Data Structure:-

It is a wave to organize the data in some way so we can do the operations on these data in effective way. Data structure may be organized in many different ways. The logical or mathematical model of a particular organization of data is called data structure. Choice of data model depends on two considerations.

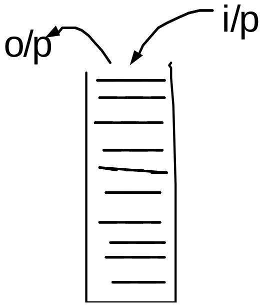
It must be rich enough in structure to mirror the actual relationship of the data in the real world.

The structure should be simple enough that one can effectively process the data when necessary.

Data structure can be classified as:-

1. **Linear data structure:-**

Processing of data items by linear fashion is linear data structure e.g. array, stack, queue, linked list.



In case of stack:- -Last input is the first output

1 input process is called push.

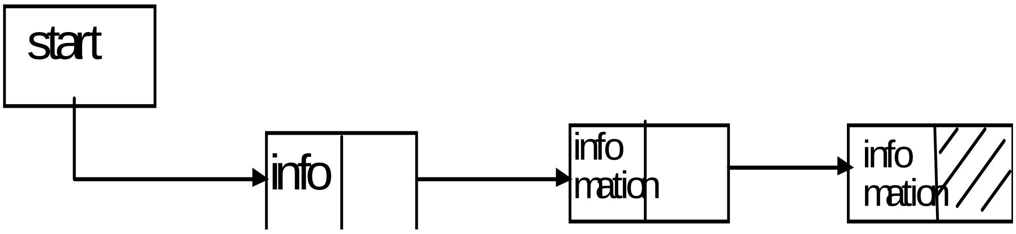
2 Output process is called pop.

In case of queue - First input is the first output.

3 Input is called rear.

4 Output is called front.

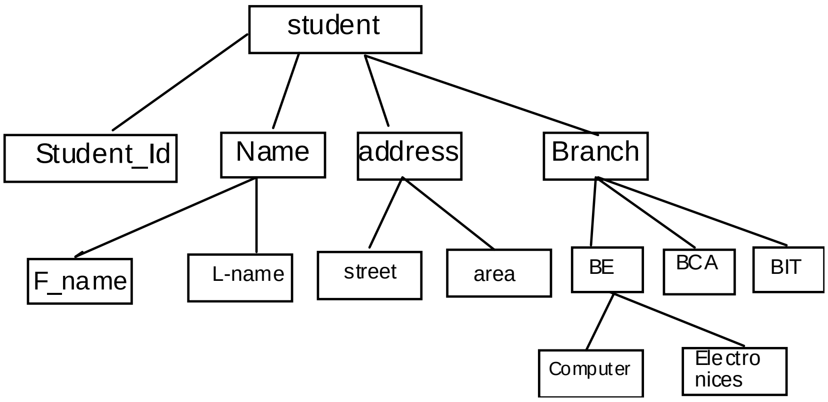
In case of linked list



First part consists of information and second part consists of pointer to second block.

2. **Non Linear data structure:-**

A data structure in which insertion and deletion is not possible in a linear fashion is called non –linear data structure. E.g. tree, graph.



- It is hierarchical in nature.

Graph:-

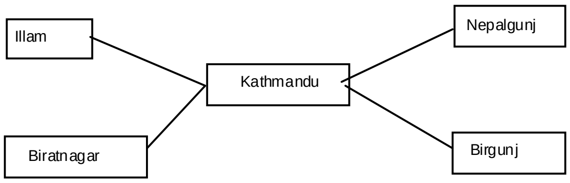


fig. airlines flights

- Contents relationship between pair of elements which is not in hierarchical in nature.

3. **Abstract Data type (ADT):-**

ADT is used to specify the logical properties of the data type. It is a set of operation which is called with the component of the element of that abstract data type.

e. g. list as ADT — the term ‘ADT’ refers to basic mathematical concept that defines the data type.

<u>Component</u>

Item

<u>Operations</u>

-Insertion

- Deletion
- Search
- Display

Here, item is component of ADT.

An ADT consists of two parts.

(a) value definition

i. Definition clause

ii. Condition clause

(b) operator definition

# ADT for Relation:-

/*value of definition */

Abstract type def <integer, integer> RTIONAL // value definition

Condition RATIONAL [1] = 0;   // denominator is not equal to zero; cond^n.

