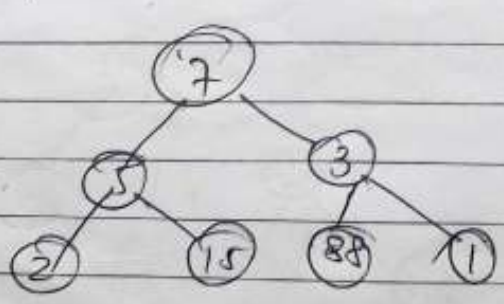
BST =>

A Binary Search tree is a binary tree. which is either empty OR satify the following rules,

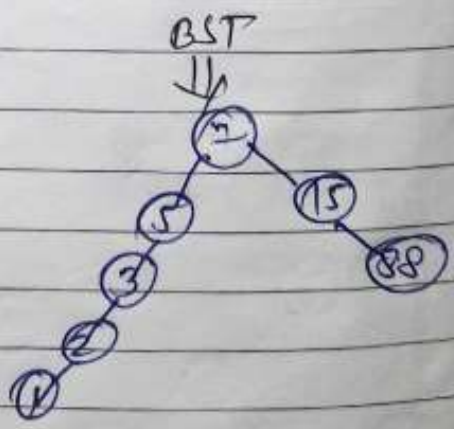
Rule (1) - The value of the key in the left child OR left subtree is less than the value of the root

(2) The value of the ~~tree~~ key in the right child OR right subtree is more OR equal to the value of the root.

for eg. 7, 5, 3, 2, 15, 88, 1



BST

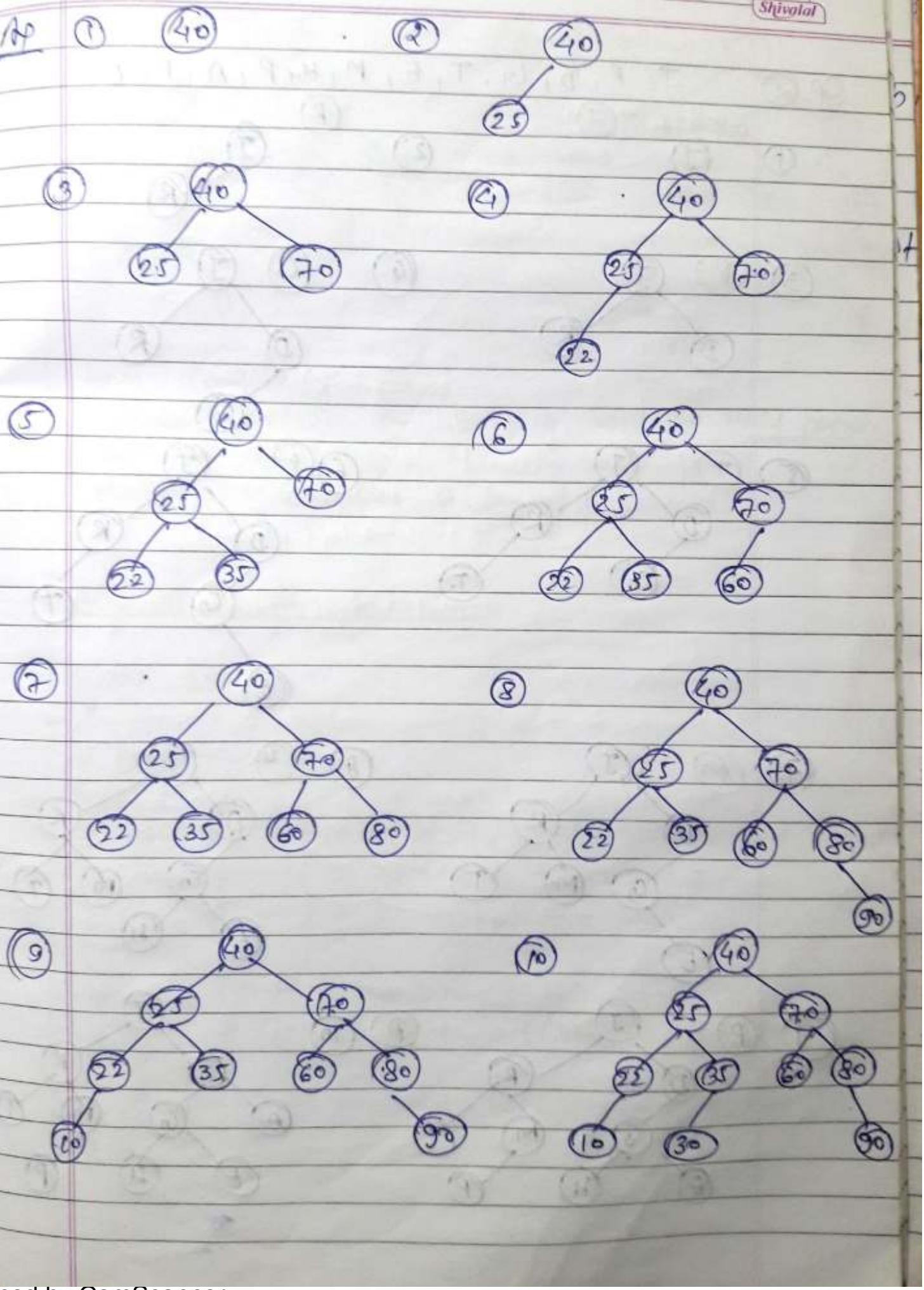


Qp Draw the BST for

40, 25, 70, 22, 35, 60, 80, 90, 10, 30

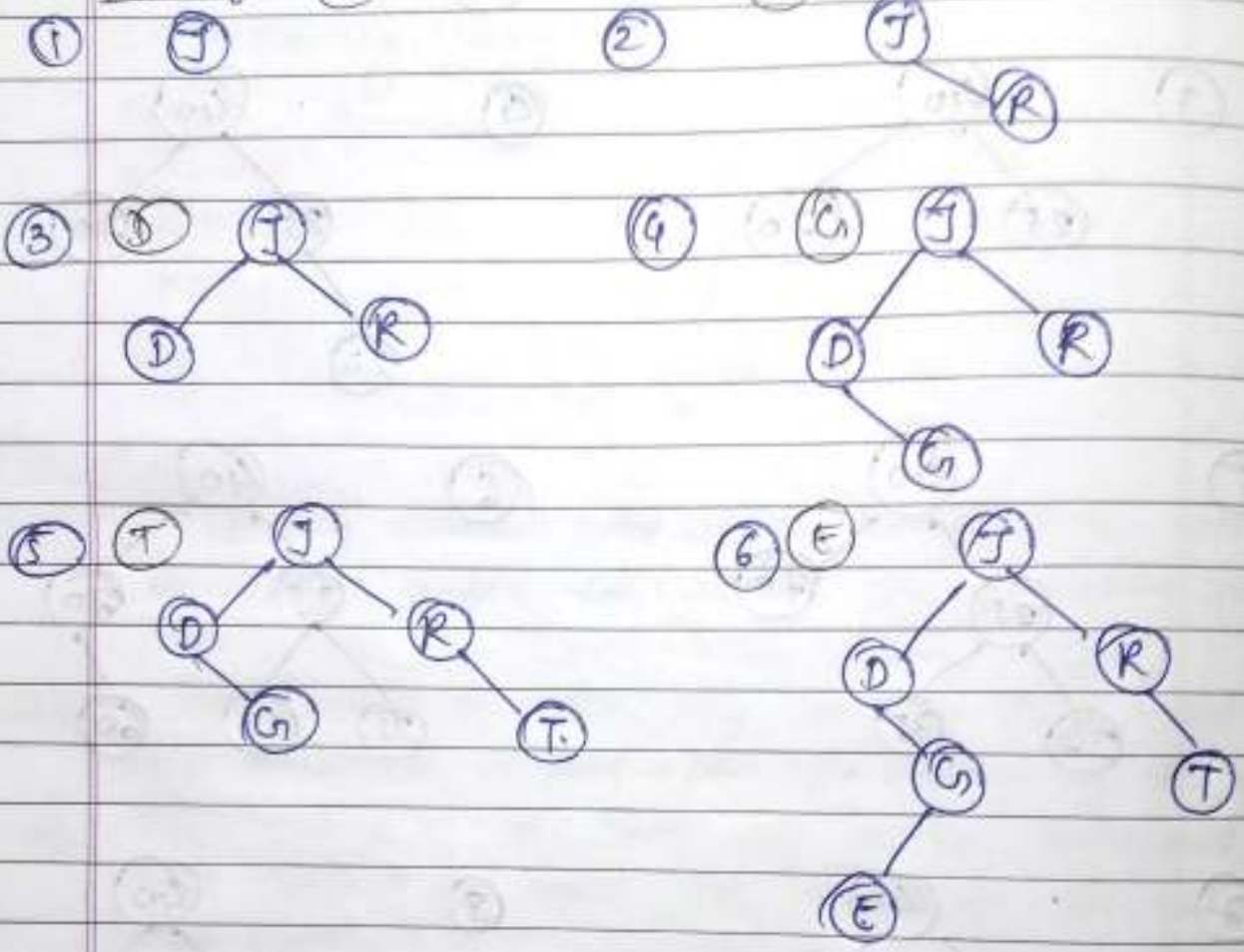
Date / /
Page No.
Shivalal

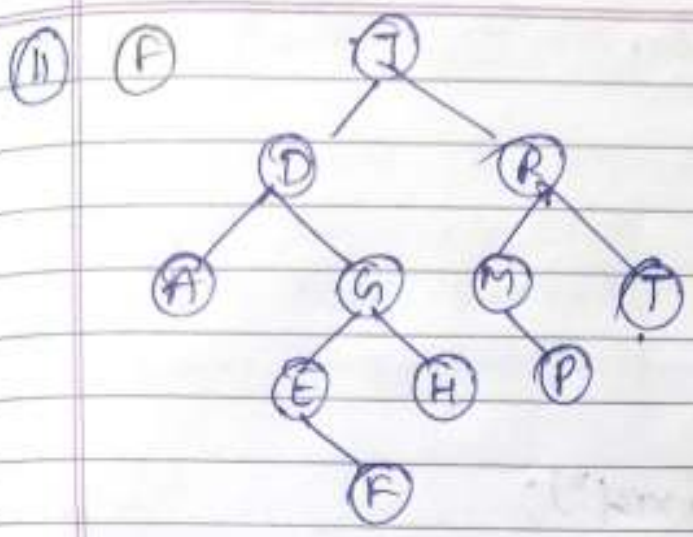
14

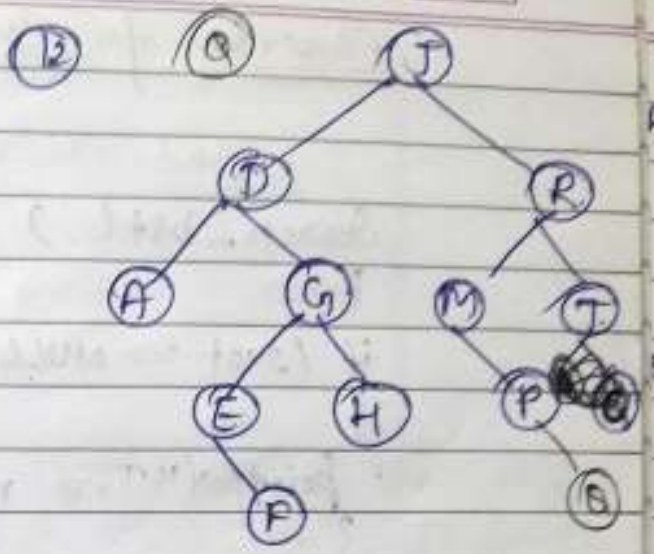


Q. (2) J, R, D, G, T, E, M, H, P, A, F, Q

Inserting - (J)







Here Complexity for Insertion, Deletion & Searching will be Half portion traversed upto height H.
Hence, O(h) or O(log n)

Algorithm for Insertion =>

```
Insert_bst( )
{
if (root == NULL)
{
Struct node *root = (Struct node *) malloc(Size of (Struct node));
    root → left = root → right = NULL;
      root → info = num;
}
 else if (root → info > num)
   root → left = insert_bst(root → left, num);
 else
    root → right = insert_bst(root → right, num);
}
```

Date / /
Page No.

Algorithm for Searching

```
Search_bst( )
{
if (root == NULL)
{
 printf("Tree not exist");
}
else if (num == root->info);
 {
  printf(" element found");
else if (num < root->info);
   search_bst (root -> left, num);
  else
   search_bst (root -> right, num);
}
```

Algorithm to find Minimum value & Maximum value

```
Min_bst(struct node *t)
{
while (t -> left != NULL)
   t = t -> left;
}
   return(t);
}
```

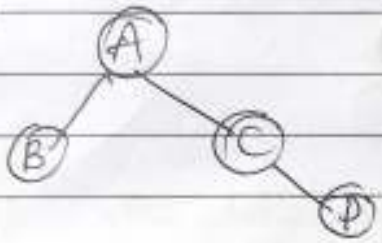
```
Max_bst (struct node *
 {
while (t -> right != NUL
    t = t -> right
 }
    return(t);
}
```

Algorithm for Deletion in binary search tree

There are 3 cases in deletion.

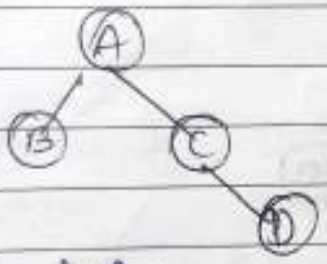
1. Node has no child
2. Node has one child
3. Node have two child.

Case - 1 =>



for deleteting B. we will put NULL in left pointer of A.

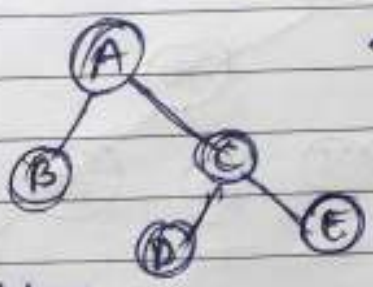
Case 2 =>



i.e.

if we want to delete C., than right pointer of A will contain address of D.

Case 3



Here.

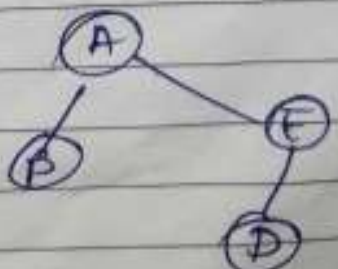
for deleting C. than c will be replaced by inorder successor of that node.

i.e.

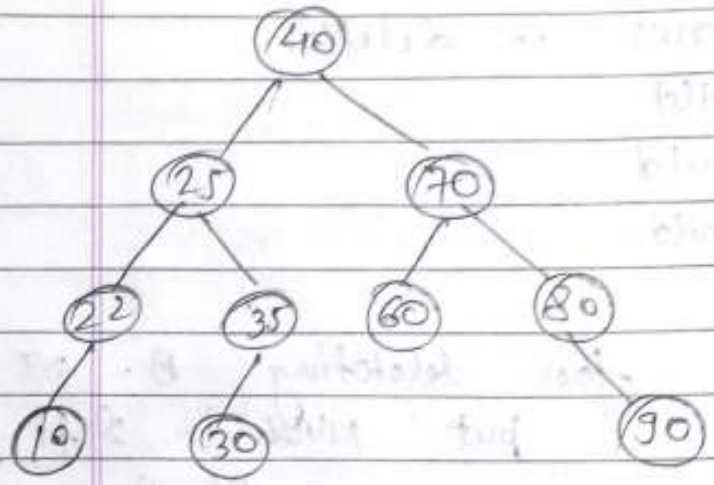
Inorder = DCE

So C will be replaced by E

=>

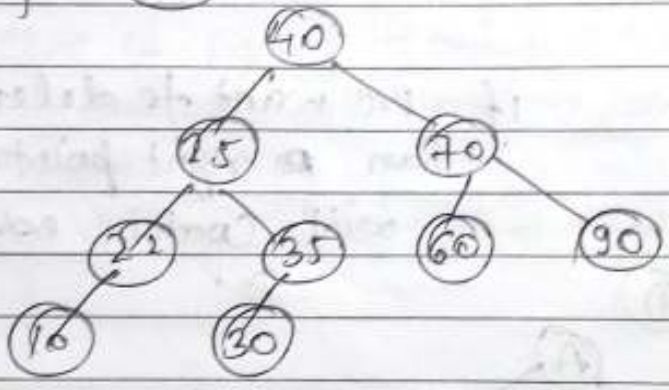


Q. Delete 80, 30 from given tree

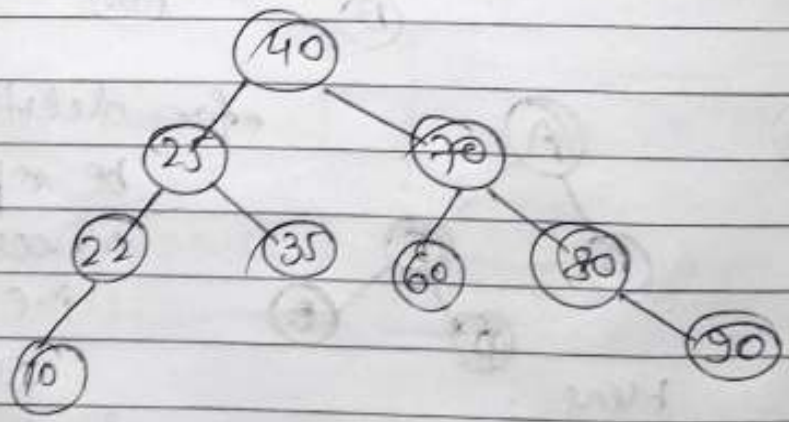


Answer

Case-I ⇒ for 80



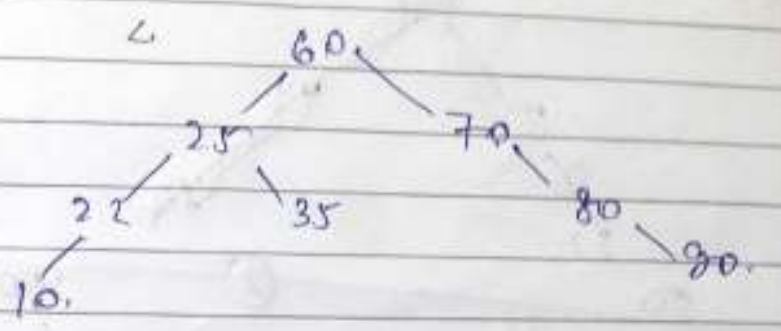
Case-II ⇒ for 30



Inorder → 10, 22, 25, 35, 40, 60, 70, 80, 90

Date / / Page No. Shivalal

Case-III for (10)



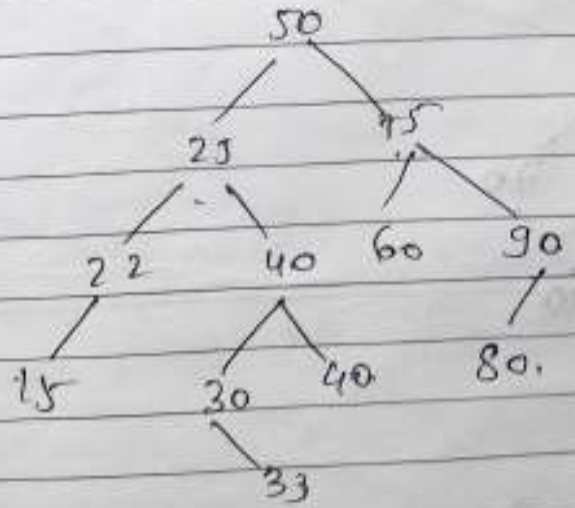
Q-1 Suppose the following 8 numbers are inserted inorder into an empty binary search tree. Draw the tree.

50, 33, 44, 22, 77, 35, 60, 40

Q-2-1 Consider the given BST & perform the following operation.

(a) Add 20, 15, 88

(b) Delete 22, 25, 75



Page No.

Shivalal

Ans-1 ① 50

② 50
33

③ 50
33 44

④ 50
33 44
22

⑤ 50
33 77
22 44

⑥ 50
33 77
22 44
35

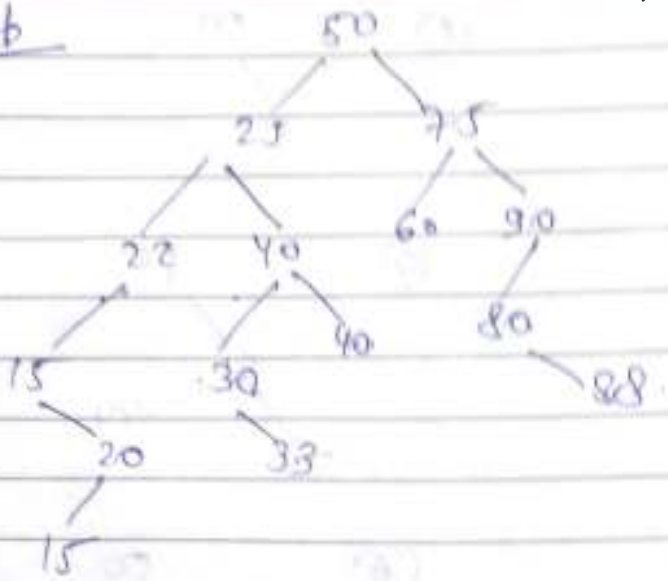
⑦ 50
33 77
22 44 60
35

⑧ 50
33 77
22 40 60
35 40

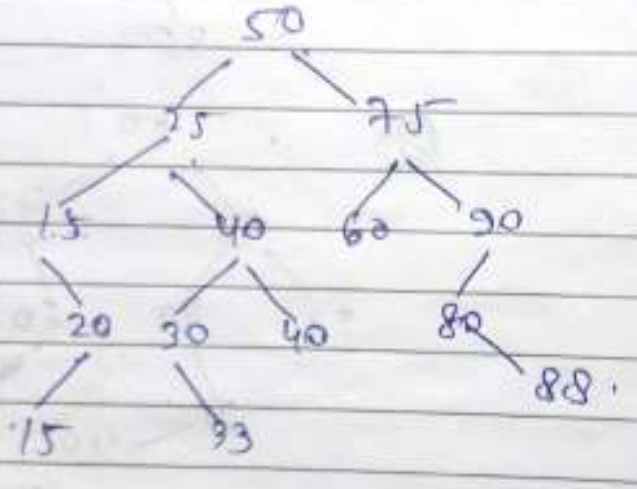
Ans-2 ① 50
25 75
22 40 60 90
15 30 40 80
20 33

② Step 50
25 75
22 40 60 90
15 30 40 80
20
15

(5) Step

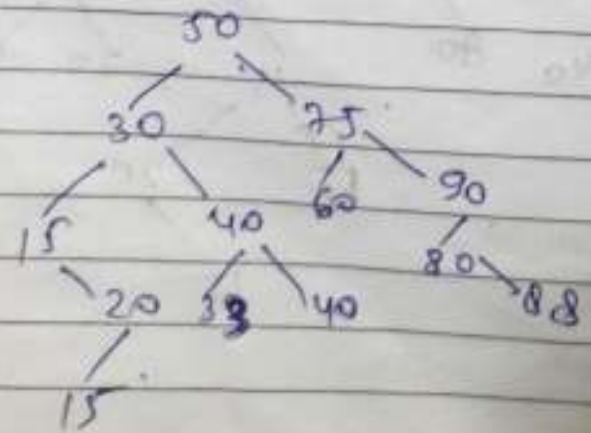


(b) Delete 22

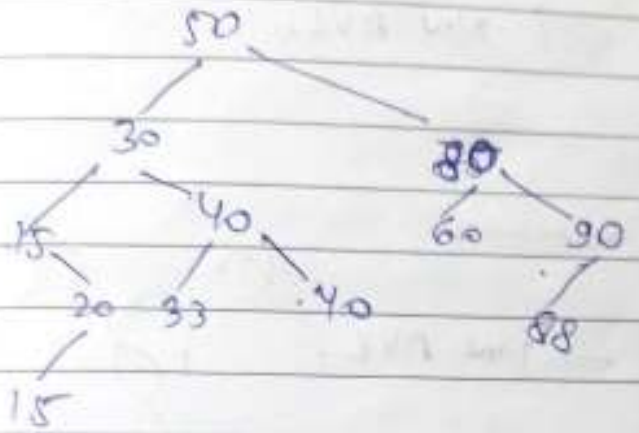


Delete 25

Inorder → 15, 15, 20, 25, 30, 33, 40, 40, 50, 60, 75, 80, 88, 90.



Delete - 75 preorder successor - 88



Ab

AVL Tree (Height Balanced Tree)

(Adelson Velski & Landis)

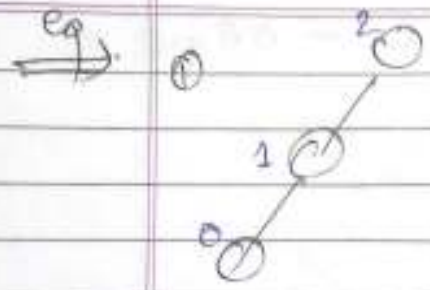
An empty binary search tree is an AVL tree.
An non-empty binary tree is an AVL if given T^L and T^R to be the left & Right Subtree of T and $h(T^L)$ and $h(T^R)$ is to be the height of left & Right Subtree Respectively. T^L and T^R are AVL tree and Hence

- $|h(T^L) - h(T^R)| \leq 1$

Balance factor (BF)

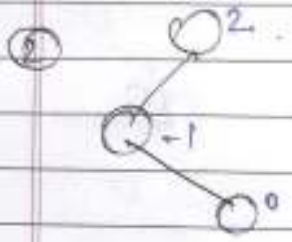
For an AVL tree the balance factor of a Node can be either Zero, One or -1.
It has the BST properties.

eg.



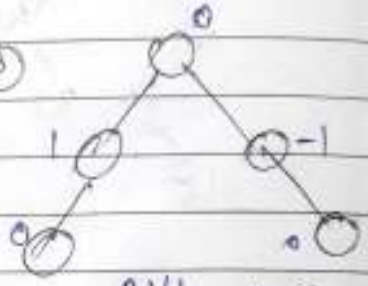
→ Not AVL.

(2)



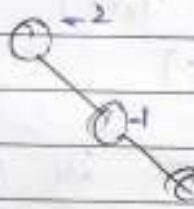
⇒ Not AVL.

(3)



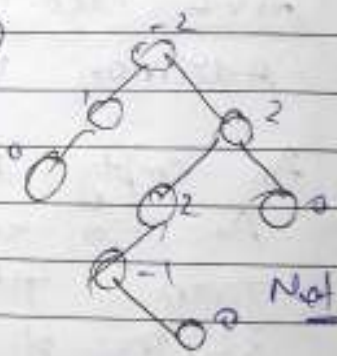
AVL ✓.

(3)



⇒ Not AVL.

(4)



Not AVL

Insertion in AVL tree ⇒

Insertion in AVL tree is similar to the binary Search tree, After insertion of the element the balance factor of any node in the tree is affected so the BST is unbalanced and we re-sort the techniques that known as Rotations.

To perform Rotation it is necessary to identify a specific Node A whose balance factor is neither 1, 0 or -1.

The Rebalancing rotations are classified as.

1) LL
2) RR
3) LR
4) RL

Date / /
Page No.
Shivalal

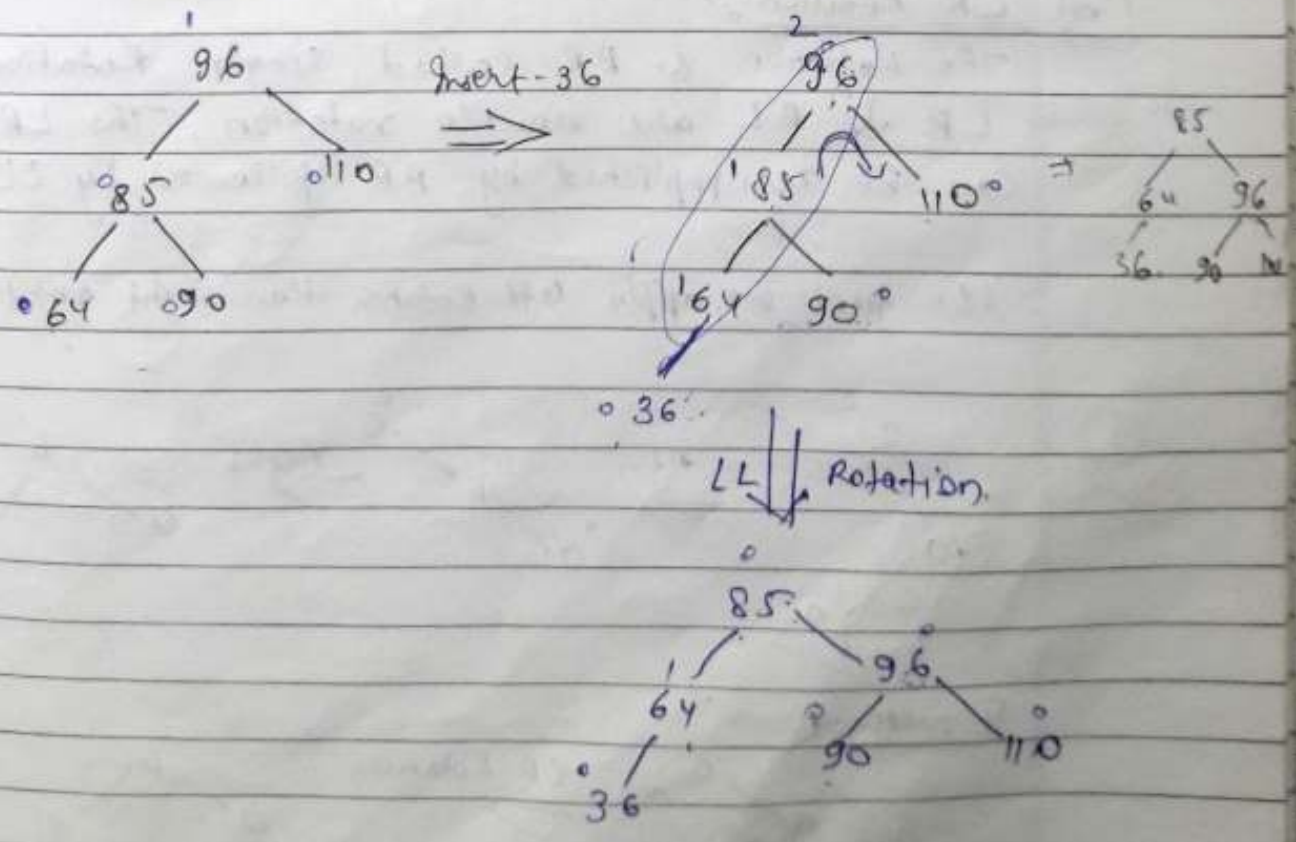
(i) LL =>
The Inserted Node is in the left Subtree of ~~Node A~~ left subtree of Node A.

(ii) RR =>
The Inserted Node is in the Right Subtree of right subtree of node A

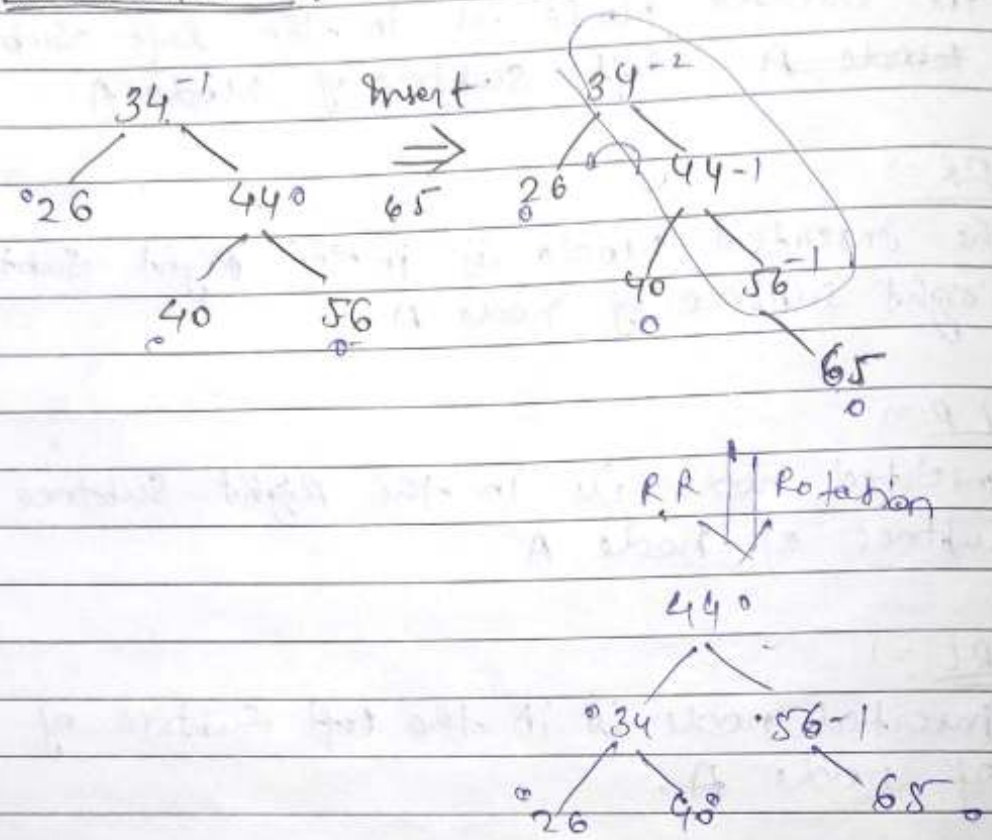
(iii) LR =>
Inserted node is in the Right Subtree of left Subtree of node A.

(iv) RL =>
Inserted node is in the left Subtree of Right Subtree of node A.

(i) LL Rotation :-



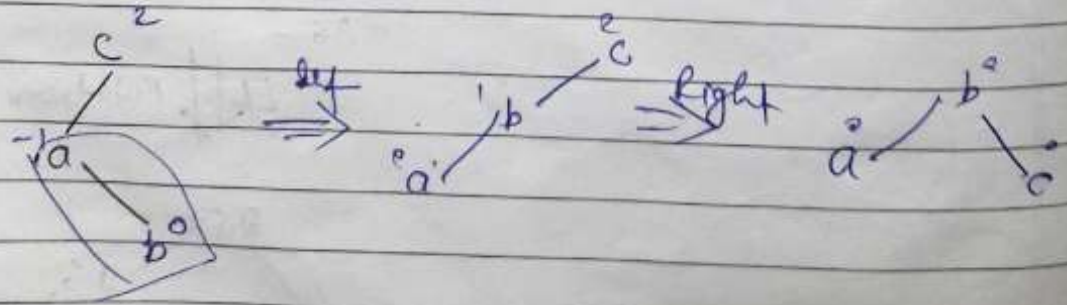
(2) RR Rotation :-



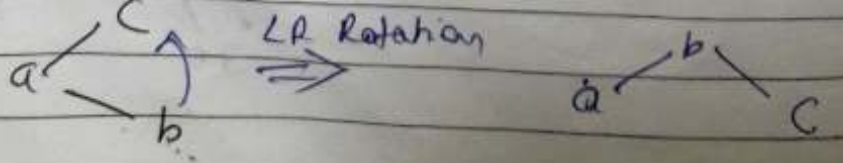
(3) LR Rotation :-

The Rotation & RR called single Rotation & LR & RL are double rotation. The LR rotation can be accomplished by RR followed by LL.

i.e. first we apply left Rotation than Right rotation.

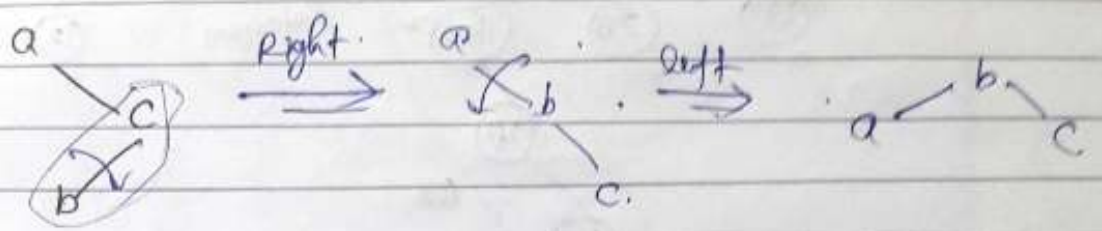


II method

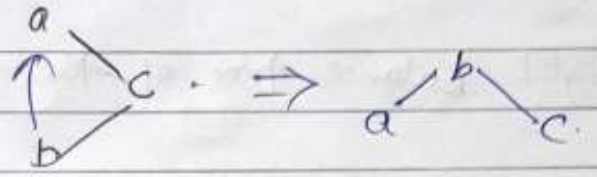


(4) RL Rotation :-

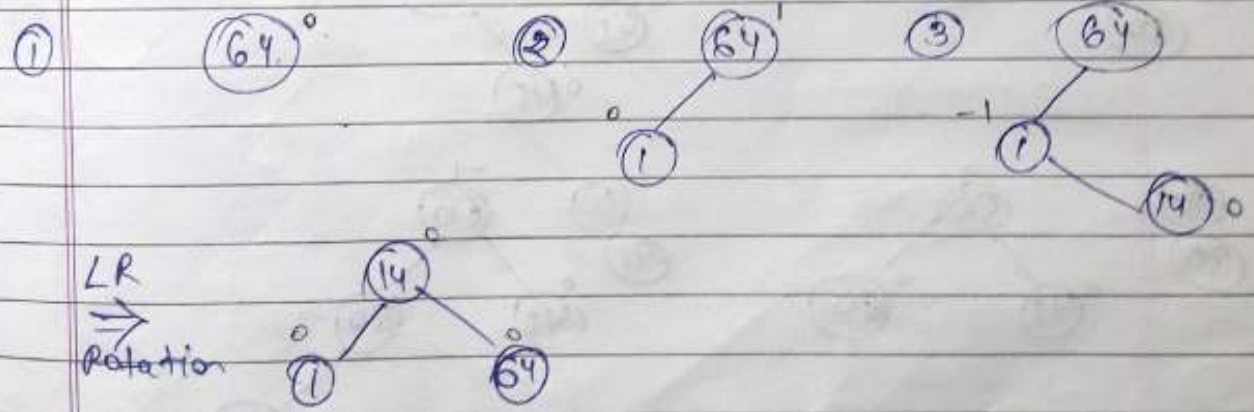
Here we apply first Right & then Left Rotation.



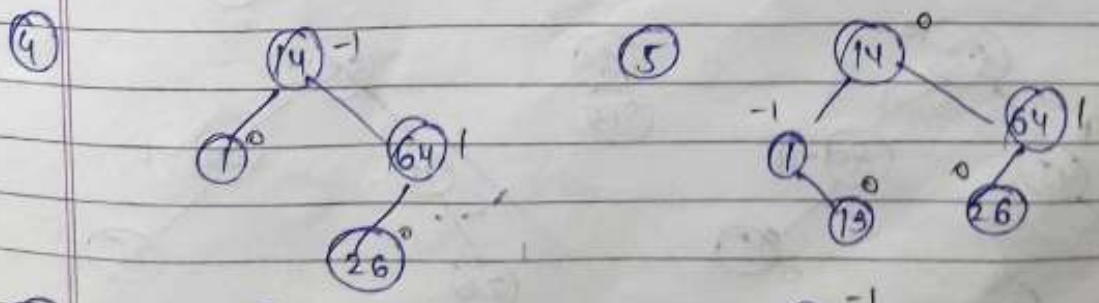
II method

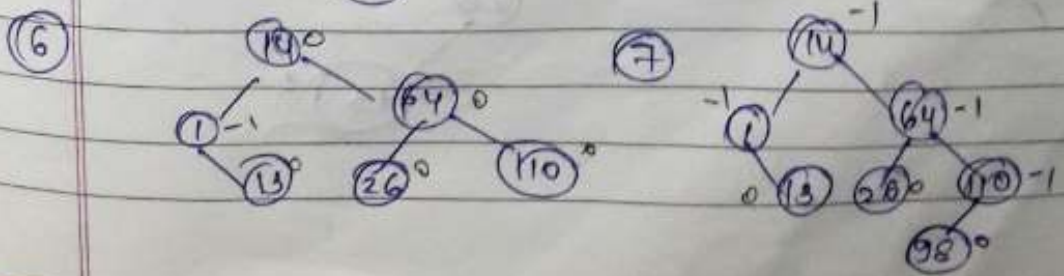


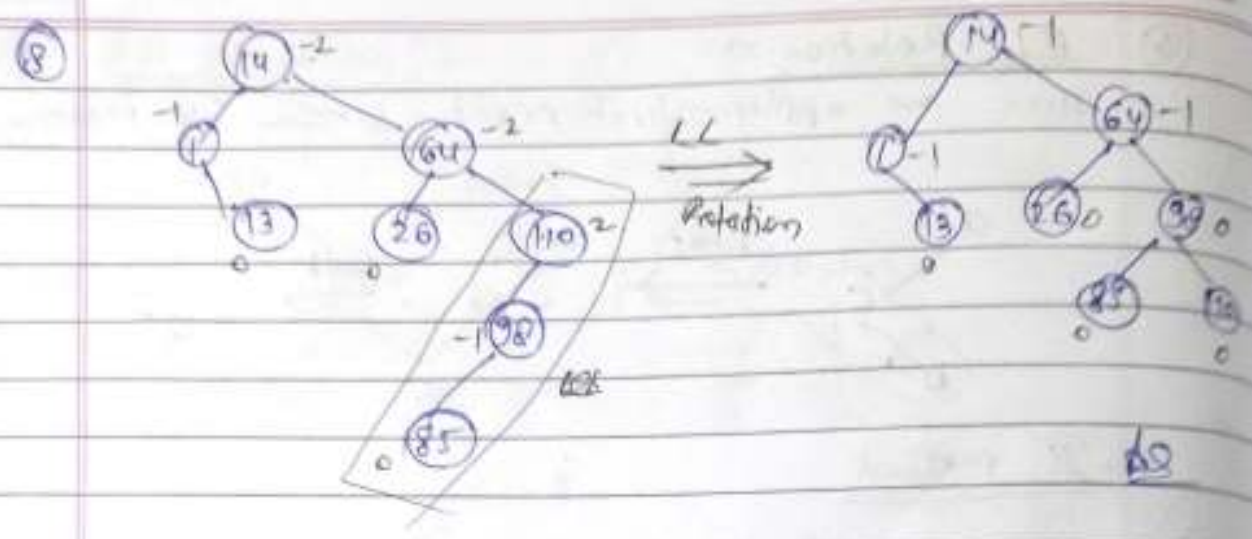
Q. Construct AVL tree by inserting 64, 1, 14, 26, 13, 110, 98, 85.



LR ⇒ Rotation





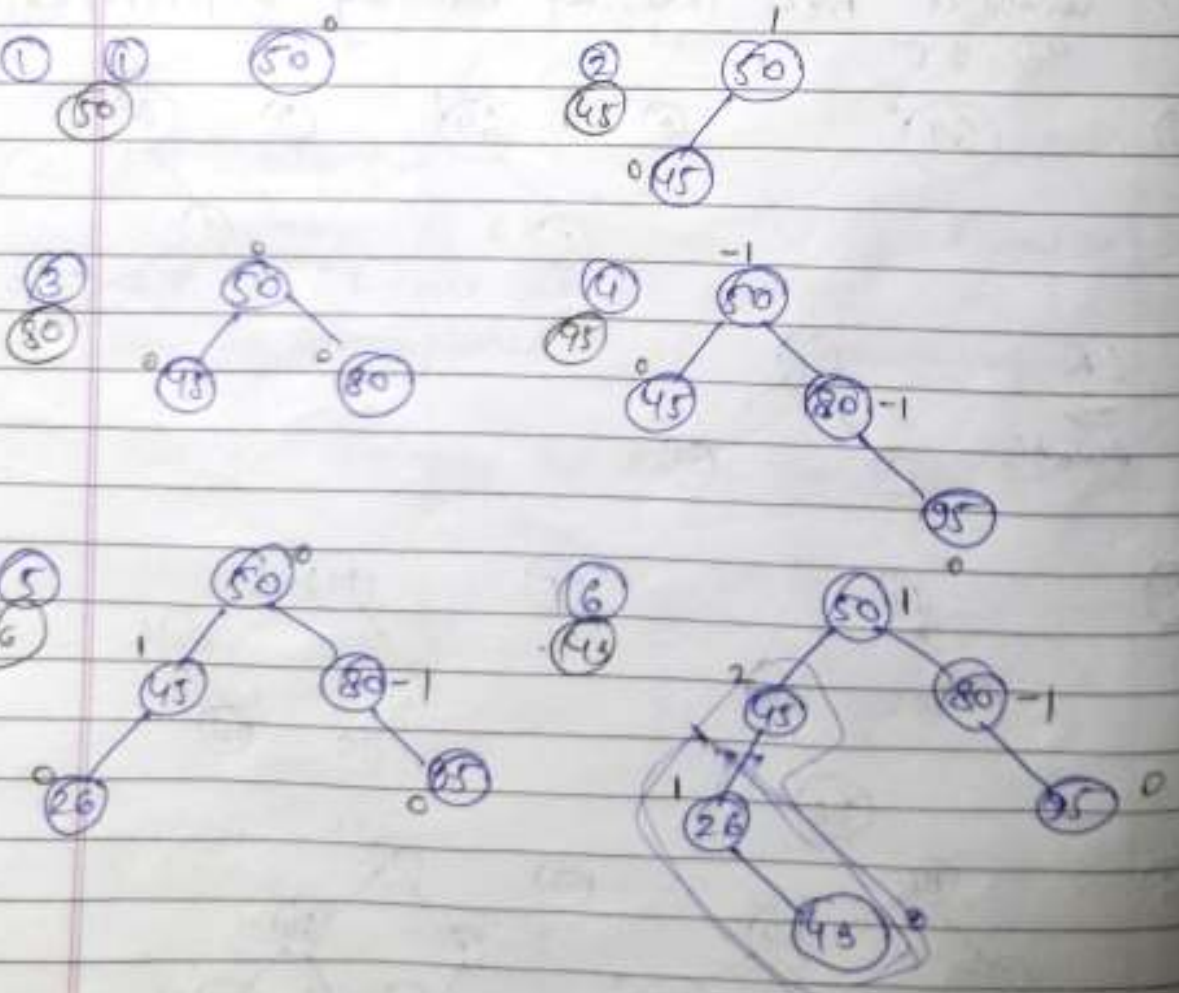


Q4 Construct height balance tree of the given data.