/* operator definition * /

Abstract RATIONAL make rational (a, b)

Int a,b;

Precondition b! = 0;

Post condition make rational [0] == a;
                    Make rational [1] == b;

Abstract RATIONAL add (a, b)

RATIONAL a,b;

Past condition add [1] == a[1] * b[1];
                     Add [0] == a[a] * b[1] + b[0] * a[1];

Abstract RATIONAL mult (a,b)

RATIOANL a, b;

Post condition   mult [0] ==a[a] * b[0];
                     Mult [1] == a[1] *b[1];

Abstract RATIONAL equal (a,b)

RATIONAL a,b;

Post condition equal == (a[0] * b[1] == b [0] * a[1]);

Goals of data structure:-

Data structure involves two complementary goals:-

(1) identify and develop useful mathematical entities and operations and determine what class of problems can be solved by using these entities and operations.

(2) Determine representation fro those abstract entities and implement the abstract operations on these concrete representations.

Properties of Data structure:-

- Generalization:-

Entity generalizes the primitive data type (integer, real, float)

- Encapsulation:-

All the operation of that type can be localized to one section of program.

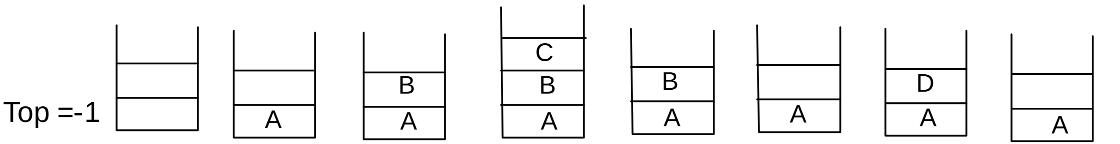
Chapter:- 2

Stack:- Stack is an ordered collection of items in to which new items may be inserted and from which items may be delete at one end, called tope of the stack.

Two operations are possible on stack:-

(1) push:- Add item in to stack.

(2) Pop :- delete the item from stack.



(a) Empty (b) Push A (c) Push B (d) Push C (e) Pop C (f) Pop B (g) Pus D (h) Pop D

From figure we see that the items inserted at last is the first item to be poped or deleted. Thus stack follows last in first out (LIFO) or first in last out (FILO) policy.

Stack of ADT:-

```
Abstract typedef << dtype>> STACK(datatype);  //define datatype (stack type element)
Abstract empty (s)                            // for empty function.
STACK (eltype) s;
Past condition empty == (len(s)== 0);         // condition for empty.
Abstract eltype pop (s)                       //function  declaration.
STACK (eltype ) S;                            // variable declaration.
Precondition empty (S)== FALSE;               //condition for fulfill.
Past condition  pop == first (S');            /length in pree condition.
                S= = subh(S",1, len(S')-1);  // S'- original length, 1- deleting item,
                                                     len(S)1- post condition .-
                                              push
abstract push (S, elt)                        //  add new element elt.
STACK       (eltype) S;                       // new variable declearation.
Post condition S== <elt>+S';                  //
```

Stack Implementation:-

(i) Array implementation

(ii) Linked list.

Array Implementation:-

In Array we can push elements one by one from 0th position 1st position …….. n-1th position. Any element can be added or deleted at any place and we can push or pop the element from the top of the stack only.

1. When there is no place for adding the element in the array, then this is called stack overflow. So first we check the value of top with size of array.
2. When there is no element in stack, then value of top will be -1. so we check the value of top before deleting the element of stack.

	5	10	15				
Stack_array	[0]	[1]	[2]	[3]	[4]	[5]	[6]

Here, Stack is implemented with stack array, size of stack array is 7 and value of top is 2.

Operation of Stack:-

Push Operation:-

```
If (top == (max-1)]
        Printf("stack over flow");
Else
{
Top == top + 1;
Stack_arr[top] = pushed –item;
}
```

Pop Operation:-

```
If (top ==-1)
        Printf("stack underflow");
Else
{
        Printf("Poped element is %d",stack_arr[top]);
Top= top – 1;
}
```

Algorithm for push & Pop

PUSH:-

```
Let stack [Max size] is an array for implementing the stack
1. [check for stack overflow?]
   If top = maxsize-1, then print overflow & exit
2. set top = top + 1 (increase top by 1)
3. set stack [top] = item (inserts item in new top position )
4. exit.
```

POP:-

```
1. check for the stack underflow
       If Top <0 then
           Print stack underflow and exit
       Else
           [Remove the top element]
           Set item = stack [ top]
2. Decrement the stack top.
3. Return the deleted item from the stack.
4. Exit.
```

Application of Stack:-

1 Conversion of an expression from infix to post fix.

2 Evaluation of an arithmetic expression from post fix expression.

Precedence:-

$ (Power), %(remainder)	- 5
*(mul), /(div)	- 4
+ (add), -(sub)	- 3
(	-2
)	- 1

Conversion of Infix to post fix:-

Algorithm:-

1. Add a unique symbol # in to stack and at the end of array infix.
2. Scan symbol of array infix fro left to right.
3. If the symbol is left parenthesis '('then add in to the stack.
4. If symbol is operand then add it to array post fix.
5. (i) If symbol is operator then pop the operator which have same precedence or higher precedence then the operator which occurred.
   (ii) add the popped operator to array post fix.
   (iii) Add the scanned symbol operator in to stack.
6. (i) If symbol is right parenthesis ')' then pop all the operators from stack until left parenthesis '(' in stack.
   (ii) Remove left parenthesis '(' from stack.
7. If symbol is # then pop all the symbol form stack and add them to array post fix except #.
8. Do the same process until # comes in scanning array infix.