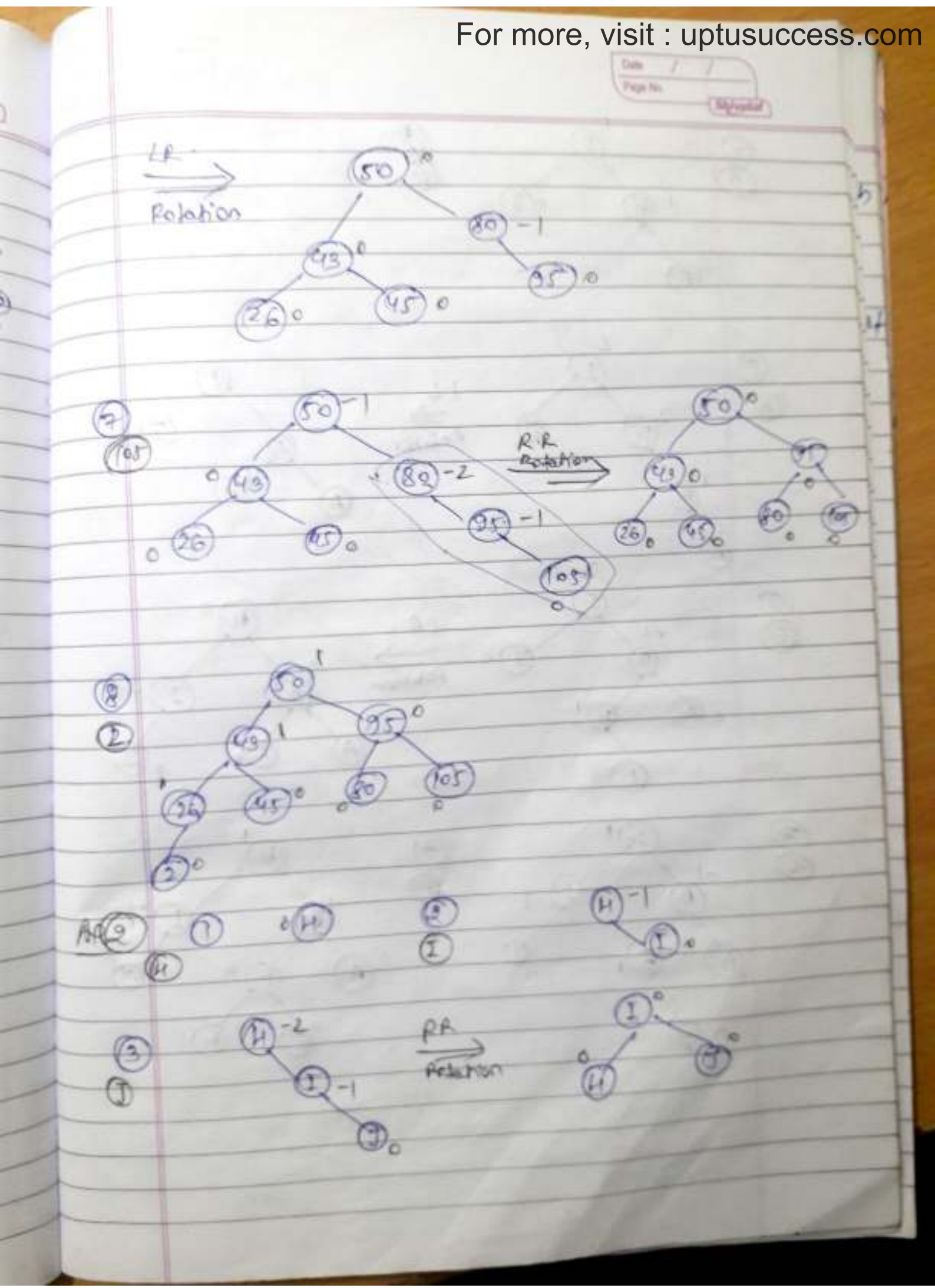
(i) 50, 45, 80, 95, 26, 43, 105, 2.

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

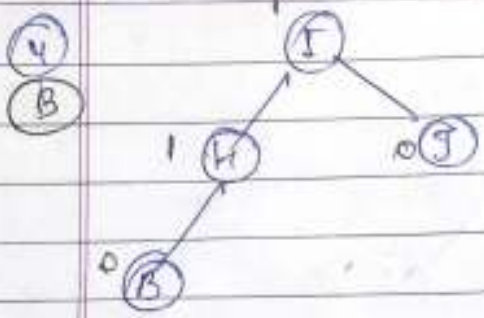
Ans. (i)



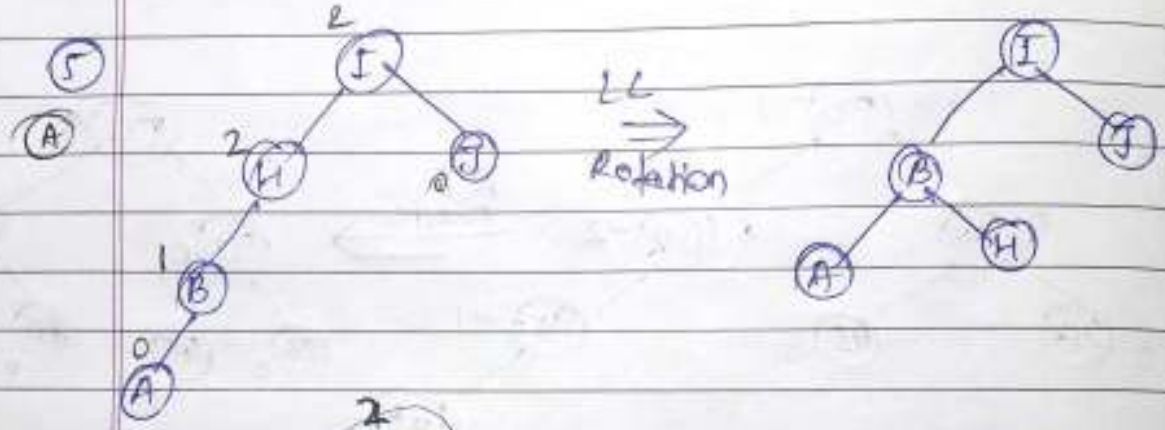
Scanned by CamScanner



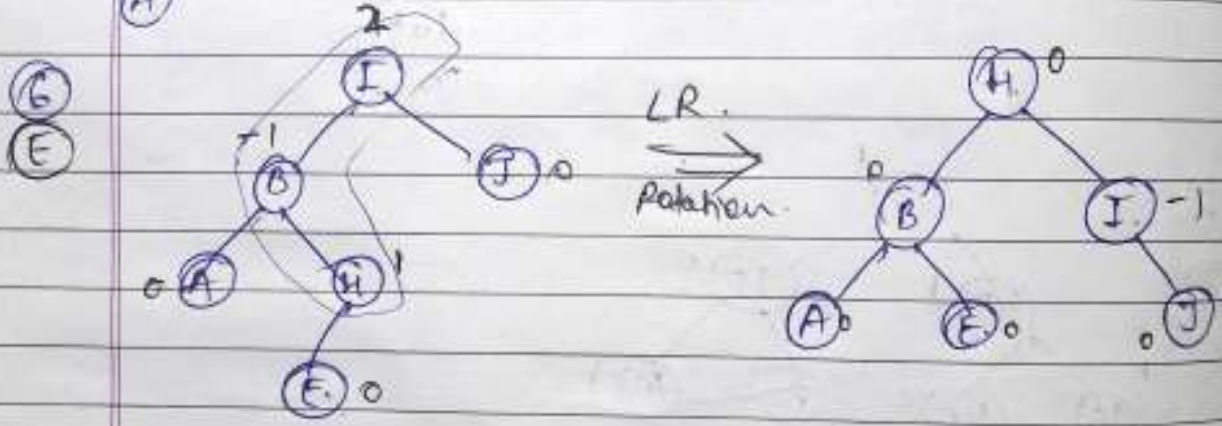
4 B



5 A

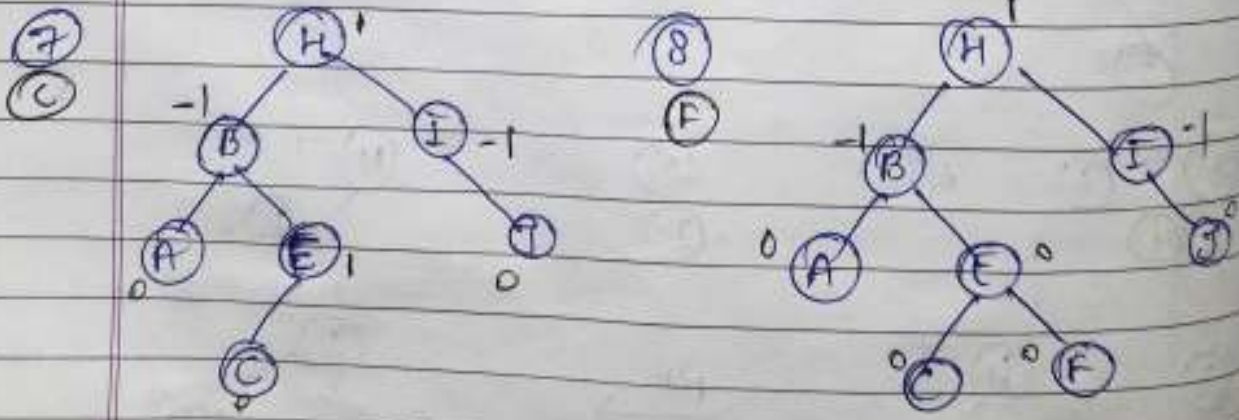


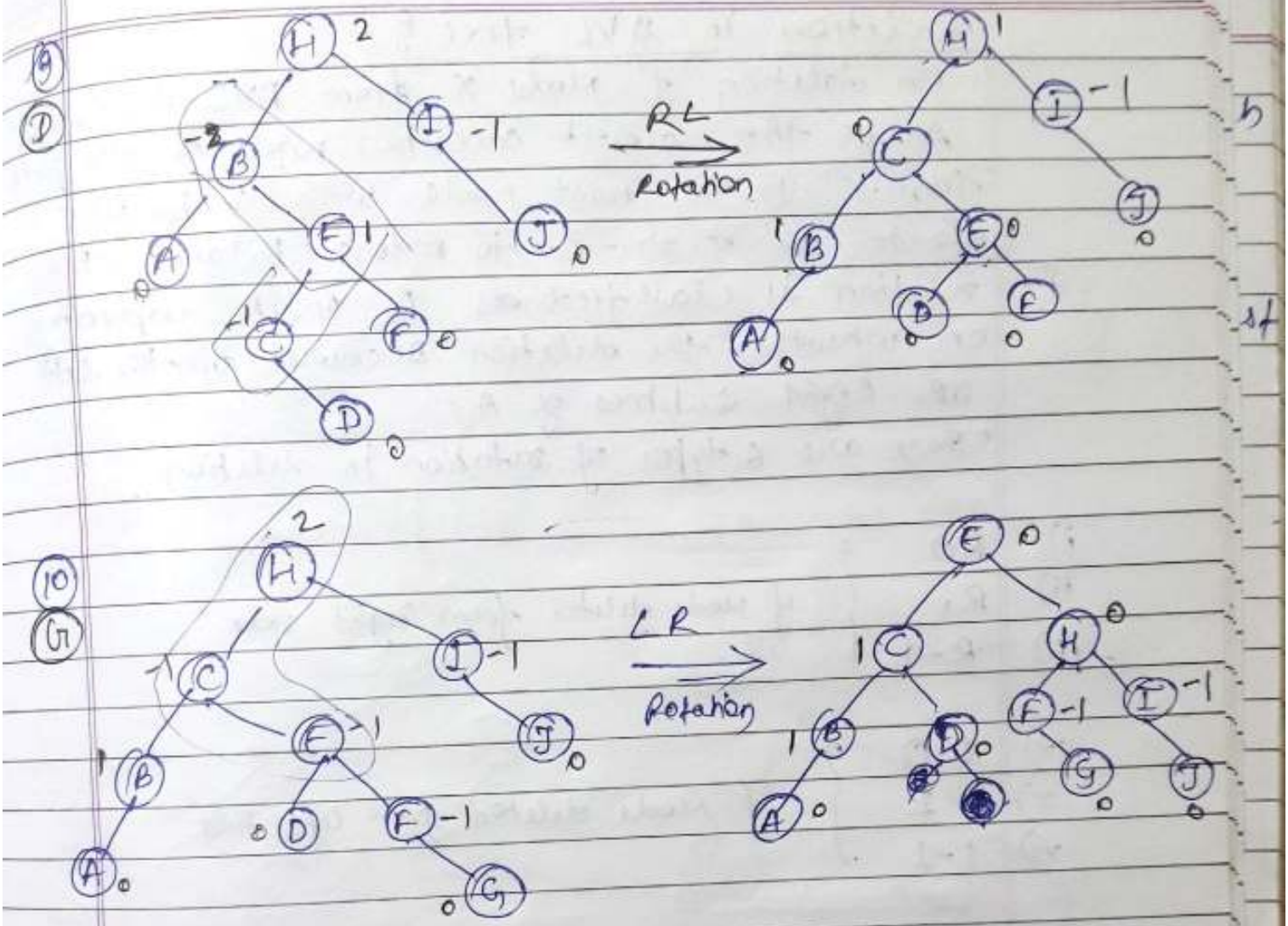
6 E



7 C

8 F





Deletion in AVL tree -

On deletion of node X from AVL tree. Let A be the closest ancestor node on the path from X to the root node with a balance factor of 2 or -2. to restore balance the rotation is classified as L or R depending on wheather the deletion occoured on the left OR Right subtree of A.

There are 6 types of rotation in deletion.

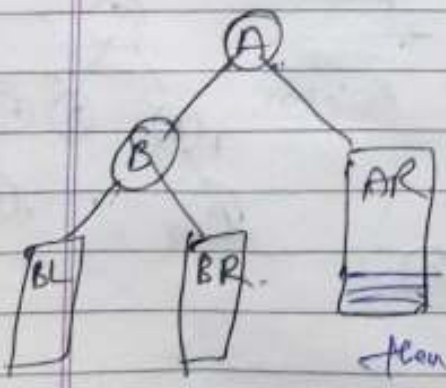
i) R0
ii) R1
iii) R-1

} if node deleted from right side

iv) L0
v) L1
vi) L-1

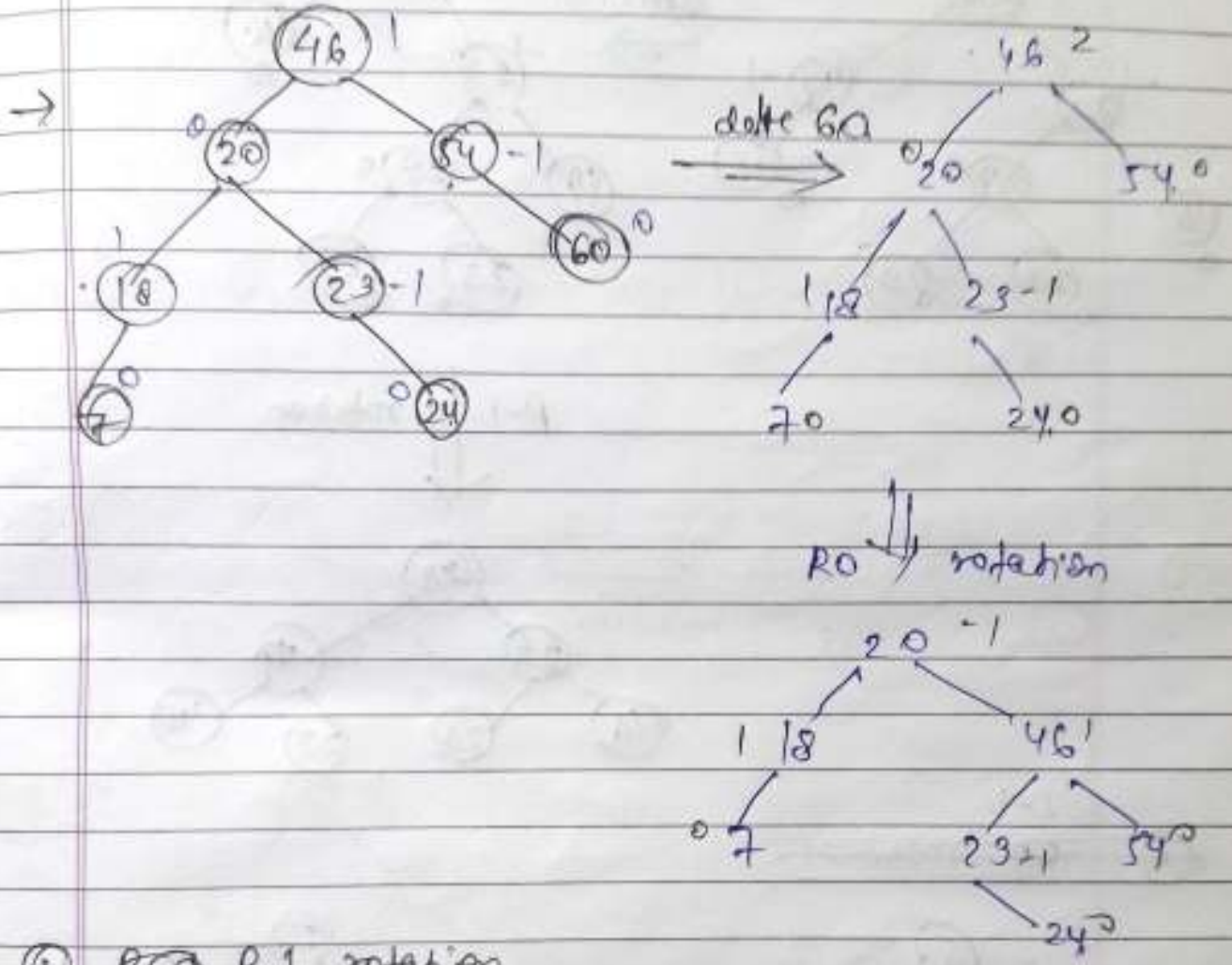
} if node deleted from left side.

1. ~~R0 rotation deletion~~ rotation -

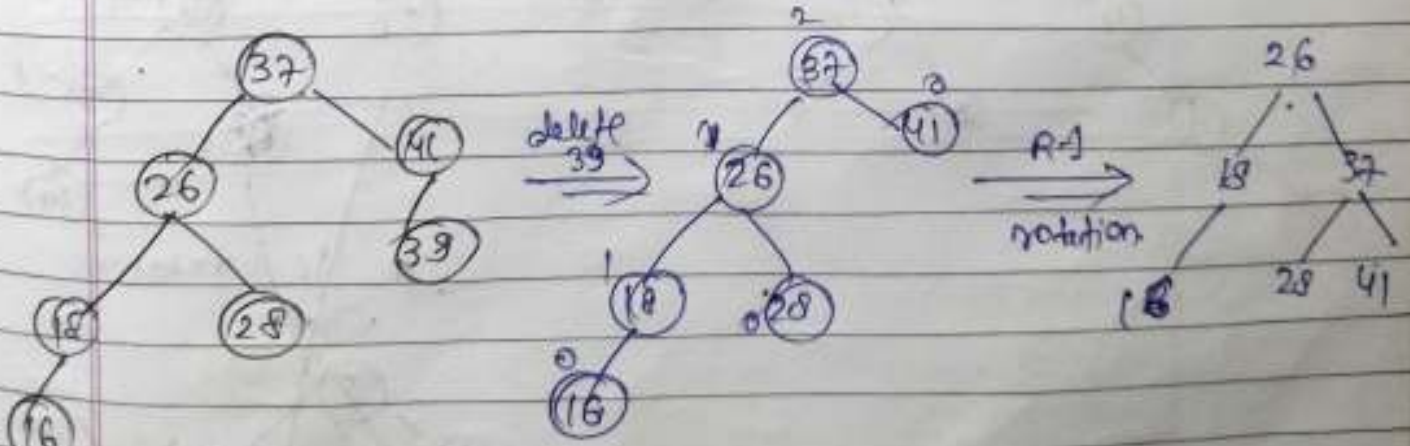


x - deleted
then check B.F of B
& apply rotation accordingly 0, 1, -1

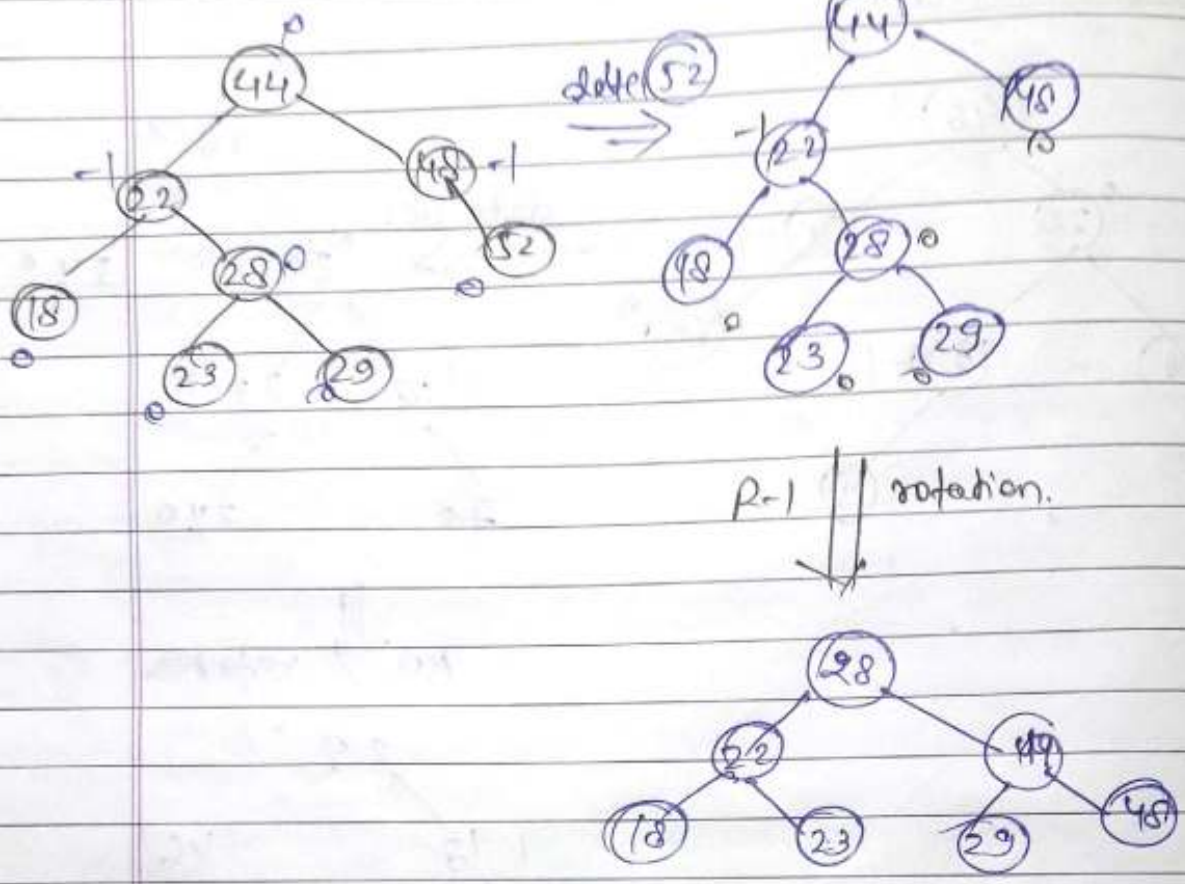
① RO rotation =>



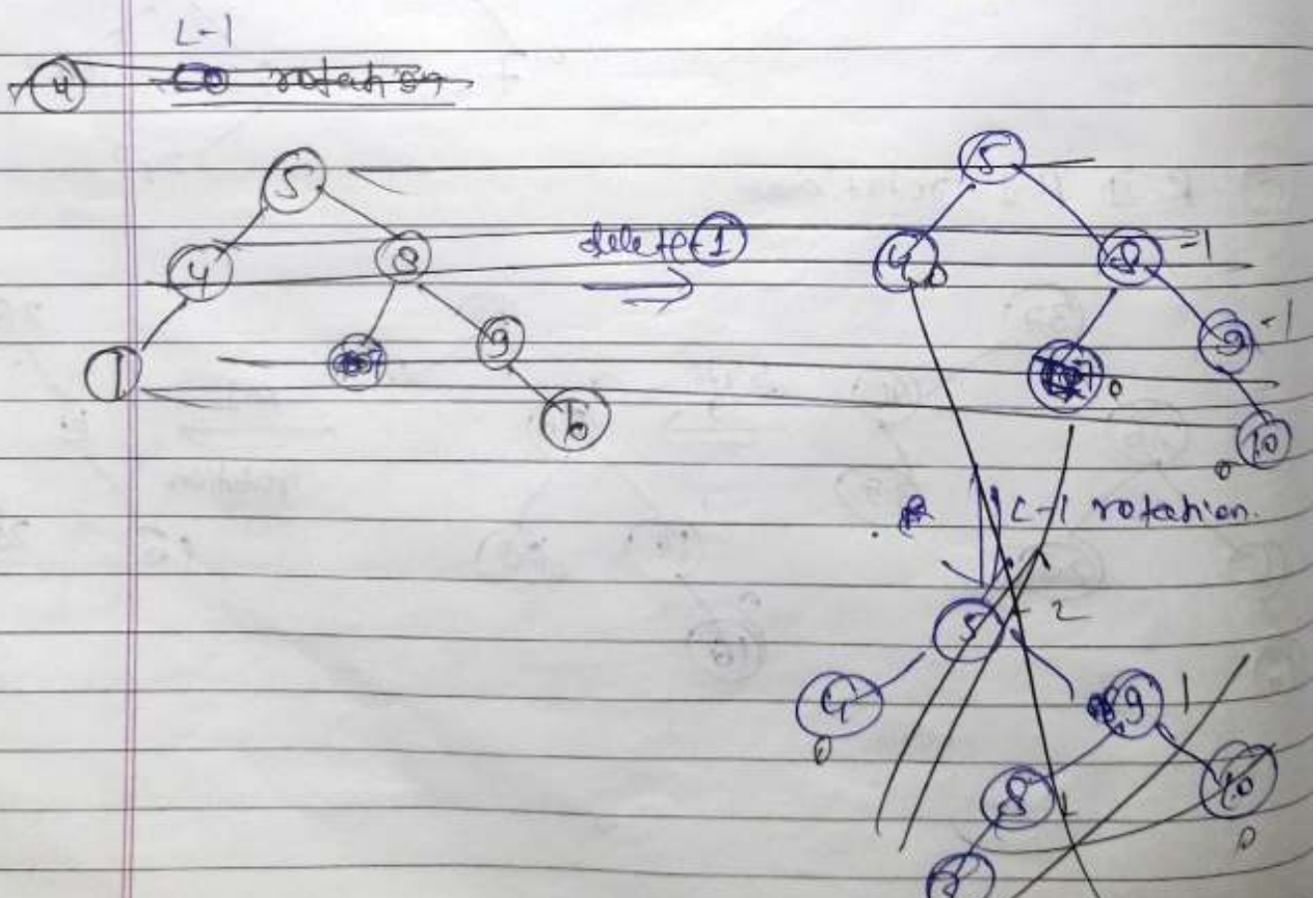
② RR R1 rotation



(3) R-1 rotation.



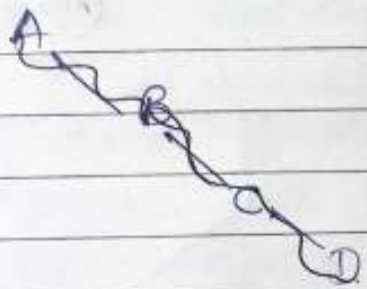
L-1 rotation



Date / /
Page No.
Shivalal

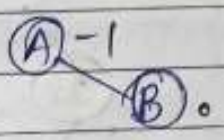
Q Create AVL tree by inserting A B C D E & delete D E

Ans

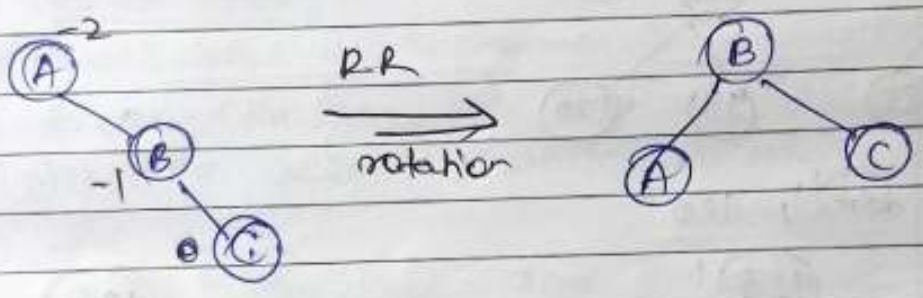


Step (1) A. 0

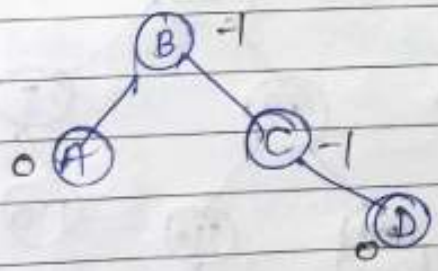
Step - (2)



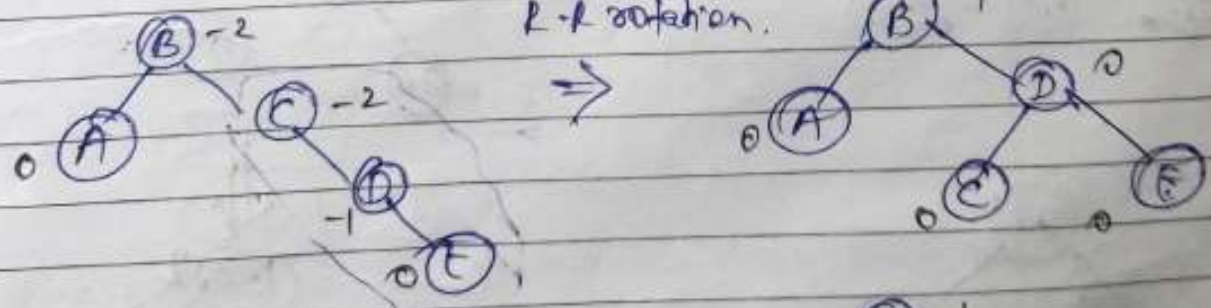
Step - (3)

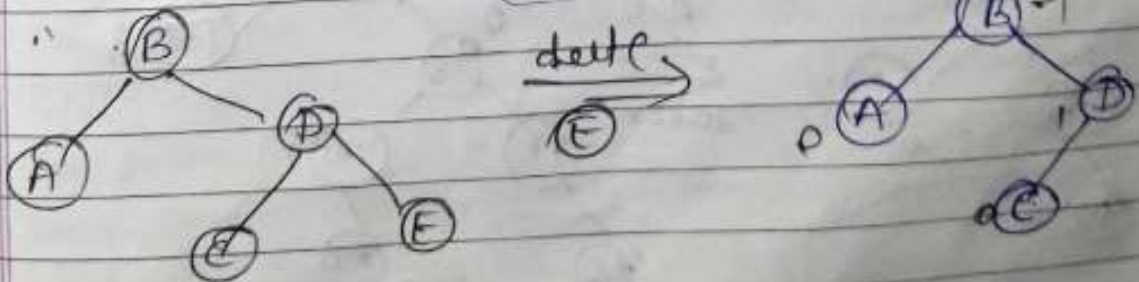


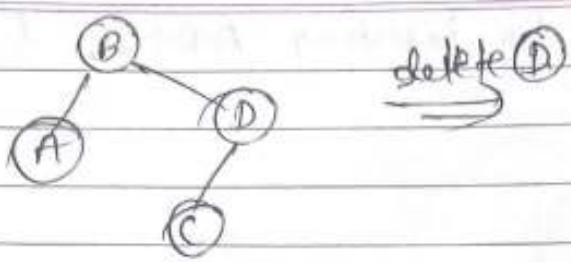
Step (4)

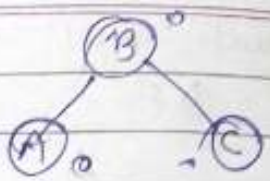


Step - (5)

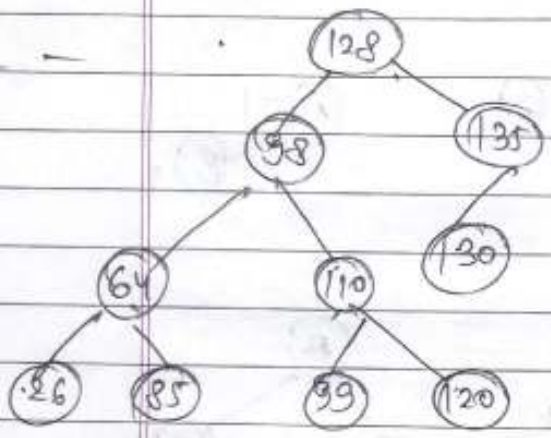






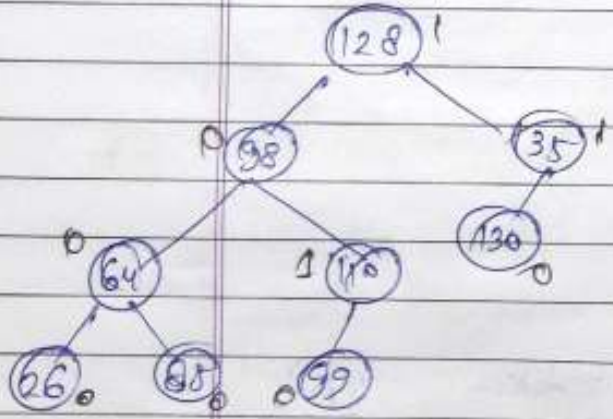


Eg -

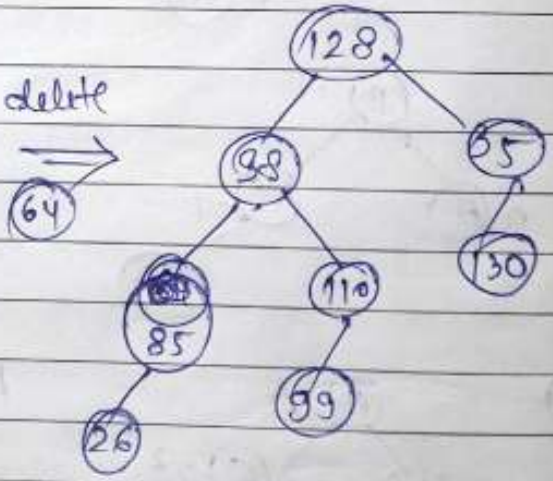


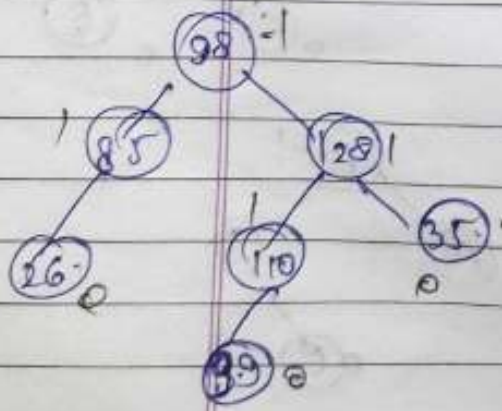
delete 120, 64, 130.

delete 120.

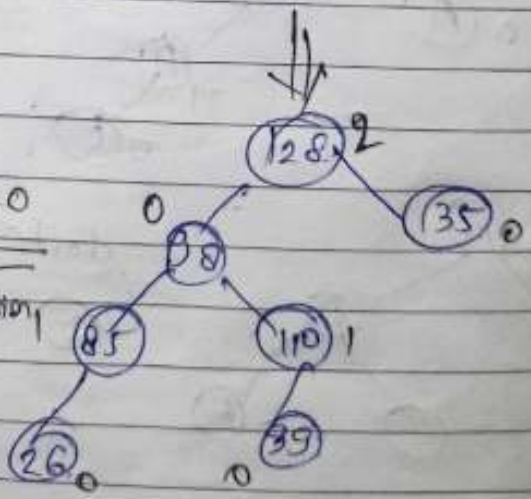


delete 64





R.O rotation



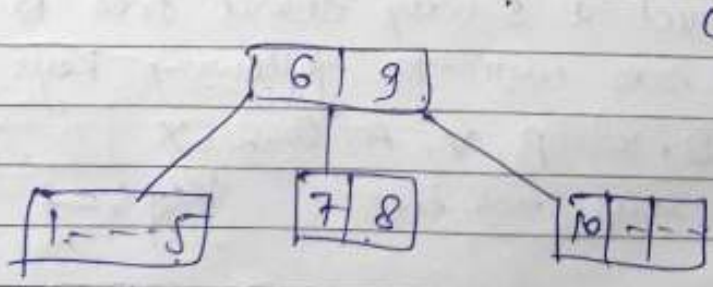
Date / /
Page No.
Shivalal

All the data Structure discussed, the data stored in internal memory and support internal information retrieval.

To favour retrival & manipulation of data stored in external Memory such as tape, disk. there is a need for some special data structure & the eg. of these data structure are m-way Search Tree, B-Tree, B+ Tree. These are used in file Indexing.

m-way Search Tree :-
m-way Search Tree is a generalised version of Binary Search tree. The goal of m-way Search tree is to minimize the accesses while retreiving the key from a file.
Tree having (m-1) keys and m children called m-way search tree.

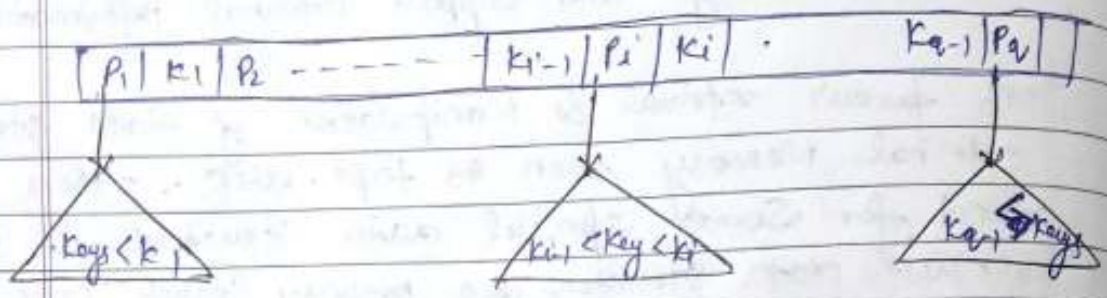
for eg. In 3-way search tree



A binary tree is a two-way tree that means it have one key in every node & have maximum two children.
An m-way tree may be a empty tree and if it is Non-empty it satisfies the following properties.

(1) All the nodes have degree less than equal to m.

(2) Each Node has the following structure.



here $K_i \rightarrow$ keys. $1 \leq i < q-1$, $q \leq m$

$P_i \rightarrow$ pointers to subtrees, where $1 \leq i \leq q$, $q \leq m$

Benefits of m-way Search tree :-

→ Fast Updation & Fast Information Retreival.

Problems -

(1) The tree is not balanced.
(2) The leaf nodes are on different labels.
(3) More bad space uses because can become skew

Qu - Construct a 3-way search tree out of an empty search tree with the following keys in the order

D, K, P, V, A, G, W, X

In 3-way search tree, key = 2, at most child = 3

(1) [D]

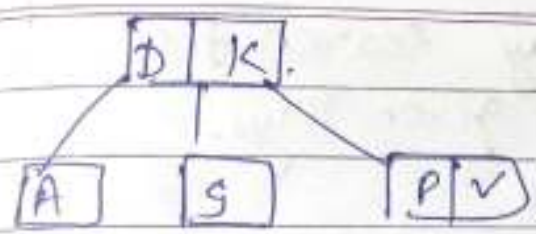
(2) [D | K]

(3) [D | K] → [P]

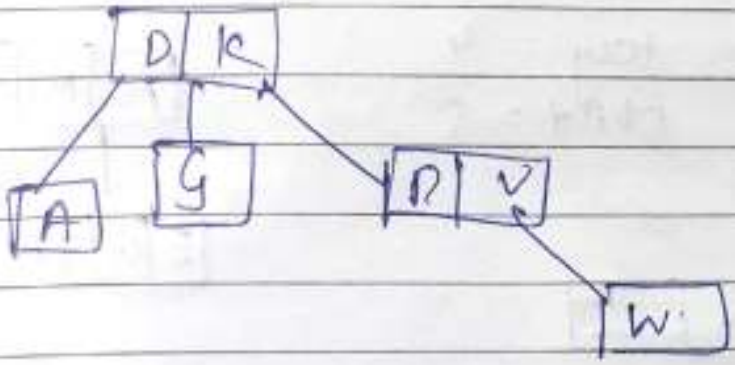
(4) [D | K] → [P | V]

(5) [D | K] → [A] [P | V]

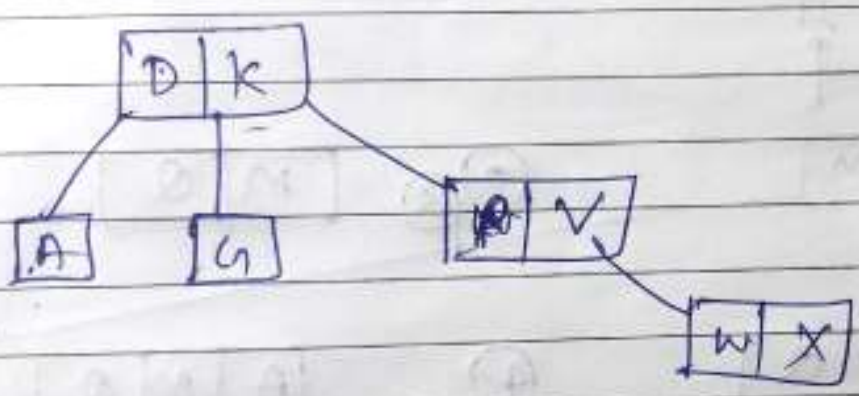
(6)



(7)

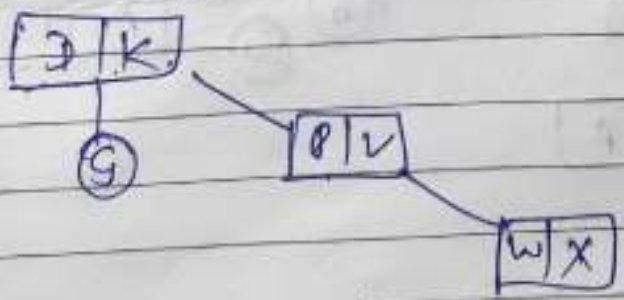


(8)

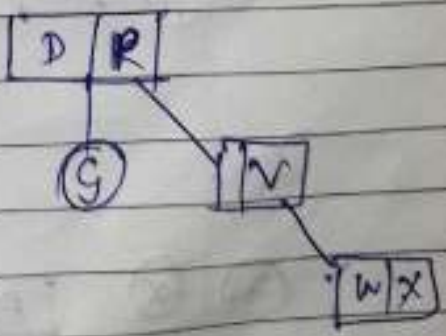


Now delete : A , K

→ delete (A)



→ delete K.



Complexity = $o(h)$ or $o(\log n)$

Date / / Page No. Shivalal

(Balance) Tree

B-Tree :-

A B-Tree of order m is an m-way search tree in which

1. The root has at least two child & at most m-child.
2. The internal node except the root node have at least $\lceil \frac{m}{2} \rceil$ child nodes & at-most m-nodes.
3. The No. of keys in each internal node is one less than the No. of child Nodes. & these keys partition the keys in the subtree of a Node in a manner similar to the m-way search tree.
4. All leaf Node are on the same level.

Eg. Construct a B-Tree of order-3, for the following

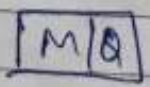
M, Q, A, N, P, W, X, T, G, E, J

order = 3

Here, Max = m-1 = 2.

Min = $\lceil \frac{m}{2} \rceil - 1 = 1$

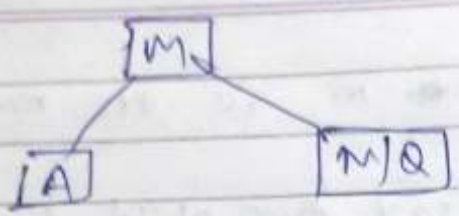
1. 

2. 

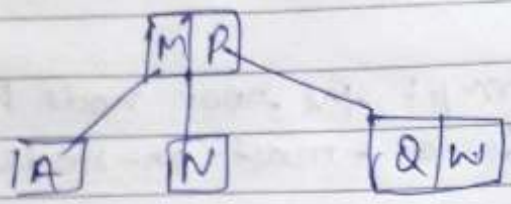
3. 

Here, child = Node-1 [illegible]
Insertion at leaf Node
Middle One split

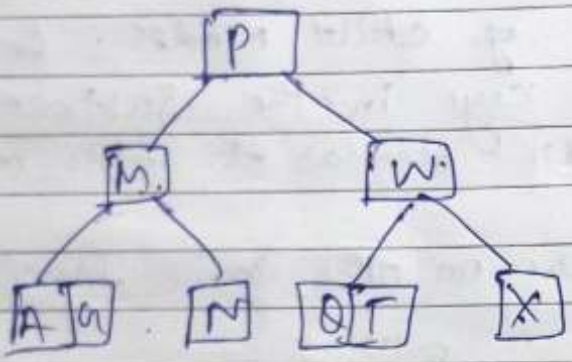
④



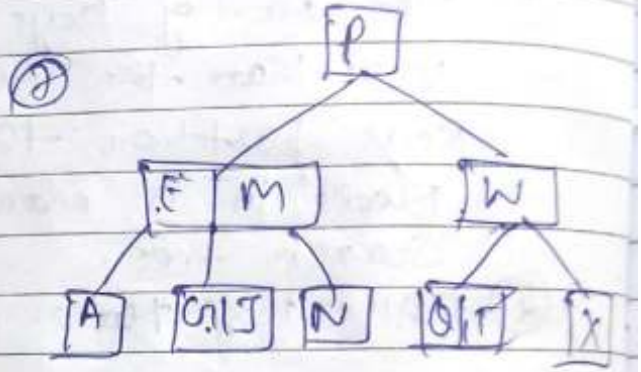
⑤



⑥



⑦

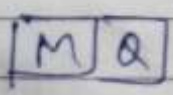


Order : 5

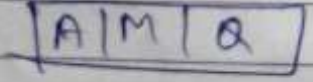
$\text{Min} = \left\lceil \frac{m}{2} \right\rceil - 1 = 2$

$\text{Max} = m - 1 = 4$

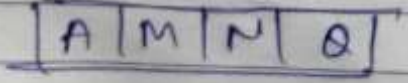
①



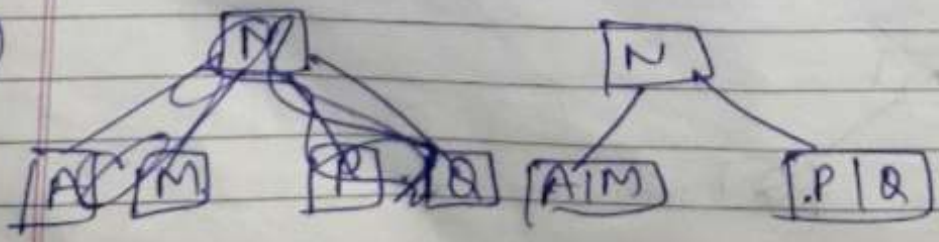
②



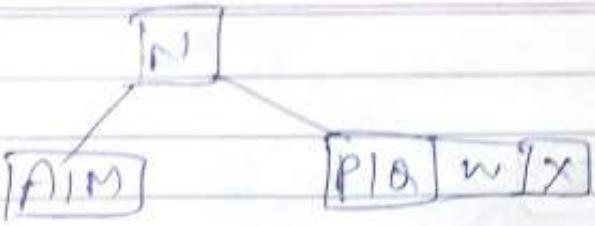
③



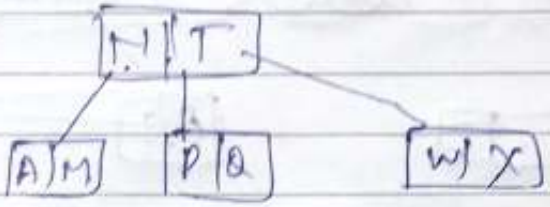
④



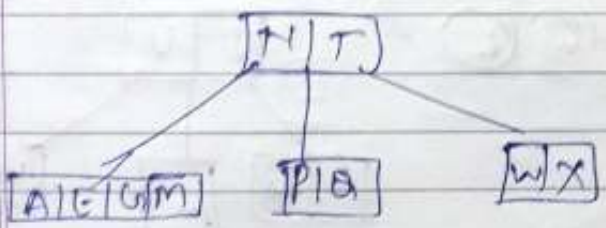
(5)



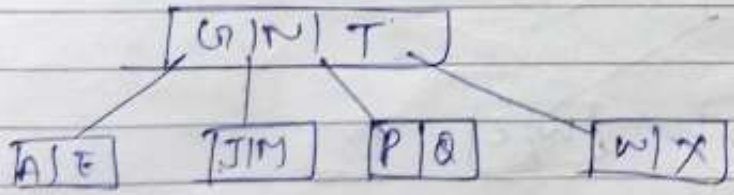
(5)



(6)



(7)



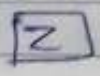
Q.P Construct a B-Tree of order - 3

Z, U, A, J, W, L, P, X, C, J, D, M, T, B, Q, E, H, S, K, N, R, G, Y, F, O, V

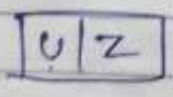
Min = 1

Max = 2.