Change Infix to post fix:-

Q. 1. A + B – C + D

1 A+ B – C + D #

Step	**Symbol**	**Operator in stack**	**post fix expression**
1	A	#	A
2	+	#-	A
3	B	#+	AB
4	-	#-	AB+
5	C	#-	AB+ C
6	+	#+	AB +C -
7	D	#+	AB + C -D
8	#	#	AB+C-D+

Q.2.A* (B+C $D) – E $F * (G/H)

A*(B+C$D) – E$ F*(G/H)

Step	**Symbol**	**Operator in stack**	**post fix expression**
1		A	
	#		
	A		
2	*+	#*	A
3	(	#+	AB
4	B	#-	AB+
5	+	#-	AB+ C
6	C	#+	AB +C -
7	$	#+	AB + C -D
8	D	#	AB+C-D+
9	)	#*	ABCD $+
10	-	#-	ABCD$+*
11	E	#-	ABCD$+*E
12	$	#-$	ABCD $ + *E
13	F	#-$	ABCD$ +*EF
14	*	#-*	ABCD$ +*EF$
15	(	#-*(	ABCD$+*EF$
16	G	#-*(	ABCD$+*EF$G
17	/	#-*(/	ABCD$+*EF$G
18	#	#-*(/	ABCD$+ *EF$GH/
19	)	#-*	ABCD$ +*EF$GH/*-
20	#	#	ABCD$+*EF$GH/*-

Hence, the required post fix expression is ABCD$ +* EF $ GH/*-

Evaluation of post fix expression:-

In this case stack contents the operands. In stead of operators. Whenever any operator occurs in scanning we evaluate with last two elements of stack.

Algorithm:-

(i) Add the unique symbol # at the end of array post fix.

(ii) Scan the symbol of array post fix one by one from left to right.

(iii) If symbol is operand, two push in to stack.

(iv) If symbol is operator, then pop last two element of stack and evaluate it as [top- 1] operator [top] & push it to stack.

(v) Do the same process until '#' comes in scanning.

(vi) Pop the element of stack which will be value of evaluation of post fix arithmetic expression.

Postfix expression ABCD $+* EF$GHI * -

Evaluate postfix expression where A = 5, B = 5, C = 4, D = 2 E = 2, F= 2 , G = 9, H= 3 now, 4, 5, 2, $, +, *, 2, 2, $, 9, 3, /, *, - , #

Step	**Symbol**	**Operand in stack**
1	4	4
2	5	4, 5
3	4	4,5,4
4	2	4,5,4,2
5	$	4,5,16
6	+	4,21
7	*	84
8	2	84,2
9	2	84, 2,2
10	$	84,4
11	9	84,4,9
12	3	84,4,9,3
13	/	84, 4, 3
14	*	84,12
15	-	72

The required value of postfix expression is 72

For checking:-

A*(B+C$D) – E$F*(G/H)

$4*(5+4^2)-2^2*(9/3\ \ )$

4* (5+16)-4*3

84-12

72

Q. Convert the post fix:-

((A+B)*C – (D-E) 4F+G

Evaluate post fix where A=1, B=2, C= 3, D= 4, E= 3, F=2, G=1.

Verify it by evaluating it’s infix expression.

Step	Symbol	Operand in stack	Post fix operation
1	(	#(	
2	(	#((	A
3	+	#((+	A
4	B	#((+	AB
5	)	#()	AB+
6	*	#(*	AB+
7	C	#(*	AB+C
8	-	#(-	AB+C*
9	(	#(-(	AB+C*
10	D	#(-(-	AB+C*D
11	-	#(-(-	AB+C*D
12	E	#(-(-	AB+C*DE
13	)	#(-	AB+C*DE-
14	)	#	AB+C*DE--
15	$	#$	AB+C*DE--
16	(	#$(	AB+C*DE--
17	F	#$(	AB+C*DE - -f
18	+	#$(+	AB+C*DE--f
19	G	#$(+	AB+C*DE - - FG
20	)	#	AB+C*DE- - FG+ $

The required postfix expression is AB+C*DE - - FG +$

Now 1,2, +3, *4, 3, -, -, 2, 1, +, $, #

Step	**Symbol**	**Operand in stack**
1	1	1
2	2	1,2
3	+	3
4	3	3,3
5	*	9
6	4	9,4
7	3	9,4,3,1
8	-	3,1
9	-	8
10	1	8,2
11	1	8,2,1
12	$	8,3
13	$	24