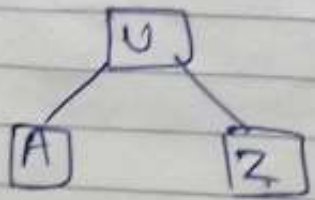
(1)

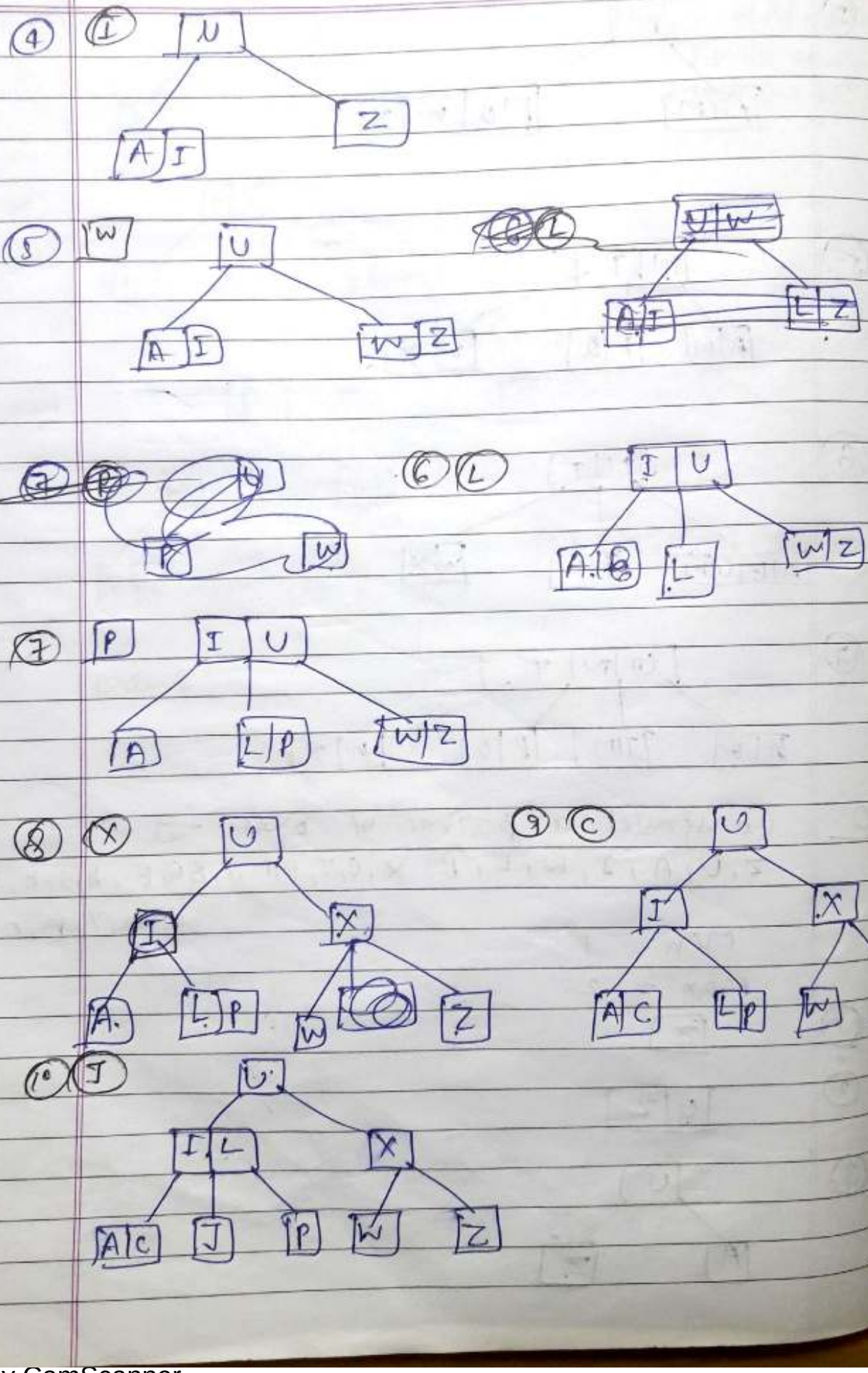


(2)



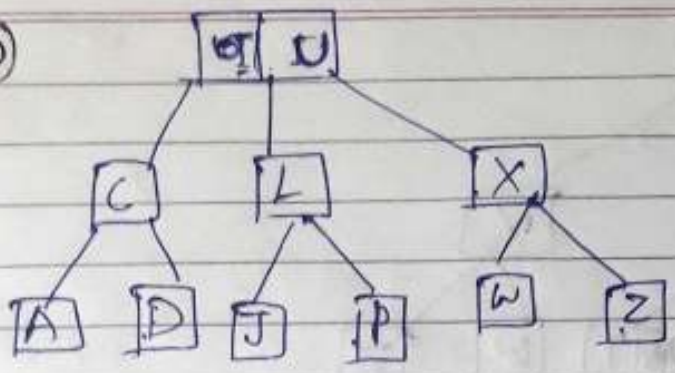
(3)



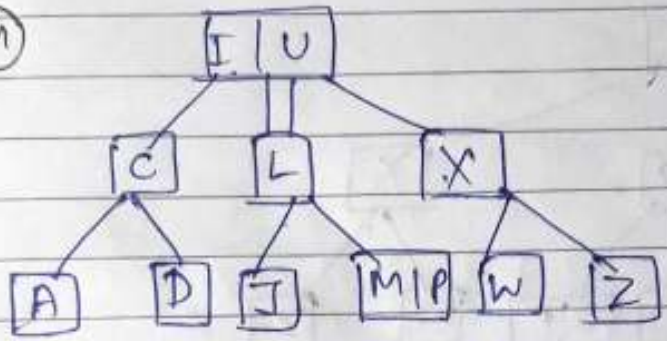


Date / /
Page No.
Shivalal

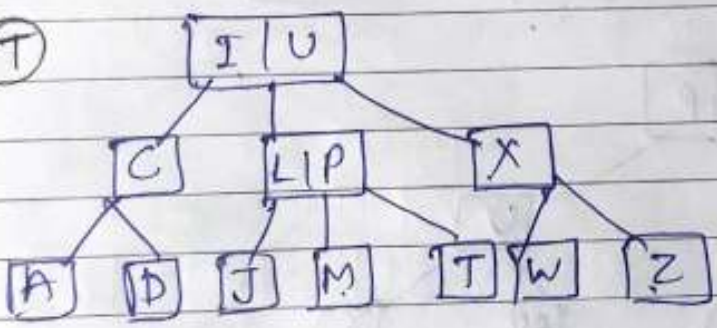
(11) (D)



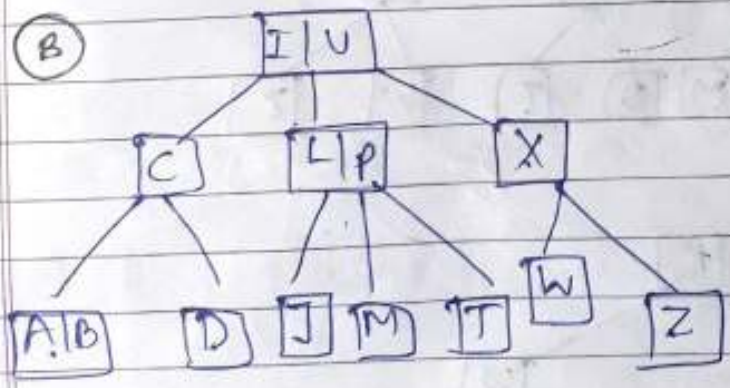
(12) (M)



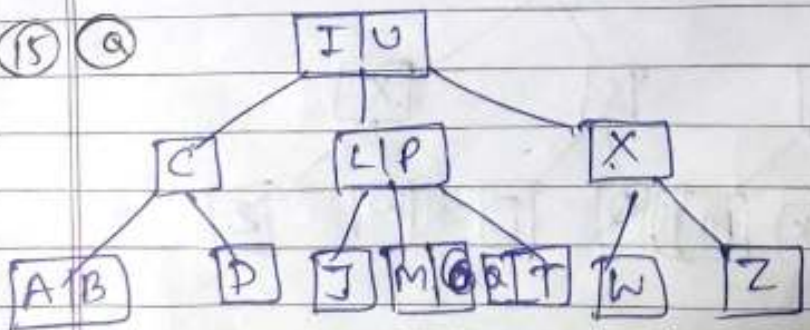
(13) (T)



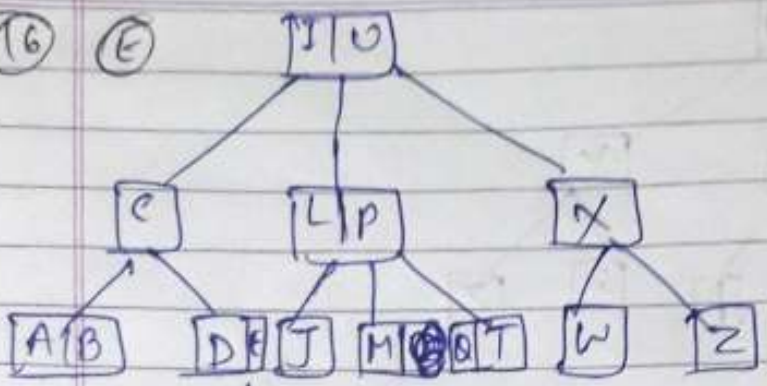
(14) (B)



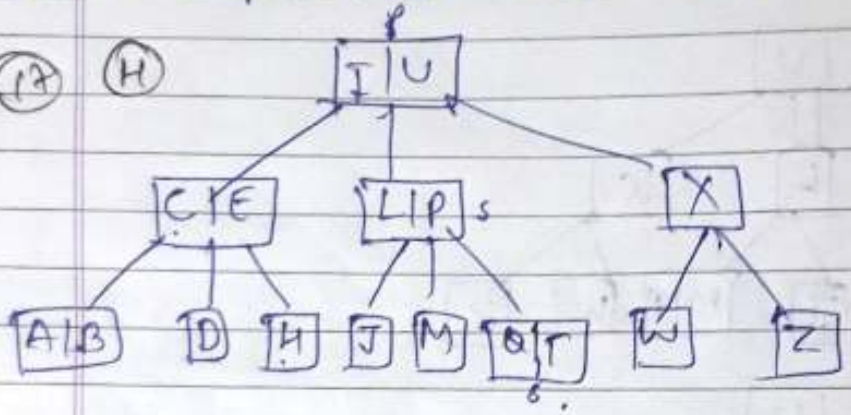
(15) (Q)



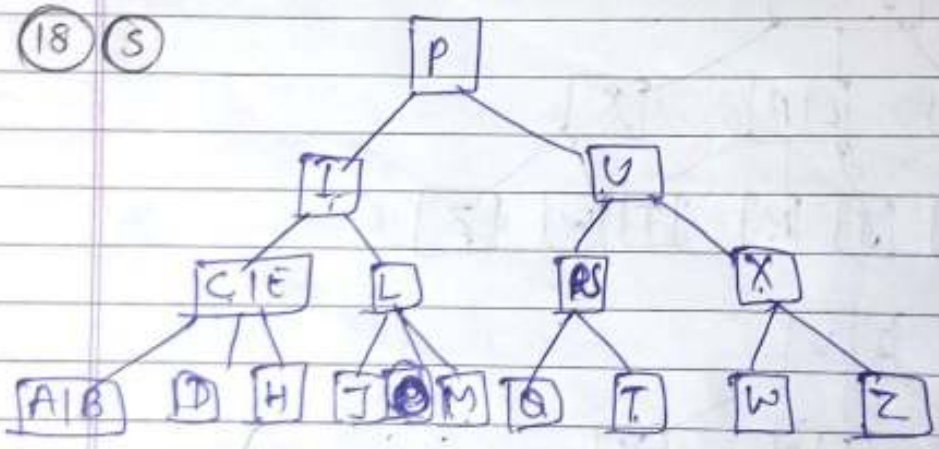
(16) (E)



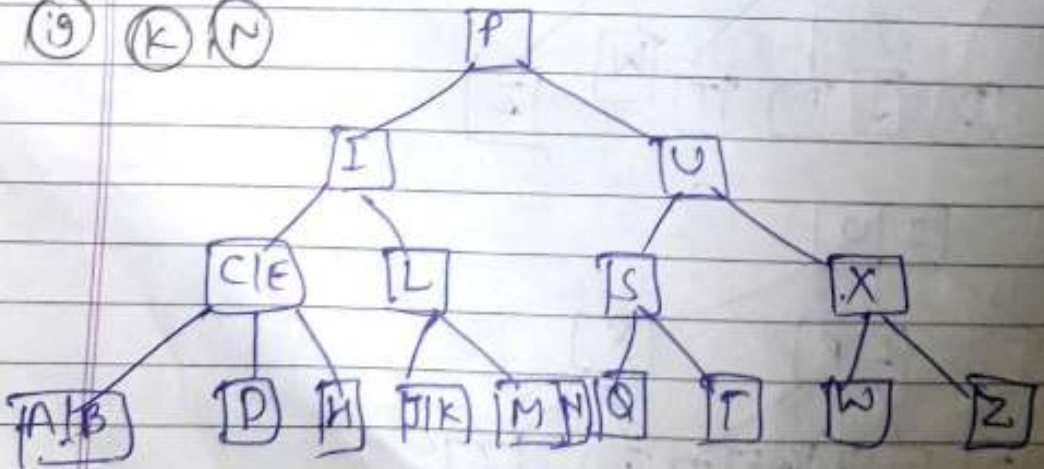
(17) (H)



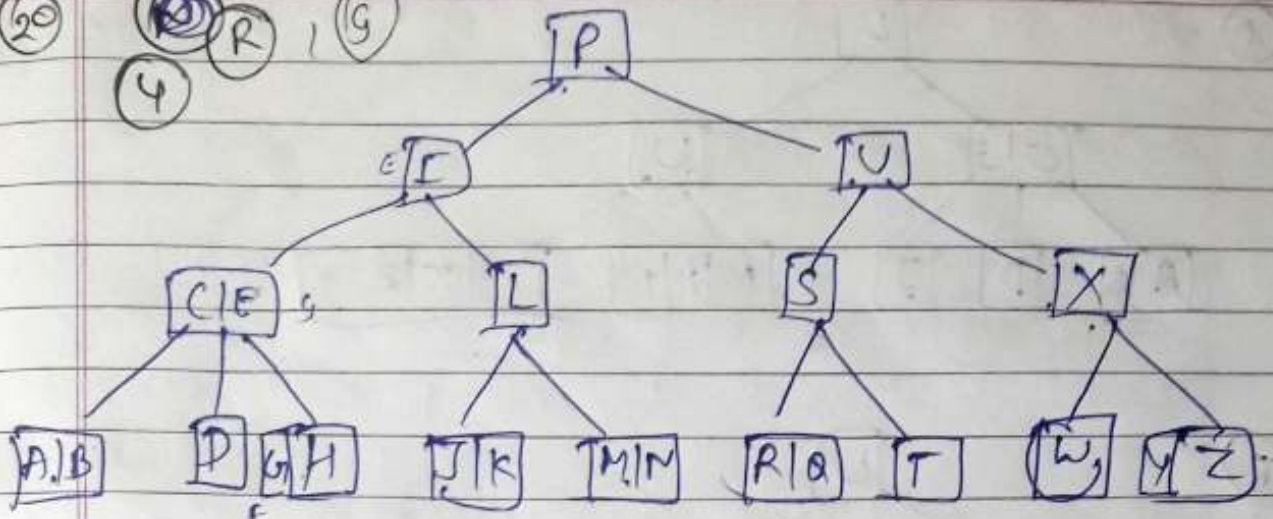
(18) (S)



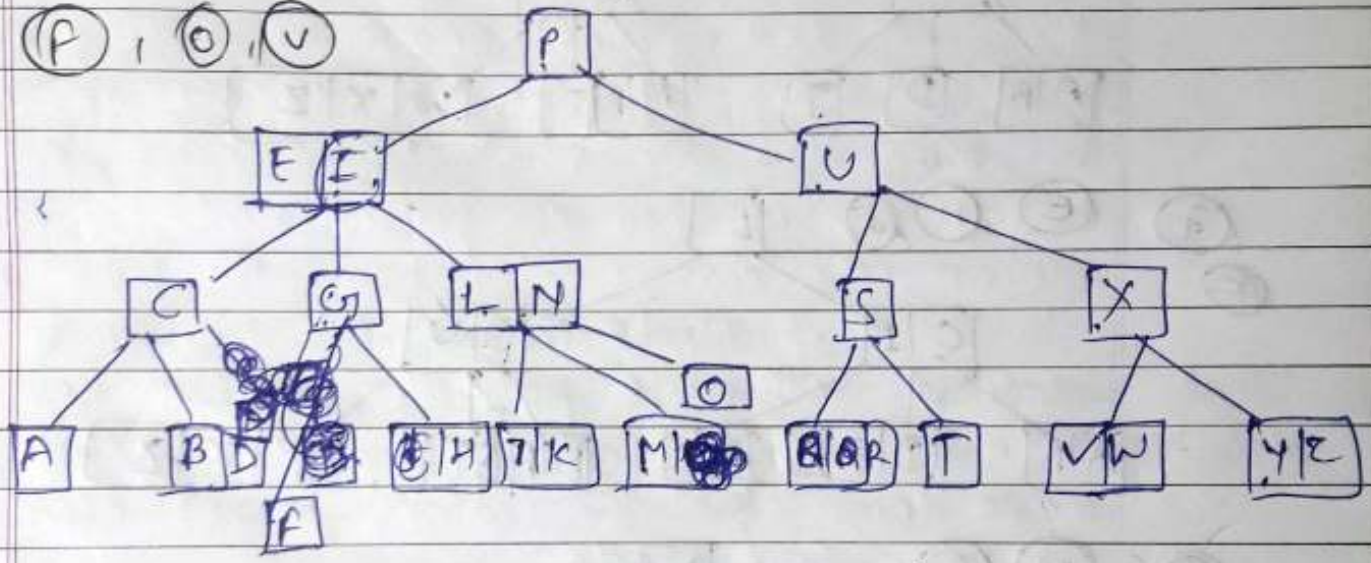
(19) (K) (N)



(20) (R), (G), (Y)



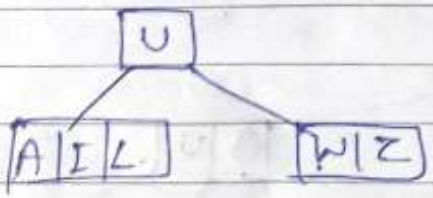
(21) (F), (O), (V)



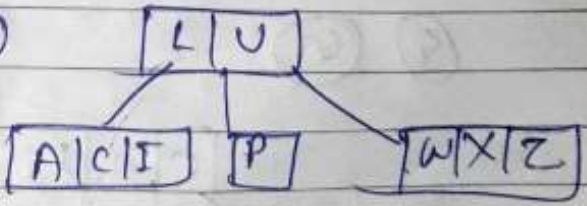
Ans By order-4. Min = 1
Max = 3

step (1) ~~A I U Z~~ A U Z

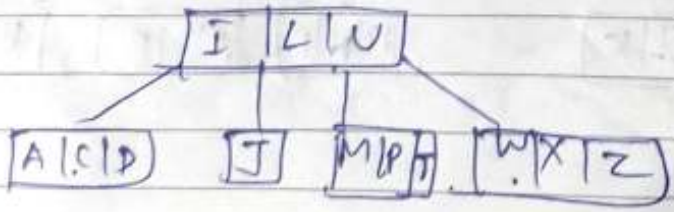
(2)



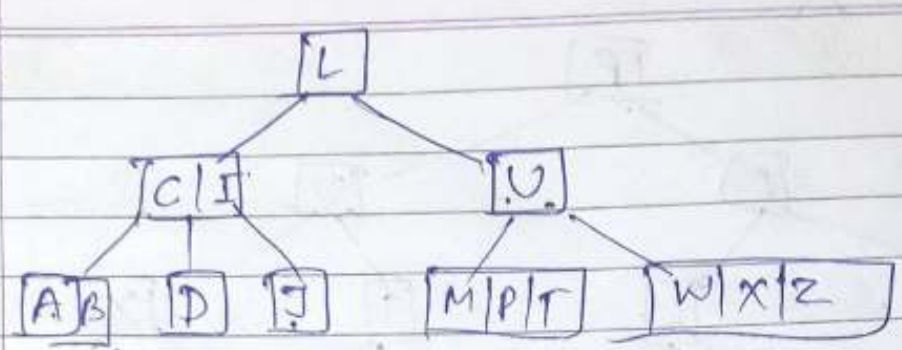
(3)



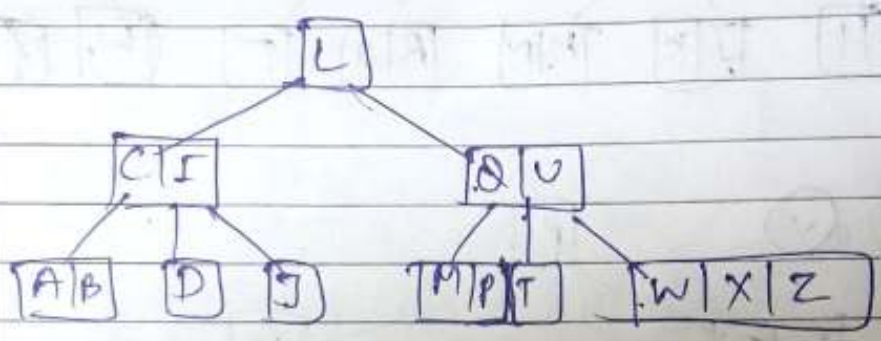
(4)



(5)