For checking

The given infix expression is

((A+B)*C-(D-e))$(F+G)

= ((1+2)*3-(4-3))4(2+1)

= (3*3 -1) $3

= (9-1)$3

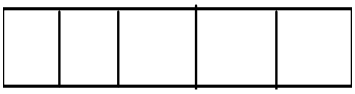
= 8 $ 9

= 24

Chapter:- 3

Queue:-

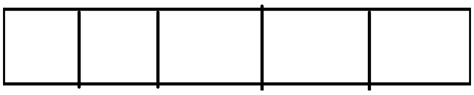
A queue is an ordered collections of items from which items may be deleted at one end called the front of the queue and in to which items may be inserted at the other end called rear of the queue.

Deletion ⇒  ⇐ Insertion

Front rear

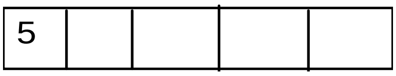
e.g.

queue – arr [5] [0] [1] [2] [3] [4]



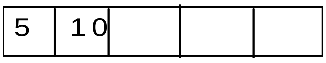
front =-1 (a) empty queue.

Rear =-1



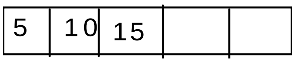
Front =0 fig:- adding an item in queue

Rear =0



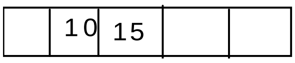
Front =0 fig:- adding an item in queue

Rear = 1



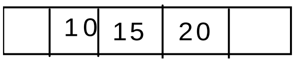
Front =0

Rear =2 fig:- adding an item in queue.



Front = 1 fig:- deleting an element from queue

Rear =2



Front = 1

Rear = 3 fig:- adding an element in queue

from figure we see that item are inserted at rear end and deleted at front end. Thus queue follows FIFO policy. (First in first out).

Queue as an A-DT

```
Abstract typedef <<eltype>>QUEUE (eltype);   //queue type element.
Abstract empty (q)
QUEUE (eltype) q;
Post condition empty = = (len (q) = =0);
Abstract eltype delete (q)
QUEUE(eltype ) q;
Pre condition empty (q) = = FALSE
Post condition remove = = front (q’);
                        q = = sub(q’, 1, len(q’-1);
abstract insert (q, elt)
QUEUE (eltype) q;
Eltype elt ;
Post condition  insert = rear (q’);
                        Q = q’+<elt>;
```

Queue Implementation :-

8 array implementation
9 linked list implementation

Array implementation of Queue:-

A queue top two pointers front and rear pointing to the front and rear element of the queue.

- when there is no place for adding elements in queue, then this is called queue overflow.

- when there is no element for deleting from queue, then value of front and rarer will be -1 or front will be greater then rarer.

[0]	[1]	[2]	[3]	[4]
	5	10		

Front t =1

Rear =1

Operation in queue

1. Add operation :-

```
If (rear = = Max -1)
Printf(“queue overflow”);
Else
{
If (front = = -1)
Front = 0;
Rear = rear + 1;
Queue_arr[rear]= added-item;
}
```

Delete operation:-

```
If (front = = -1)|| (front >rear)
{
Printf(“Queue under flow”);
        Return;
}
Else
Printf(“Element detected from queue”, queue_arr[fornt]);
Front = front +1;
}
```

Drawbacks:-

[0]	[1]	[2]	[3]	[4]
		5	10	15

Front =2

Rear = 4

Here we see that the rear is at the last position of array and front is not at the zeroth position. But we can not add any element in the queue because the rear is at the n-1 position.

There are two spaces for adding the elements in queue. But we can not add any element in queue because are is at the last position of array. One way is to shift all the elements of array to left and change the position of front and rear but it is not practically good approach. To overcome this type of problem we use the concept of circular queue.

Algorithm to insert an element in a queue

```
Step:-1
[check overflow condition]
condition]
If rear (=) > [Max -1]
o/p :"over flow"
return;
Step:-2
[increment rear pointer]
Rear = rear +1
Step:-3
[insert an element ]
Q[rear]= value

Step:-4
[set front pointer]
  If front =-1
Step:-5
Return
```

Deletion

```
Step:-1
[check underflow

if front =-1
o/p:"under flow & return;

Step:-2
Remove an element
value: Q[Front]
Step:-3
[check for empty queue]
if front = =  rear
        Front =-1
        Rear = -1
else
        front = front +1
Step:-4
        Return [value]
```

Circular Queue:-

[0]	[1]	[2]	[3]	[4]
		10	20	30

A circular queue is one in which the insertion of new element is done at the very first location of the queue if the last location of the queue is full.
e.g consider C queue_arr[4]

a.

[0]	[1]	[2]	[3]
	5	10	15

from = 1 rear = 3.

initial queue

b.

		10	15

front = 2

fig. deletion in queue

rear = 3

c.

35		10	15

front =0 rear = 1

fig. addition in queue.

d

35	20	10	15

rear =1

front =2

fig. addition in queue.

e

35	20		15

front =1

rear = 3

fig. deletion in queue.

f

35	20		

front = 1

rear =1

fig. deletion in queue

g

			20

front = 1

rear =1

fig. deletion in queue

h

rear = 1

Front =-1

fig. deletion in queue

Insersation:-

```
If ((front == 0 && rear = Max -1 ) || (front = = rear +1))
        {
Printf("Queue overflow");
Return;
}
If (front = = - 1)
{
Front =0;
Rear = 0;
}
Else
If(rear == Max -1)
Rear =0;
Else
Rear = rear +1;
(queue_arr[rear] = added_item;
```

Operation

Deletion:-

```
If (front = = -1
{ printf("queue underflow");
Return;
}
Else
Printf("element deleted from queue is %d");
(queue_arr[front];
If (front == rear)
{
Front =-1;
Rear = -1;
}
Else
```

```
If (front = = Max – 1)
    Front = 0;
Else
    Front = front +1;
```

Dequeue (Double ended queue)

In dequeue we can add or delete the element from both sides i.e. front end or form the rear end. It is of two types.

(i) I/P restricted

(ii) O/P

In input restricted dequeue element can be added at only one end but we can delete the element from both sides.

In output restricted dequeue eleme4tn can be added from both side but deletion is allowed only at one end.

Operation in Dequeue:-

We assume a circular array fro operation of addition or deletion.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
		5	10	15			

Left = 2

Right = 4

We maintain two pointers right & left which indicate positions of dequeue. Here, left pointer is at position 2 and right pointer is at position 4.

For I/P restricted:-

Add 8 in the queue from right.

		5	10	15	8		

left =2 right =5

Delete the element from left of the queue

			10	15	8		

left =3 right =5

Add the element 20 in the queue

			10	15	8	20	

left =3 right =6

Add the element is in queue

			10	15	8	20	16

left =3 right =7

Delete the element form right of queue.

			10	15	8	20	

left =3 right =6

Add the element 80 in the queue

			10	15	8	20	30

left =3 right =7

Add the element 12 in the queue.

n			10	15	8	20	30

left =0 right =3

Add the element 35 in the queue.

12	35		10	15	8	20	30

left =1 right =3

Add the element 5 in the queue.

12	35	6	10	15	8	20	30

left =2 right =3

Add the element 45 in the queue. No it overflows because right pointer will become equal to the left pointer after adding an element.

Right addition:-

```
If(left = = 0 && right == Max -1) || (left = = right +1))
{
Printf("Queue overflow");
Return;
}
If (left==-1)
{
Left =0;
Right =0;
}
Else
If (right = = Max -1)
```

```
Right = 0;
Else
Right = right +1;
Dedequeue_arr[right] = added_item;
```

Left Addition:-

```
If (( left == 0 && right == Max-1) || (left == right +1))
{
Printf("Queue overflow");
Return;
}
If (left==-1)
{
Left =0;
Right =0;
}
Else
If (left == 0)
Left = Max -1;
Else
Left = left -1;
Dequeue_arr[left] = added_item;
```

Delete left:-

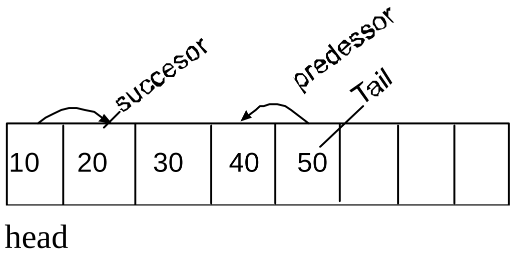
```
If (left==-1)
{
Printf("Queue flow");
Return;
}
Printf("Element deletd from queue is "%d", dequeue_arr[left].  // point delete item
If (left == right ) // make empty
{
Left ==-1;
Right =-1;
}
Else
If(left== Max -1)          //move from zero.
Left =0;
Else
Left = left +1;            // increase space.
```

Delete Right:-

```
If (left ==-1)
{
Printf("Quieue underflow");
Return;
}
Printf("element delected from queue is %d", deque_arr[right]);
If(left==right)
{
Left =-1;
Right =-1;
}
Else
If(right==0)
Right =Max-1;
Else
Right = right [-1;
}
```

Chapter:- 4

A linear list is an ordered set consisting of a umber of elements to which addition w& deletion can be made. A linear list displays the relationship of physical adjenci. The first element of a list is called the hear of list & the last is called the tail of the list. The next element of the head of the list is called it's successes. The previous element to the tail (if it is not head of the list) is called it's predessor. Clearly a head doesn't have as predessor & a tail doesn't have a successor. Any other element of the list has both one successor & one predessor.