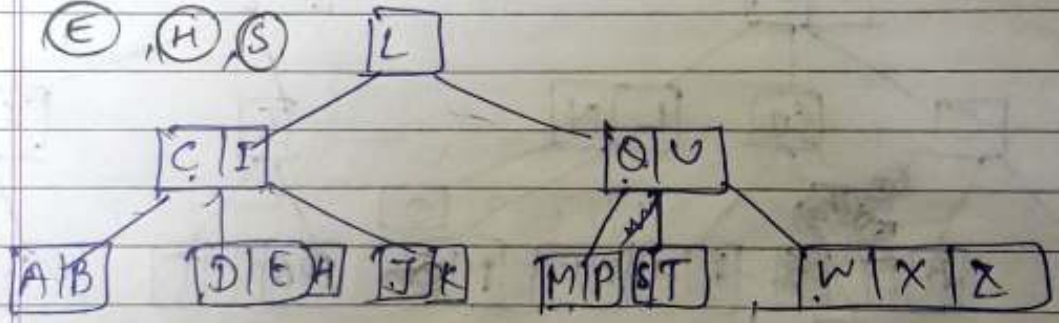


(6)

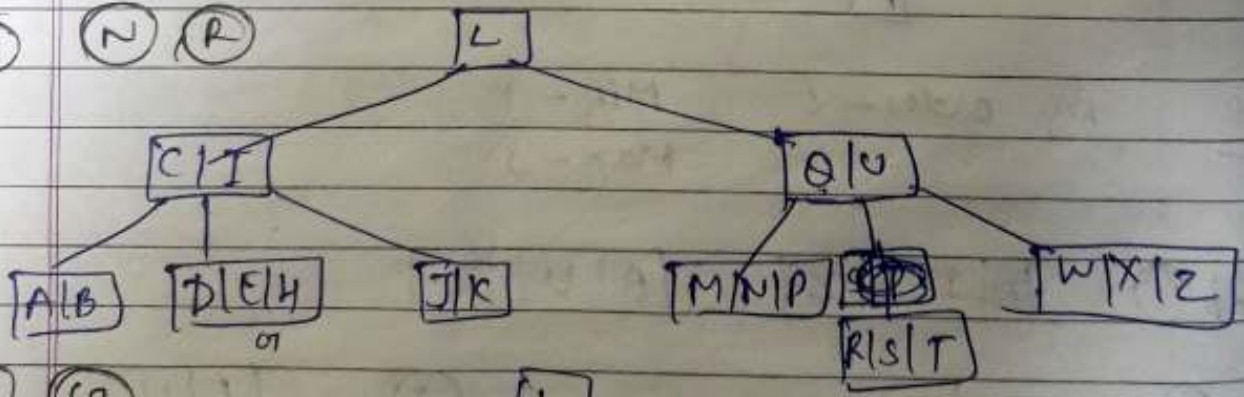


(7) (E), (H), (S)

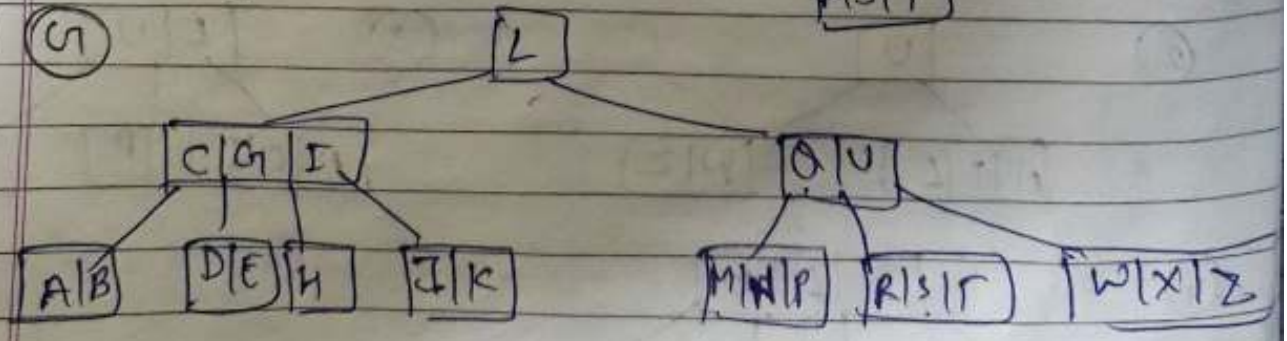
(E)



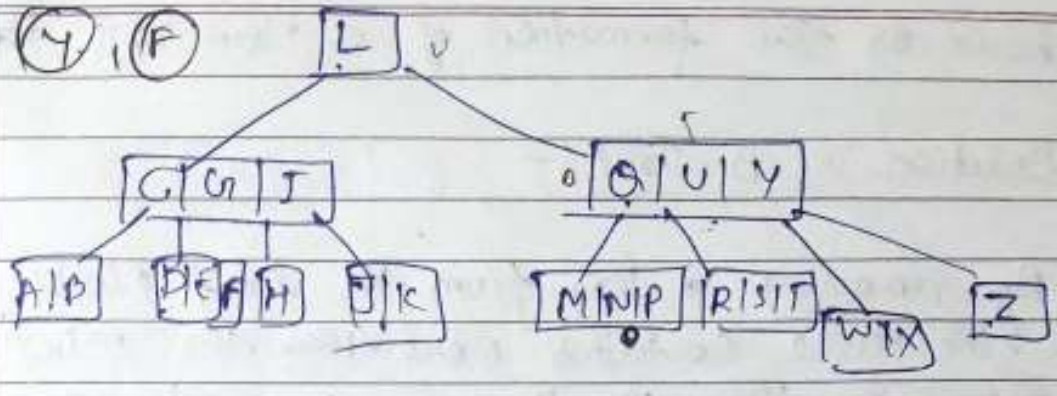
(8) (N) (R)



(9) (G)



(4) (4), (F)



(5) (O), (V)

Insertion in ~~Binary~~ B-Tree:-
The insertion of a key in B-Tree proceeds if one-word searching for the key in the tree when search terminates at the leaf node the key is inserted acc. to the following procedure:-

(1) if the leaf node in which the key is to be inserted is not full than the insertion is done in the node. A node is said to be full if it contains max^m keys. $(m-1)$.

(2) If the node is full than insert the key in order into the existing set of ~~tree~~ keys in the node, split the node at ~~at~~ its median into two nodes at the same level & pushing the median element up by one level. Insert the median element in the parent node if it is not full otherwise repeat the same procedure for rearrangement of keys in the root

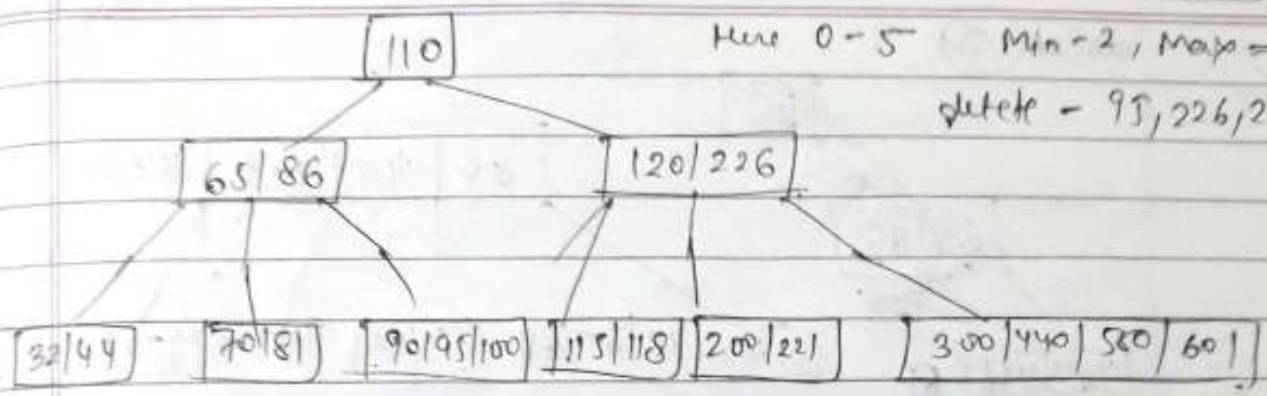
node or the formation of a new root itself.

Deletion in B-Tree :-

Step-① By removing a key from the leaf node, if the node contains more than the Min^m No. of elements than the key can be easily removed.

Step-② If the leaf node contains just the Min^m No. of elements than searching for an element from either the left sibling node OR Right sibling node to fill the vacancy. if the left sibling node has more than the Min^m No. of keys, pull the largest key up into the parent node & move down the current entry from the parent node to the leaf node where the key is to be deleted. otherwise pull the smallest key of the right sibling node to the parent node and move down the the current parent element to the leaf node.

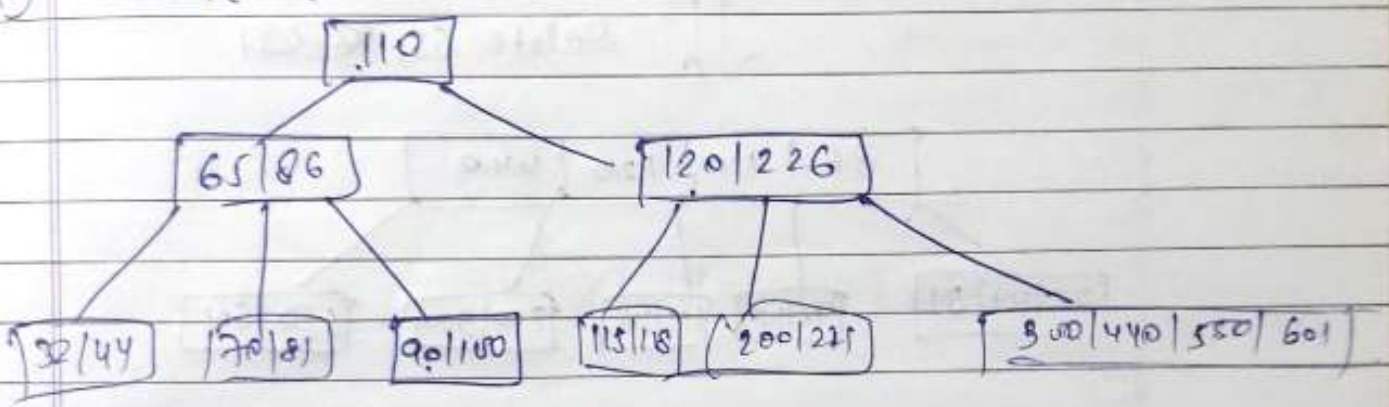
Step-③ If both the sibling node have only Min^m No of entries than create a new leaf node out of the two leaf node and the current element of the parent node but ensure that the total number does not exceed the Max limit for a node. while borrowing the node from the parent node it leaves the No. of keys in the parent node to be below the Min^m Number & then we propagate the process resulting in the reduction of height of B-



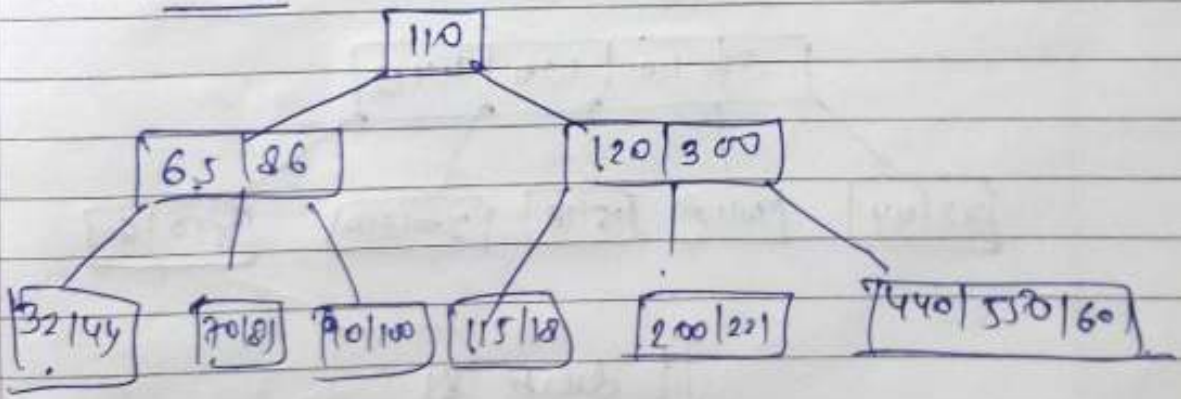
Here 0 - 5 Min = 2, Max = 4

delete - 95, 226, 221, 70

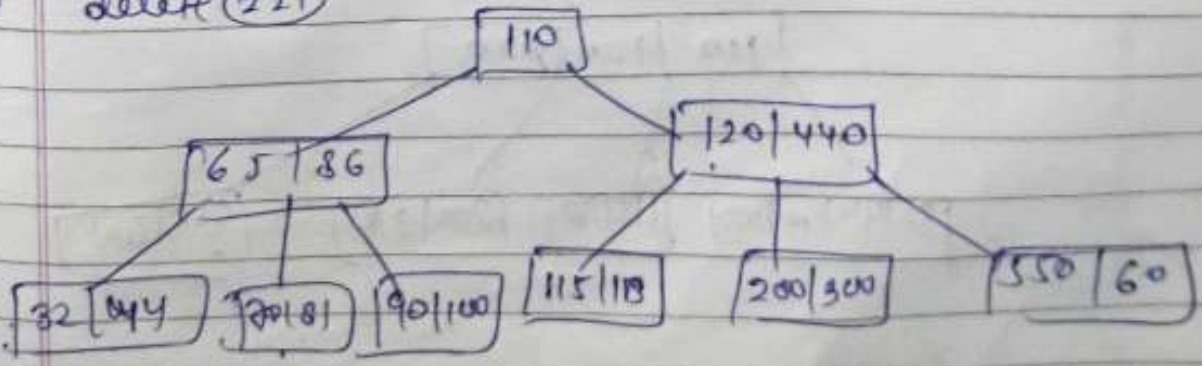
(1) delete - 95

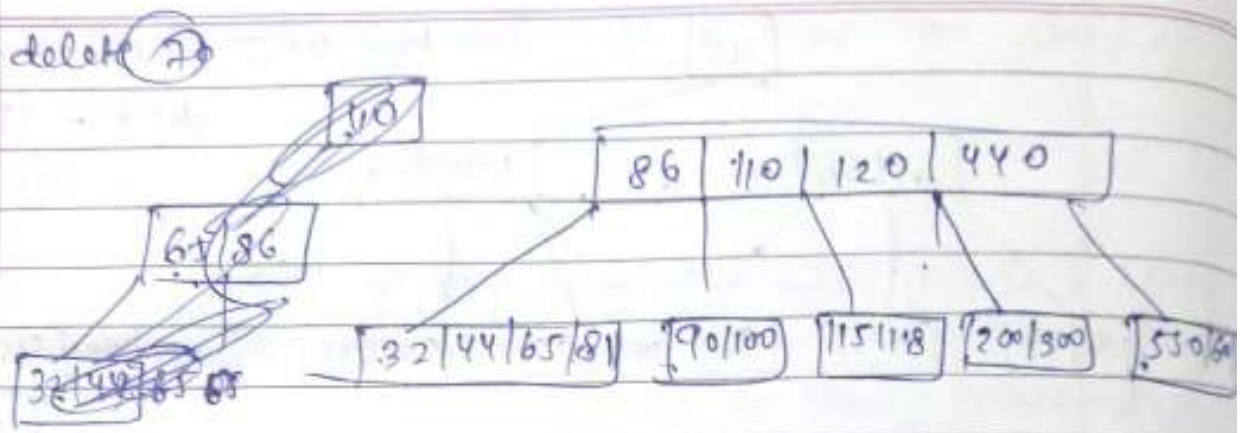


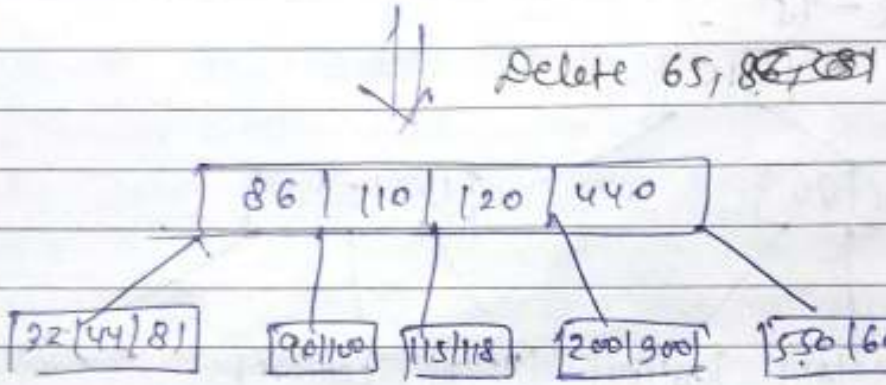
(2) delete 226

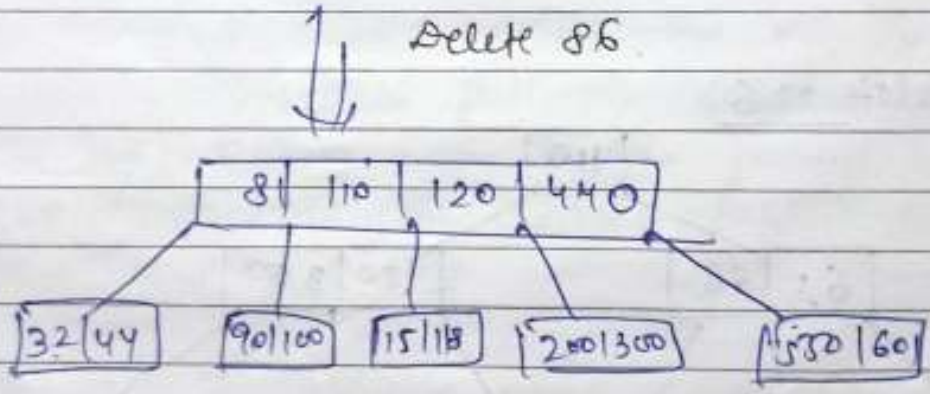


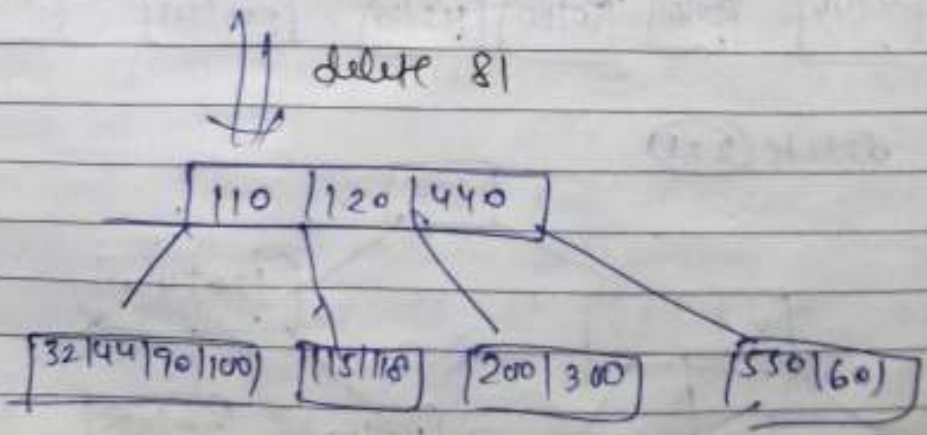
(3) delete (221)

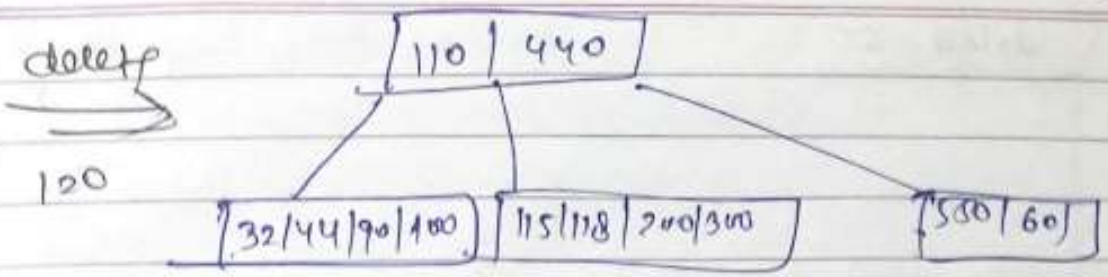






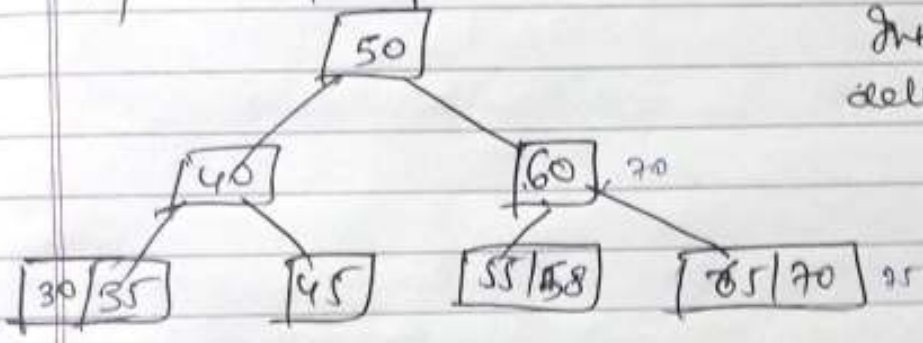




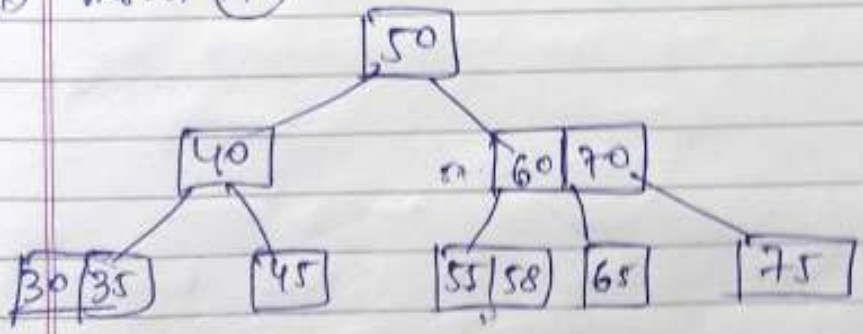


Q. Consider the B-Tree of O(3) & perform the following operation on tree.

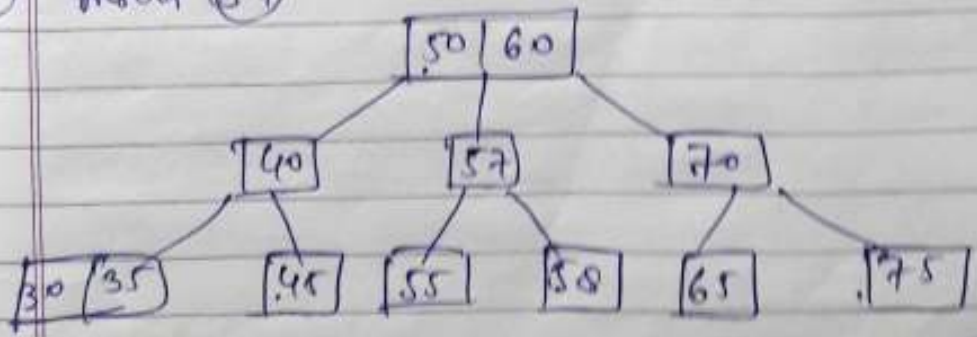
Insert – 75, 57
delete – 35, 65



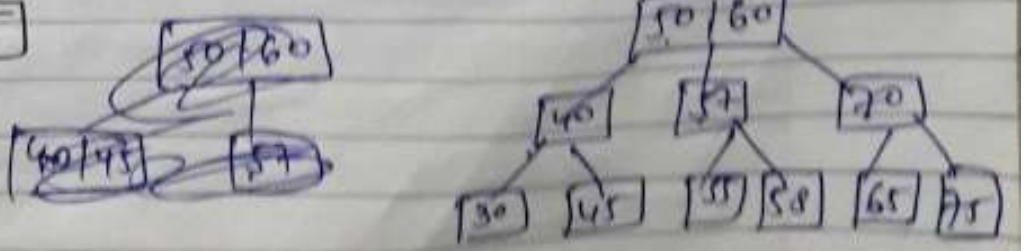
① Insert (75)



② Insert (57)



③ delete 35



Graph

Unit - IV

Graph =>

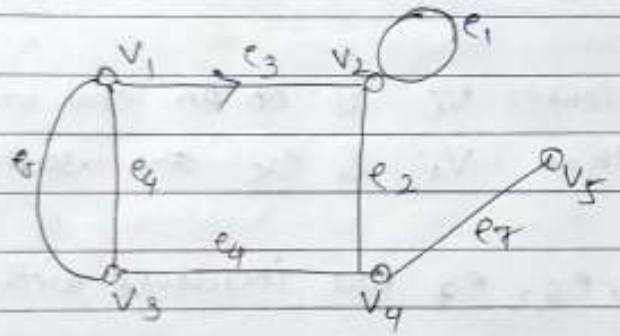
The graph $G = (V, E)$ consists of set of objects

$V = \{v_1, v_2, v_3 ----\}$ called vertices

$E = \{e_1, e_2, ----\}$ called edges

and each edge is identified with an unordered pair (v_i, v_j) of vertices.

eg.