Operations perform in list:-

1. Traversing an array list
2. Searching an element in the list
3. Insertion of an element in the list
4. Deletion of an element in the list.

Array Implementation:-

Take an array of size 10 which has 5 elements

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
10	20	30	40	50					

⇒ **Traversing an array list:-**

Here each element can be found through indeed no. of array & is incremented by 1.

When index =0.

Then,

Arr[10] = 10 ⇒ determines 1st element in array

Index = index +1

Then

Arr[index]= 20

In this way we can transverse each element of array by incrementing the index by 1 until index is not greater than no. of elements.

⇒ Searching in an array list:-

For searching an element, we first travers the array list and while traversing , we compare each element of array with the given element.

```
Int I, item;
For (i=0;i<n;i++)
‘{
If(item== arr[i])
Return(i+1);
}
```

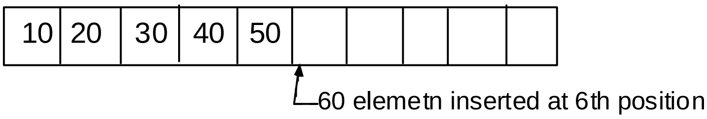
Insertion of an element in the list:-

-two ways:-

(i) insertion at end

(ii) insertion in between.

(i) Insertion at end



10	20	30	40	50					

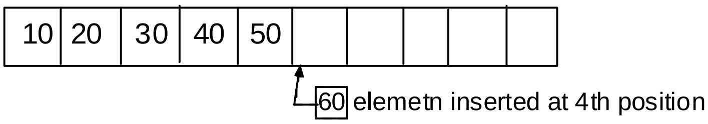
60 elemetn inserted at 6th position

Set the array index to the total no of elements & then insert the element

Index = Total no of element (i.e 5)

Arr[index] = value of inserted element

2. Insertion in between



10	20	30	40	50					

60 elemetn inserted at 4th position

Shift right one position all array elements from last array element to the array element before which we want to insert the element.

```
Int tem, item, position;
If (n = = max)
{
```

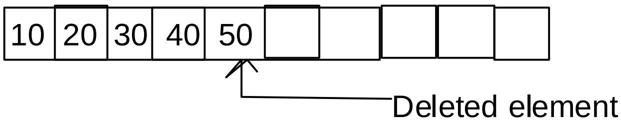
```
Printf("list overflow");
Return;
}
If (position >n+1)
{
Printf("enter position less than or equal to %d"n+1);
Return;
}
If(position = = n+1)          /*insertion at the end */
{
Arr [n]= item;
N= n+1;
Return;
}
Temp = n-1             //insertion in between
While (temp>=positon-1)
{
Arr[temp+1] = arr [temp];
Temp - -
}
Arr [positon -1] = item;
n = n+1;
```

Deletion of an element in the list:-

deletion of the last element

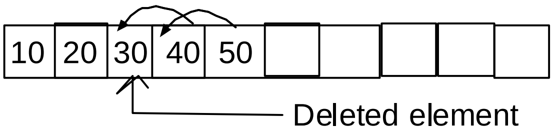
deletion in between

Deletion of the last element:-



Traverse the array last and if the item is last item of the array, then delete that element & decrease the total no of element by 1,

Deletion in between:-



First traverse the array list & compare array element with the element which is to be deleted, then shift left one position from the next element to the last element of array and decrease the total no. of elements by 1.

```
Int temp, position , item, n;
If (n= = 0)              // present or not element
{
Printf("list underflow");
Return;
}
If (item == arr [n-1]          //deletion at the end.
{
n = n- 1;
return;
}
Temp = position  -1;
While (temp < = n -1)
{
Arr [temp] = arr [temp +1]
Temp + + ;
}
n = n -1;
```

Advantage of list:-

⇒ Easy to compute the address of the array through index 4 can be access the array element through index.

Disadvantage:-

⇒ Use of contiguous list which is time consuming.

⇒ As array size is declared we can't take elements more then array size.

⇒ If the elements are less than the size of array, then there is wastage of memory.

⇒ Too many shift operation on each insertion and deletion.

To overcome this problem we use liked list or DLL.

Index to prefix conversion :-

Rules:-

1. Parenthesize the expression starting from left to right.
2. During parenthesizing the expression, the operands associated with operator having higher precedence are first parenthesized.
   e.g. B * C is first parenthesized before A+B in A+B *C
3. the sub expression which has been converted in to prefix is to be treated as single operand.
4. once the expression is converted to postfix from remove the parenthesis.