$V = \{v_1, v_2, v_3, v_4, v_5\}$

$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$

$e_3 : (v_1, v_2)$ or (v_2, v_1)

$e_3 : (v_1, v_2)$ if direction is specified.

Self loop =>

An edge having the same vertex at both its end vertices is called Self loop.

eg. e_1

Parallel edges =>

if more than one edge in a graph are associated with a given vertex pair is called parallel edges.

eg - e_4 & e_5 are parallel edge

Date / /
Page No.
Shivalal

Note⇒ A graph that has neither self loop nor parallel edges is called Simple graph.

Adjacent Vertices ⇒
Two Vertices joined with an edge are called adjacent vertices.

Eg - V_1 adjacent to V_2 & V_3.
V_5 adjacent to V_4.
V_1 & V_4 are not adjacent.

Incidence :-
When a Vertex V_i is an end vertex of some edge e_k than V_i & e_k are said incidence with each other.

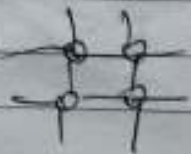
eg. e_2, e_6, e_7 are incidence with V_4.

Degree :-
The No. of edges incident on a vertex V_i, with self loops counted twice is called degree of Vertex. it is represented by $d(V_i)$.

$d(V_1) = 3$ $\quad$ $d(V_4) = 3$
$d(V_2) = 4$ $\quad$ $d(V_5) = 1$
$d(V_3) = 3$

Finite & Infinite graph :-
A graph with finite No. of Vertices as well as finite No. of edges is called finite graph. otherwise it called Infine graph.

Finite → 

 → Infinite.

Date / /
Page No.
Shivalal

Regular graph ⇒

A graph in which all vertices are of equal degree is called a regular graph.

• ⟶ 0-regular graph.

⟶ 1-regular graph

⟶ 2-regular graph

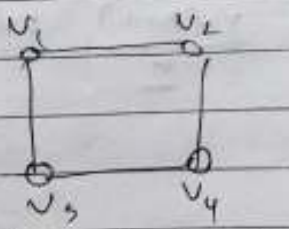
⟶ 3-regular graph.

⟶ 4-regular graph

Isolated Vertex →

A vertex having zero incident edge is called isolated vertex. means a vertex with degree zero

eg - V_5.



Pendant Vertex ⇒

A vertex of degree-1 is called pendant vertex.

eg - V_5.

Page No.

Shivalal

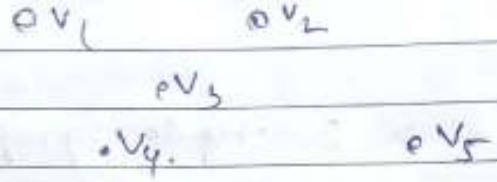
NULL graph ⇒

A graph $G = (V, E)$ is called NULL graph if the set of edges in a graph are empty.

i.e

$$\text{if } E(G) = \{\phi\}$$

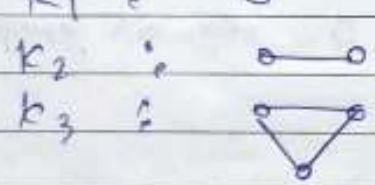
for eg.



Complete graph ⇒ / Full graph →

If in a simple graph there exist an edge b/w each & every pair of vertices then the graph is called Complete graph or full graph. it is denoted by K_p. where p is no. of vertices.

K_1 :

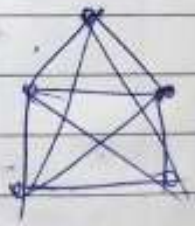


K_2 :

K_3 :

K_4 :

K_5 :

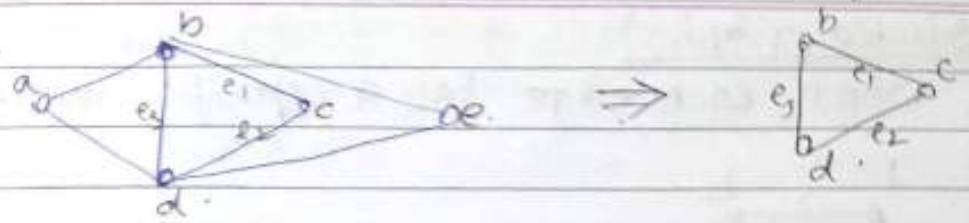


Note :- the degree of each every vertex is $n-1$. & total no. of edges are $\frac{n(n-1)}{2}$. where n is no. vertices.

Sub-graph :-

A graph g is said to be the sub-graph of graph G if all the vertices and all the edges of sub-graph in G and each edge of sub-graph has the same end vertices in G.

G



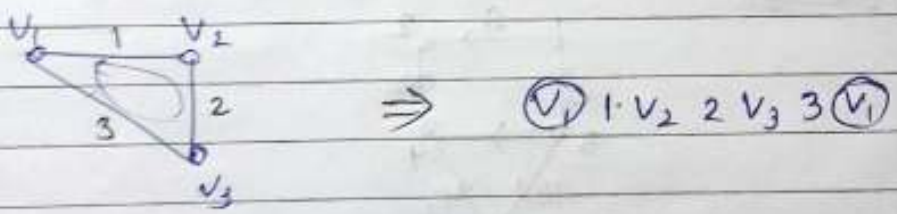
Path :-

Path is the comb^{n} of edges.

eg · P(V1, V4) ⇒ Path of V1 to V4 ⇒ V1 e3 V2 e2 V4

Cycle / Circuit ⇒

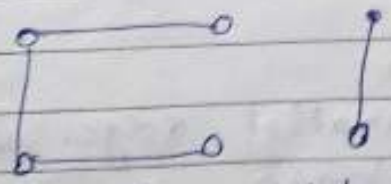
A cycle is a path in which first & last vertices are same is called cycle.



Connected graph :-

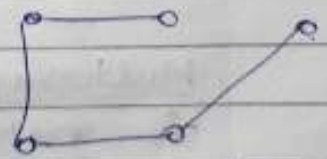
A grap is called connected if there exist a path b/w two vertices.

eg



Disconnected

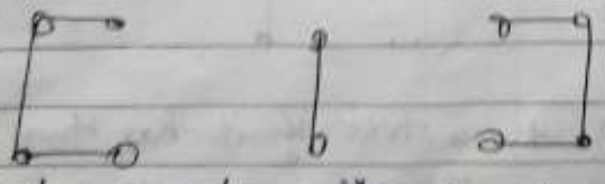
eg



Connected graph.

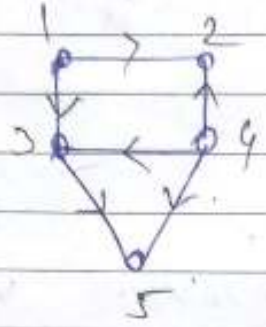
Disconnected graph ⇒ If it is called Disconnected graph with two components.

eg



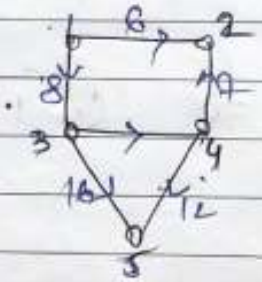
Disconnected graph with 3 - components

Directed graph ⇒
In which each edge has a specific direction.



Weighted graph ⇒
The graph is said to be weighted if every edge assign with some value.

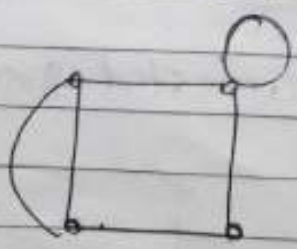
for eg



Note ⇒ Tree can be called as graph but Graph is not be tree. because Tree can't form cycle.

Multigraph :-
A graph with parallel edges & self loops is called Multigraph.

for eg-



Tree :-
A graph is a tree if it has two properties first it is Connected & there is No cycle in the graph.

for eg.

Memory Representation of graph in Memory :-

There are 2 ways of maintaining a graph in a Memory of the Computer.

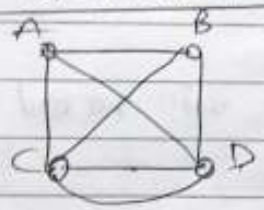
1. Sequential representation of graph – By using Adjacency Matrix
2. Linked Representation of graph – By Adjacency list.

Adjacency Matrix :-

The Adjacency Matrix for a graph with n-vertices is an n by n Matrix of words. Such that

$$a_{ij} = \begin{cases} 1 & \text{if } V_i \text{ is adjacent to } V_j \\ 0 & \text{otherwise.} \end{cases}$$

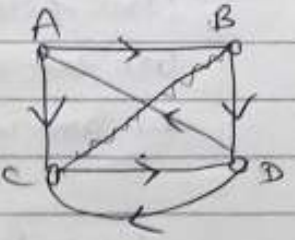
for Undirected



A =

	A	B	C	D
A	0	1	1	1
B	1	0	1	1
C	1	1	0	
D	1		1	0

for Directed.

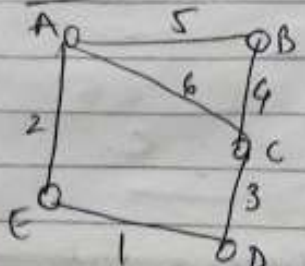


A =

	A	B	C	D
A	0	1	1	0
B	0	0	0	1
C	0	0	0	1
D	1	0	1	0

for Weighted Graph

In Undirected



A =

	A	B	C	D	E
A	0	5	6	0	2
B	5	0	4	0	0
C	6	4	0	3	0
D	0	0	3	0	1
E	2	0	0	1	0

Date / /
Page No.
Shivalal

(1) Linked list Representation of Adjacency list.

The link representation will contain two list -
① A Node list and ② edge list.

① Node List -

Each element in the list Node will be the vertex and record in the form of

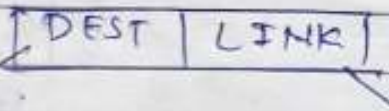
NODE will be the Name or key value of the Node

It is a pointer to the Next Node in the Node list.

ADJ will be a pointer to the first element in the Adjacency list of the Node

② Edge List -

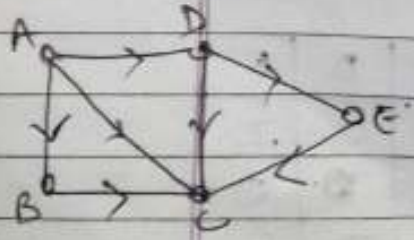
Each element in the edge list will record in the form of



The field DEST will point to the location in the Node list of the Destination

LINK field is a pointer will link together the edges with the same initial Node.

for eg -



Node list.

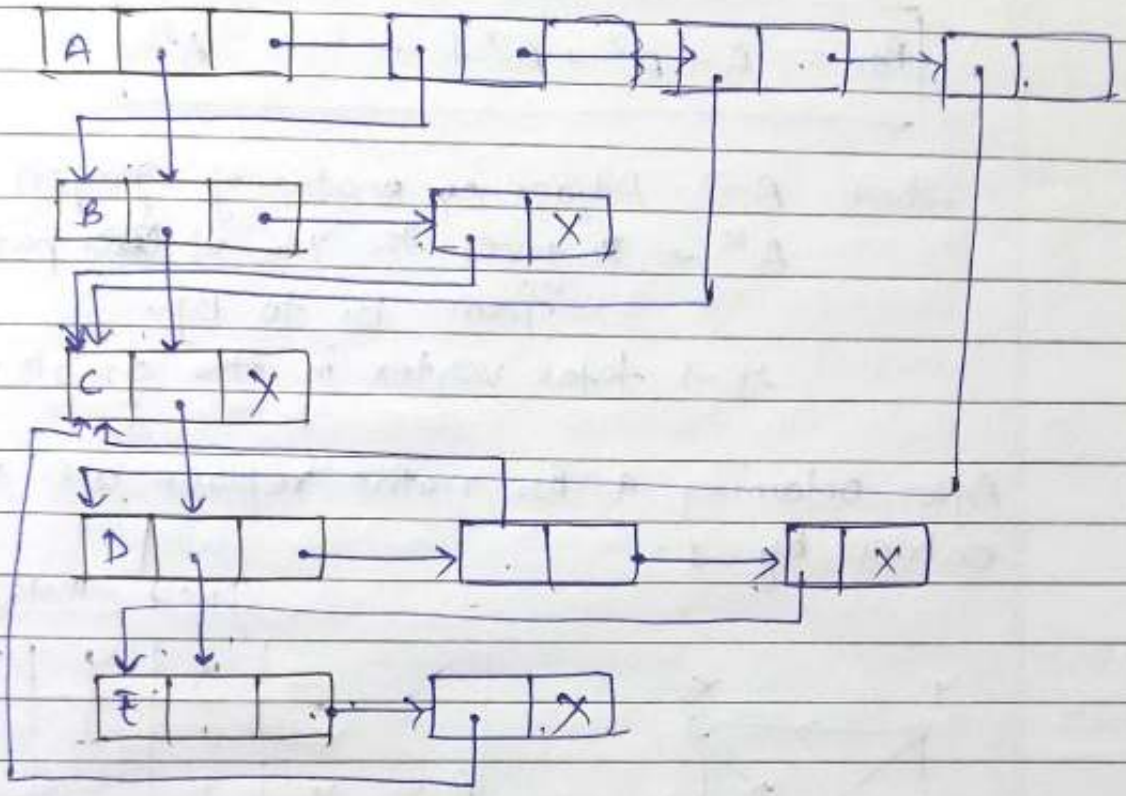
A
B
C
D
E

Edge list

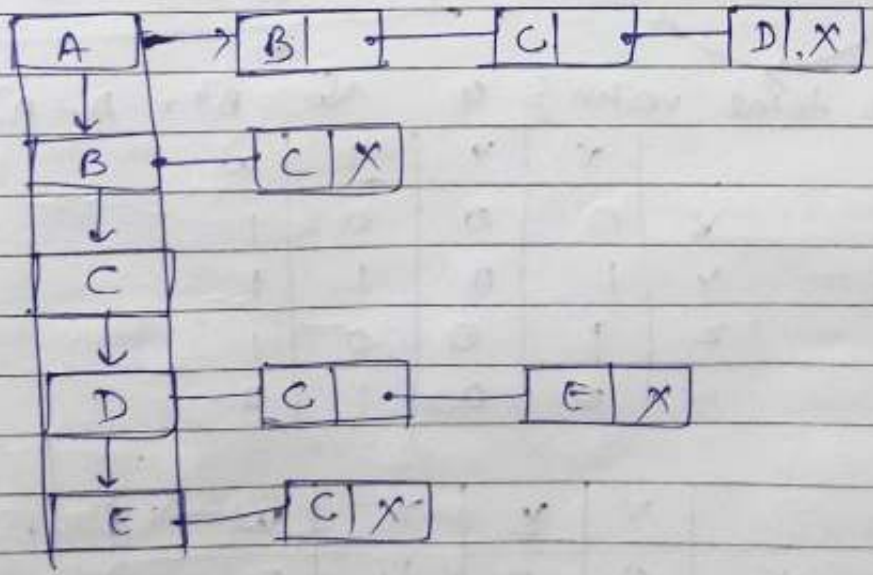
B C D
C.
~
C E
C.

Adjacency list

Memory Representation



OR



Path Matrix (Reachability Matrix) ⇒

The path Matrix of graph is the m-square Matrix defined as

$$P_{ij} = \begin{cases} 1 & \text{if there is path from } V_i \text{ to } V_j \\ 0 & \text{otherwise} \end{cases}$$

Calculation.

$$B_n = A + A^2 + A^3 + \cdots A^n$$

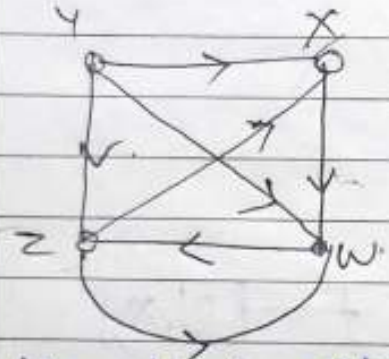
where $A \rightarrow$ Adjacency matrix of graph G.

$A^k \rightarrow$ gives the No. of path of length k from B_i to B_j.

$n \rightarrow$ total vertex in the graph.

After obtaining a B_n matrix replace all Non-Zero entries by 1.

Direct Method.



$P =$

	X	Y	Z	W
X	1	0	1	1
Y	1	0	1	1
Z	1	0	1	1
W	1	0	1	1

Here total vertex = 4 So $B_4 = A + A^2 + A^3 + A^4$

Adjacency Matrix, $A =$

	X	Y	Z	W
X	0	0	0	1
Y	1	0	1	1
Z	1	0	0	1
W	0	0	1	0

path length should be 2

$A^2 =$

	X	Y	Z	W
X	0	0	1	0
Y	1	0	1	2
Z	0	0	1	1
W	1	0	0	1

Date / /
Page No.
Shivalal

Here path length should be 3

A^3 =

	X	Y	Z	W
X	1	0	0	1
Y	1	0	2	2
Z	1	0	1	1
W	0	0	1	1

A^4 =

	X	Y	Z	W
X	0	0	1	1
Y	2	0	2	3
Z	1	0	1	2
W	1	0	1	1

B^4 =

	X	Y	Z	W
X	1	0	2	3
Y	5	0	6	8
Z	3	0	3	5
W	2	0	3	3

Note:- A graph is called Strongly connected if and only if the path Matrix of the graph has No-Zero entries.

Note:- Reachability Matrix is also known as transitive closure.

Traversing of Graph:-

There are two methods for traversing a graph.

1. Breadth first Search (BFS) : Queue
2. Depth first Search (DFS) : Stack

During the execution of algorithm each Node N of the graph will be in one of the three states called the status of N.

i) Ready state → It is the initial state of Node N.

ii) Waiting state → The Node N is on the queue or stack waiting to be processed

iii) Processed state → The Node N has been processed.

Page No.

Shivalal

Algorithm for BFS :-

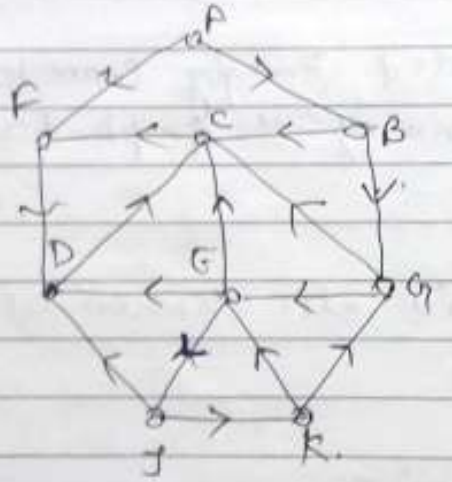
Steps - ① Initialize all nodes to the Ready state.

② Put the Starting node A in the queque and change its status to the waiting state.

③ Repeat step 4 & 5 until queque is empty.

④ Remove the front node N of the queque. Process N and change the status of N to the processed state.

⑤ Add to the rear of ~~queue~~ queue, all the neighbours of N, that are in the ready state & change their status to the waiting state.



Node list :	Edge list
A	F, B
B	C, G
C	F
D	C
E	C, D, J
F	D.
G	C, E.
J	D, K
K.	E G

Adjacency list.

Using BFS

(A)	F=0, R=0	
(B) F	F=1, R=2	
(F) C, G	F=2 R=4	i.e. the Seavence in which N is ~~stored~~ in queque traverse.
(C) G, D	F=3, R=5	
(G) D	F=4, R=5	=
(D) E	F=5, R=6	A B F C G D E J K.
(E)	F=6 R=6	
(J)	F=7 R=7	By BFS the Node will
(K)	F=8 R=8.	traverse in this order

Date / /
Page No.
Shivalal

Q. What will be the path to reach J using BFS from A.?

Ans.

A	B	F	C	G	D	E	J	K
∅	A	A	B	B	E	G	E	

Origin.

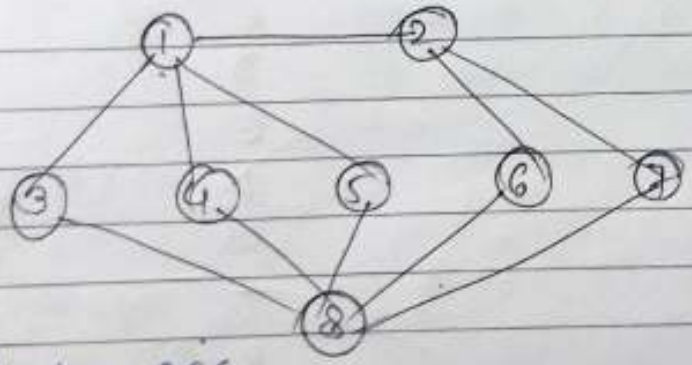
Then check.

J is generated from E
E is generated from G
G ——— B
B ——— A

E ← J
G ← E
B ← G
A ← B

So path will be A → B → G → E → J

Q. Find traversing through BFS



Node List	Edge List
1	2, 3, 4, 5
2	1, 6, 7,
3	1, 8
4	1, 8
5	1, 8
6	2, 8
7	2, 8
8	3, 4, 5, 6, 7

Using BFS,

(1)	F = 0	R = 0
(2), 3, 4, 5		F = 1, R = 4
(3), 4, 5, 1, 6, 7		F = 2, R = 6
(4), 5, 1, 6, 7, 8		F = 3, R = 8 7
(5), 1, 6, 7, 8		F = 4, R = 8 7
~~1, 6, 7, 8~~		F = 5, R = 8
(6), 7, 8		F = 6, R = 8 7
(7), 8		F = 7, R = 8 7
(8)		F = 8, R = 8

1	2	3	4	5	6	7	8

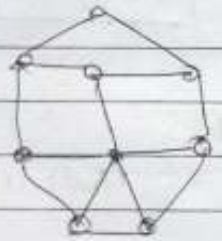
Date / /
Page No.
Shivalal

Algorithm for DFS

Qp

1. Initialize all the nodes to the Ready state.
2. Push the starting Node A into the stack & change its state to waiting.
3. Repeat step 4 & 5 until stack is empty.
4. Pop the top Node N from the stack & process N & change its status to the processed state.
5. push into the stack all the neighbours of N that are in the ready state & change their status to the waiting state.

Q. Refer prev ques.



Adjacency list

Node list	Edge list
A	B, F
B	C, G
C	F
D	C
E	C, D, J
F	D
G	C, E
J	D, K
K	E, G

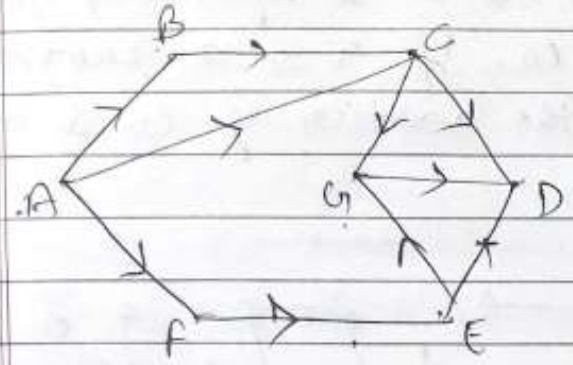
Using DFS

(A)
B (F)
B (D)
B (C)
(B) (G)
(G)
(E)
(J)
(K).

DFS: A F D C B G E J K.

Date / /
Page No.
Shivalal

Q) Show all steps & traverse the graph with BFS & DFS starting from A.



Node list	Edge list
A	B C F
B	C
C	D; G
D	-
E	D, G
F	E
G	D

By BFS

(A)	F = 0, R = 0
(B), C, F.	F = 1, R = 2
(C) F,	F = 2, R = 3
(F), D, G	F = 3, R = 5
(D), G, E	F = 4, R = 6
(G) E	F = 5, R = 6
(E)	F = 6, R = 6

A B C F D G E

By DFS

(A)
B, C, (F)
B, C, (E)
B, C, D, (G)
B, C, (D)
B, (C)
(B),

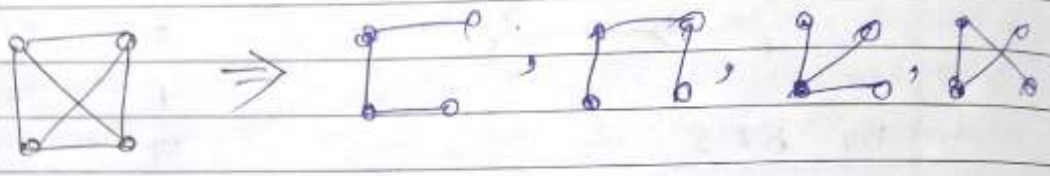
A F E G D C B

Date / /
Page No.
Shivalal

Spanning tree :-

Note-①

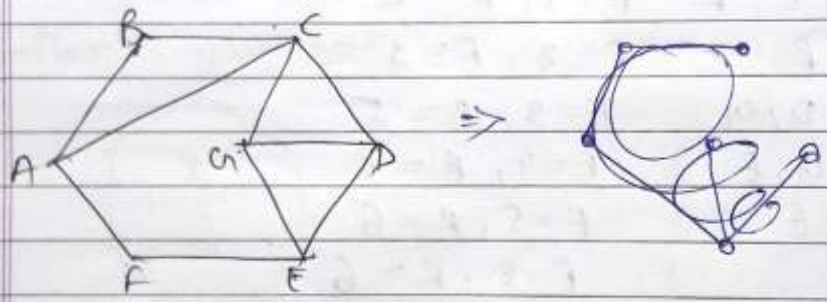
A tree T is said to be a spanning tree of a connected graph G, if T is a subgraph of G & contains all the vertices of G & does not form a cycle.



Note-2

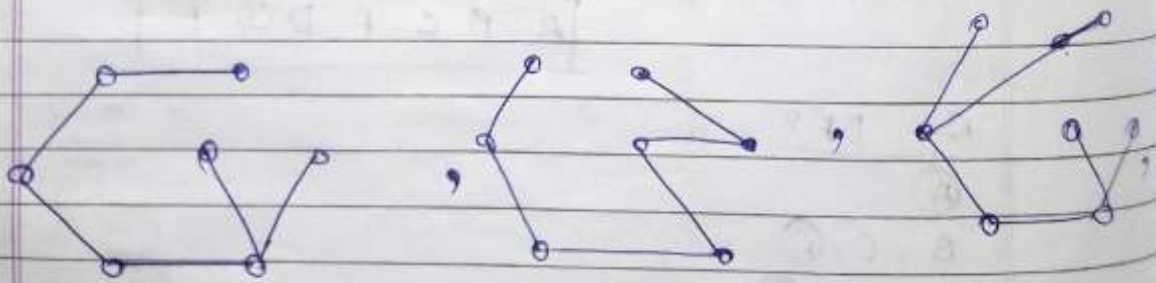
Note-3

Qy. Construct at least 5-spanning tree of the graph.

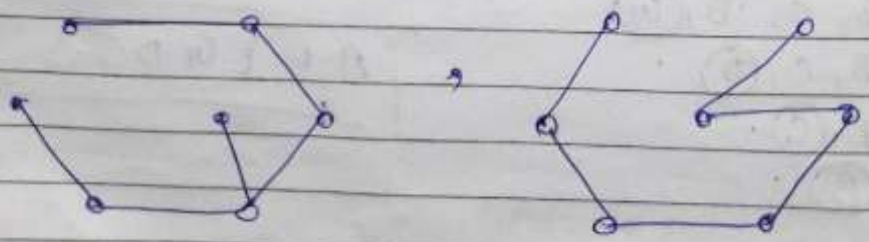


Note-4

Imp



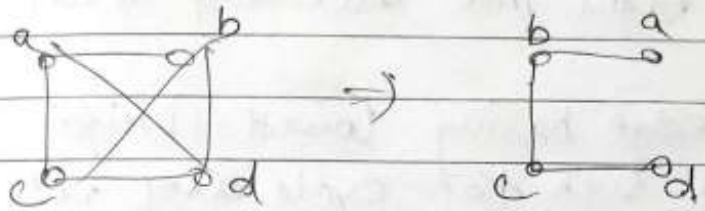
(10)



Page No.

Shivalal

Note-① An edge of the graph G which is still in its spanning tree is called branch



branch - (a,c), (a,b), (c,d)

Note-2 The edges which is not in spanning tree is called chord

chord - (b,d)

Note-3 A graph can have more than one spanning tree. & every connected graph can have at least one spanning tree.

Note-4 if a graph have m no. of edges & n no. of vertices then no. of chords are - m-(n-1) & No. of branches = n-1

Ans 7 (10)

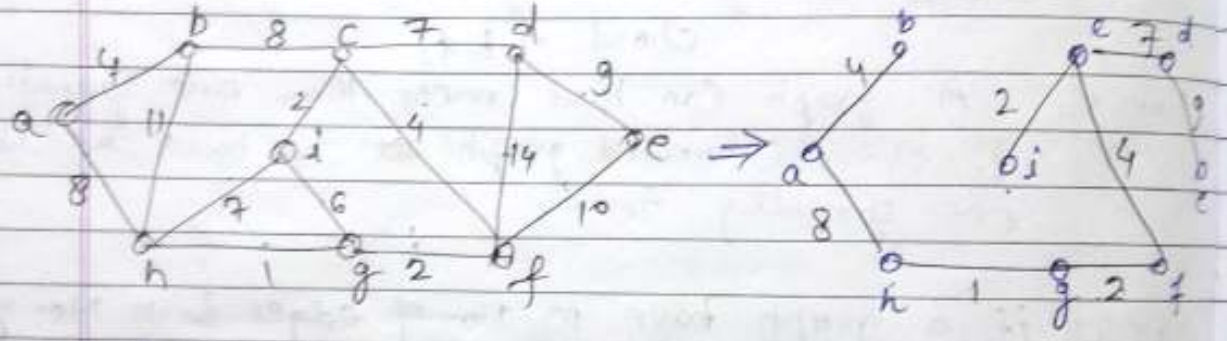
Minimum spanning tree :- (MST) ⇒

With the Application of weighted graph it is necessary to find a spanning tree for which the total weight of the edges in the tree is as small as possible. Such a spanning tree is called Minimum spanning tree or Minimal cost spanning tree.
There are two techniques to find the MST :-

① Kruskal's Algorithm
② Prim's Algorithm.

① Kruskal's Algorithm :-

Steps - ① Sort the edges into ascending order by their weights.
② Pick the edge having lowest weight from the ordered list such that cycle does not form.
③ Repeat step-2 till all the nodes become a part of MST.



Sort the tree.

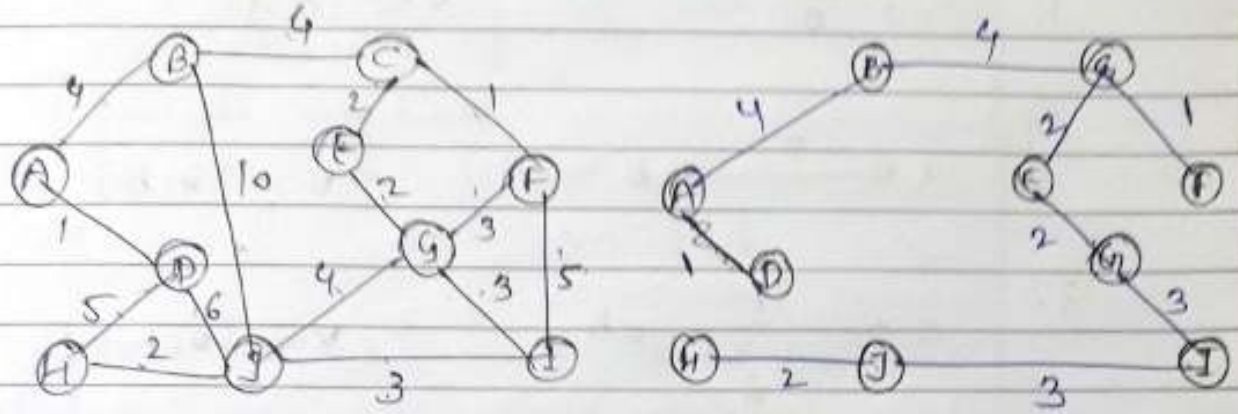
weight	edge
1	(h,g)
2	(i,c), (g,f)
4	(a,b), (f,c)
6	(i,g)
7	(h,i) (c,d)
8	(a,h) (b,c)
9	(d,e)
10	(e,f)

it forms a cycle

Total weight = 8+4+1+2+2+4+7+9
= 37 Ans

Date / /
Page No.
Shivalal

Q. Find-out the Minimal spanning tree using kruskal.



Sort the tree.

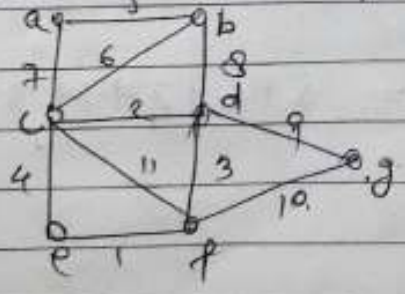
Weight	edge.
1	AD, CF,
2	HJ, CE, EG
3	JI, GI, GF — it form a cycle.
4	AB, BC, GJ.
5	HD, FI
6	DJ
10	BJ

Total weight = 1+2+3+3+2+2+1+4+4.
= 22 Ans.

(2) Prim's Algorithm ⇒

In this Algorithm we can start from any vertex U, if U contains only one path than select that path because we have no option and if you contains more than one path than we select a path having least weight. but remember that it does not form a cycle.

Q. Construct Min^m spanning tree.



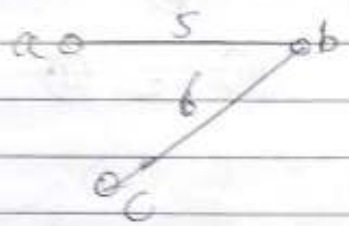
Date / /
Page No.
Shivalal

1.

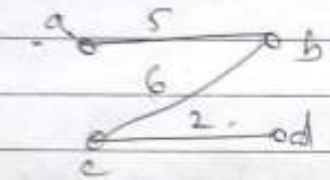
V = {a}

2. 

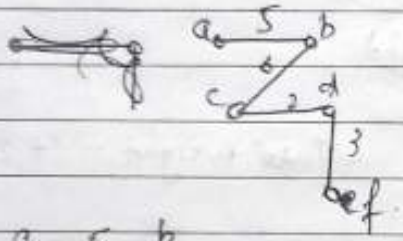
V = {a, b}

3. 

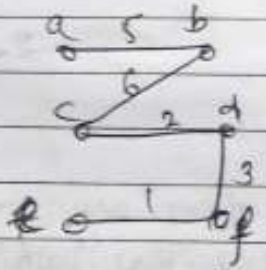
V = {a, b, c}

4. 

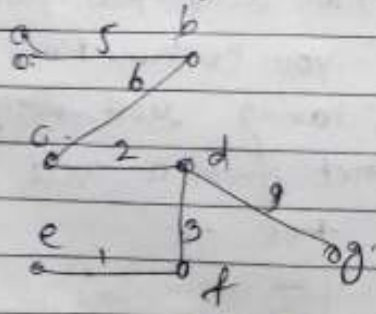
V = {a, b, c, d}

5. 

V = {a, b, c, d, f}

6. 

V = {a, b, c, d, f, e}

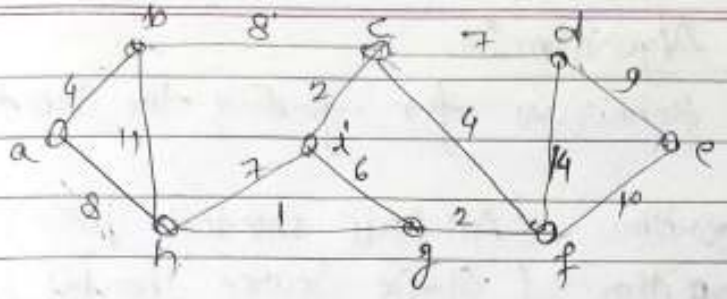
7. 

V = {a, b, c, d, e, f, g}

Total weight = 5+6+2+1+3+9 = 26

Date / /
Page No.
Shivalal

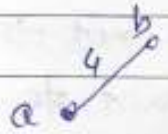
Q.1



1. a

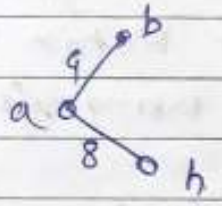
$V = \{a\}$

2.



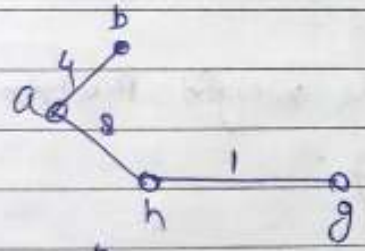
$V = \{a, b\}$

3.



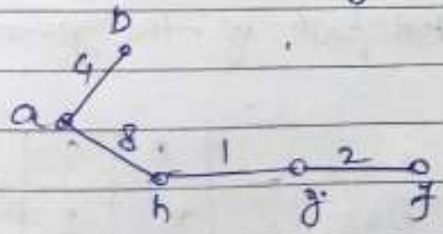
$V = \{a, b, h\}$

4.



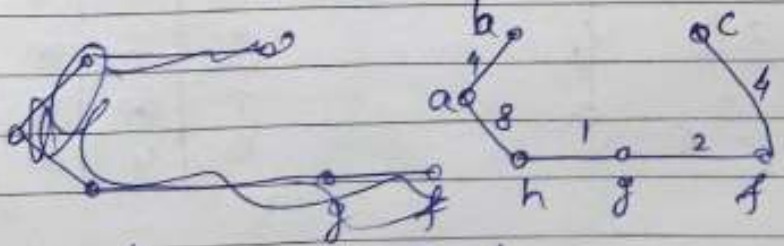
$V = \{a, b, h, g\}$

5.



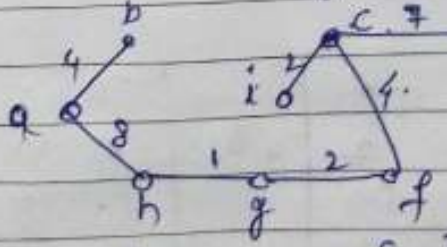
$V = \{a, b, h, g, f\}$

6.



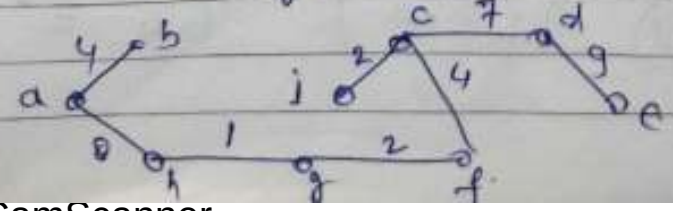
$V = \{a, b, h, g, f, c\}$

7.



$V = \{a, b, h, g, f, c, d\}$

8.



Total weight = 4+8+1+2+2+4+7+9
= 37

Page No.

Shivalal

Shortest path Algorithm :-

There are two techniques for finding the shortest path

1) Warshall Algorithm (All pair shortest path)
2) Dijkstra's Algorithm. (Single Source shortest path)

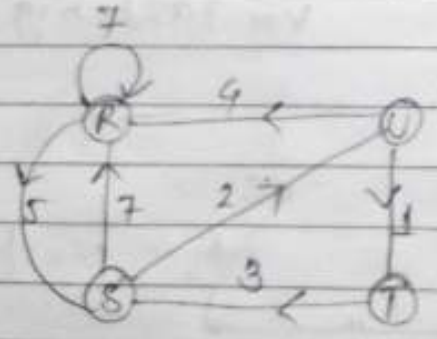
(1) Warshall Algorithm :-

The path Matrix tell us whether OR not there are paths b/w the Nodes, now we want to find a Matrix Q. which will tell us the length of the shortest path b/w the Nodes. The Matrix Q is found by the Warshall Algorithm using formula

$$Q_k[i,j] = MIN(Q_{k-1}[i,j], Q_{k-1}[i,k] + Q_{k-1}[k,j])$$

The Matrix Q_0 is same as the weight Matrix except Zero is represented by infinity.

Ex. find out the all-pair shortest path of the given graph



$Q_0 =$

	R	S	T	U
R	7	5	∞	∞
S	7	∞	∞	2
T	∞	3	∞	∞
U	4	∞	1	∞

$$Q_1 = \begin{bmatrix} 7 & 5 & \infty & \infty \\ 7 & 12 & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & 9 & 1 & \infty \end{bmatrix}, \quad Q_2 = \begin{bmatrix} 7 & 5 & \infty & 7 \\ 7 & 12 & \infty & 2 \\ 10 & 3 & \infty & 5 \\ 4 & 9 & 1 & 11 \end{bmatrix}$$

Date / /
Page No.

$$Q_3 = \begin{bmatrix} 7 & 5 & \infty & 7 \\ 7 & 12 & \infty & 2 \\ 10 & 3 & \infty & 5 \\ 4 & 4 & 1 & 6 \end{bmatrix} \quad ; \quad Q_4 = \begin{array}{c} \\ R \\ S \\ T \\ U \end{array} \begin{array}{c} \begin{array}{cccc} R & S & T & U \end{array} \\ \begin{bmatrix} 7 & 5 & 8 & 7 \\ 6 & 6 & 3 & 2 \\ 9 & 3 & 6 & 5 \\ 4 & 4 & 1 & 6 \end{bmatrix} \end{array}$$

Algorithm :-

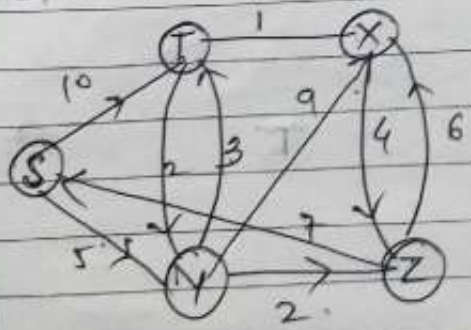
Step 1. Repeat for $I, J = 1, 2, \ldots M$
$W[I,J] = 0$ then set $Q[I,J] = \infty$
else set $Q[I][J] = W[I,J]$

2. Repeat Steps 3 & 4 for $k = 1, 2 \ldots M$
3. Repeat step 4 for $I = 1, 2, \ldots M$.
4. Repeat for $J = 1, 2, \ldots M$.
Set $Q[I,J] = MIN(Q[I,J], Q(I,K) + Q(K,J))$
End of loop
End of step 3
End of step- 2.

5. Exit.

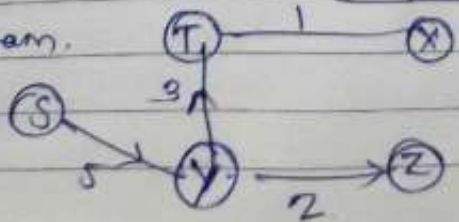
(8) Dijkstra's Algorithm :-

Qp. Find out the shortest path from node S to all other vertex.

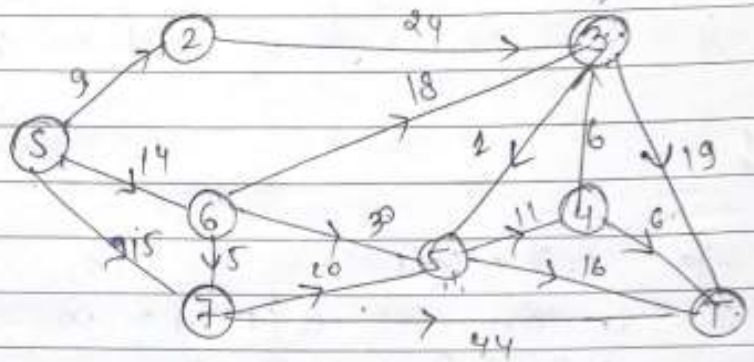


S	T	X	Y	Z
0	∞	∞	∞	∞
	10	∞	5	∞
	8	14	5	7
	8	13	5	7
	8	9	5	7

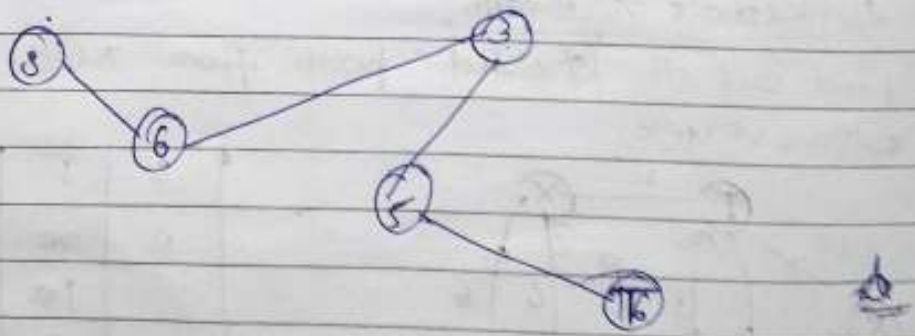
final diagram.



Ques - Trace the Dijkstra Algorithm of the following graph taking S as a Source Node & T as Destination Node.



0	S	2	3	4	5	6	7	T
	0	∞	∞	∞	∞	∞	∞	∞
		9	∞	∞	∞	14	15	∞
		9	33	∞	∞	14	15	∞
		9	32	∞	44	14	15	∞
		9	32	∞	35	14	15	59
		9	32	∞	34	14	15	51
		9	32	45	34	14	15	50



Algorithm

~~Dijkstra (G, w, s)~~

1. Initialize Single Source (G, s)
2. S = ∅
3. Q = V[G]
4. while Q ≠ ∅
5. do u = EXTACT MIN (Q)
6. S = S ∪ {u}
7. for each vertex v ∈ Adj (u)
8. DO RELAX (u, v, w)

Initialize single source (G, s)

1. for each vertex v ∈ V[G]
2. do d[v] = ∞
3. π[v] = NIL
4. d[s] = 0

RELAX (u, v, w)

1. if d[v] > d[u] + w(u, v)
2. then d[v] = d[u] + w[u, v]
3. π[v] = u

d[v] : shortest distance of v from source

π[v] = represents predecessor of node v.