Question:- (A*B + (C/D) –F

(A*B + (C/D) –F

➢ ((A*B + (C/D)) – F)	
➢ ((A*B+W) – F)	[W = /CD let]
➢ ((*AB + W) –F)	
➢ ((X +W)-F	[X = *AB]
➢ (+XW – F)	
➢ (Y –F)	[Y = +XW
➢ -YF	
➢ -+XWF	
➢ - + ABWF	
➢ - + * AB/CDF.	

Which is required prefix expression.

Dynamic memory allocation:-

In an array it is necessary to declare the size of array. This creates two possible problems.

If the records that are stored is less than the size of array, then there is a wastage of memory.

If we want to store more records then size of array, we can’t store.

To overcome these problem we use dynamic memory allocation. The functions used for dynamic memory The functions used for dynamic memory allocation and deallocaton are :-

Malloc()
Calloc()
Free()
Realloc()

Malloc():-

This function is used to allocate memory space. The malloc () function reserves a memory space of specified size and gives the strating address to the pointer variable.

Syntax:-

Ptr = (data type *) malloc (specified size)

Type of pointer size required to reserve in memory.

Example:-

```
Ptr = (int *) malloc (10);
Struct student
{
Int roll_no;
Char name [30];
Float percentage;
};
Struct student  * st_ptr.
St_ptr = (struct student *) malloc size of (struct student);
```

Calloc():-

The caloc () function is used to allocate multiple blocks of memory. This has two arguments.

Syntax:-

```
Ptr = (data type *) calloc (memory block, size )
```

Example:-

```
Ptr = (int * ) calloc (5, 2)
Struc record
{
Char name [10];
Int age;
Float sal;
};
Int to_record =100;
Ptr = (struct record *) calloc (tol_record, size of (record));
```

Free():-

This function is used to deallocate the previously allocated memory using mulloc () or calloc() function .

Syntax:-

```
Free (ptr);
```

Realloc():-

This function is used to resize the size of memory block which is already allocated. It is used in two condition.

⇒ If the allocated memory block is insufficient for current application.

⇒ If the allocated memory is much more thasn what is required by the current application.

Syntax:-

To allocate memory; we use

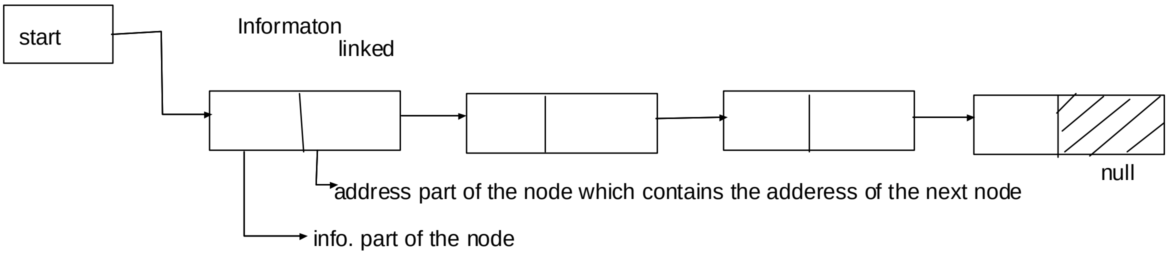
```
Ptr = (char *) mallc(6);
```

To reallocate memory;

```
Ptr = (char * ) realloc (ptr, B)
```

Chapter:- 5

A link list is a collection of elements called nodes each node hap two parts, 1st part contains the information field & the 2nd part contains the address of the next node. The address part of the last node of linked list will have null value.


start
Informaton
linked
null
address part of the node which contains the adderess of the next node
info. part of the node

address part of the node which contains
the address of the next node.

Infro. Part of the node.

In liked list one member is a pointer that points to a structure itself.

Syntax:-

```
   Struct node
  {
  Int data;
  Struct node * link;
  };
```

Here, member of the structure struct node * link points to the structures itself.

Implementation:-

Traversing a linked list:-

In liked list start is a pointer which points to the 1st element of the list for processing next element we assign the address of the next element to the pointer (ptr) as:

Ptr = ptr – link;

Each element of the liked list can travel through this assignment until ptr has null address. So, the linked list can be traversed as:-

```
Which (ptr ! = NULL)
Ptr = ptr – link;
```

Searching in to a linked list:-

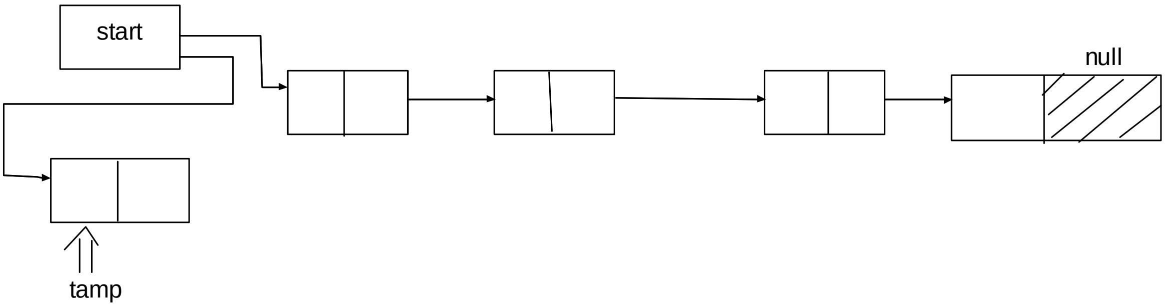
For searching the element first traverse the linked list and with traversing compare the information part of each element with the given element it can be written as:

```
While (ptr!=NULL)
{
If (ptr→ info = = data)
Printf("Item %d found at position %d", data, pos);
Else
Ptr = ptr → link;
```

Insertion in to a linked list:-

Insertion at beginning

Insertion in between



temp

Here, temp is a pointer points to the node that has to be inserted ie.

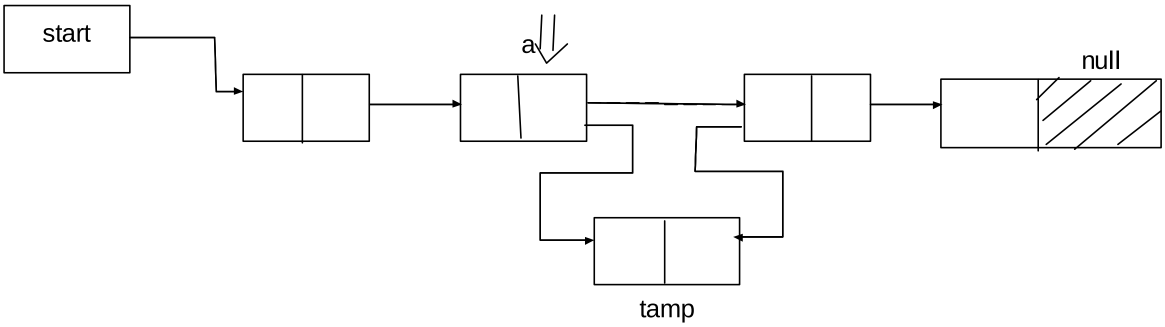
Temp → infro = data;

Start points to the 1st elements of linked list for insertion at beginning we assign the value of start to the linked part of the inserted node as

Temp → infro = start; and start can be re assigned as

Start = temp;

Insertion in between:-



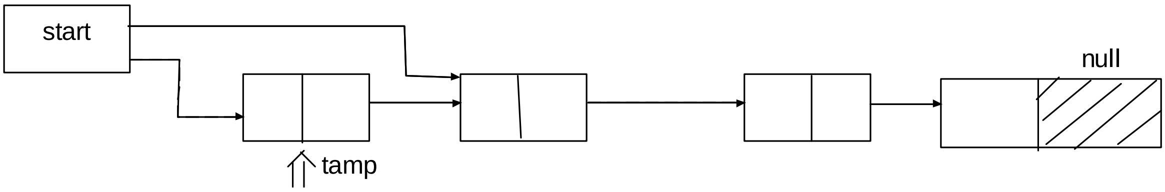
First we traverse the linked list for obtaining the node after which we want to insert the element. We obtain pointer 9 which points to the element after which we have to insert new node. For inserting the element after the node, we assign link part of that node to the link part of inserted node & the address of the inserted node is placed into the link part of the previous node. This can be written as

Temp → infro = data;

Temp → link = q → link;

q → link = temp;

Deletion in to a linked list:-



Here, start points to the 1st element of linked list. If element to be deleted is the 1st element of linked list then we assign the value of start to temp as

Temp = start;

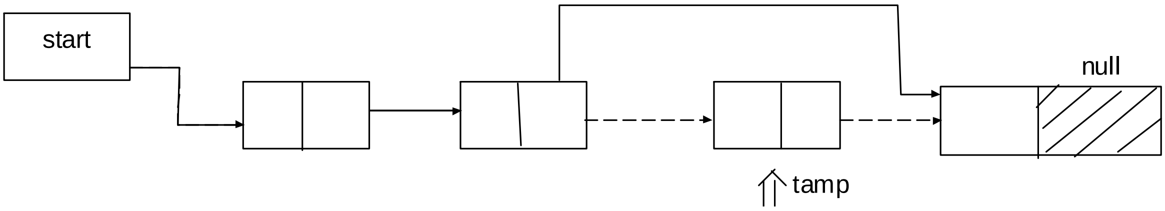
temp points to the 1st node which has to be deleted & assign link part of the deleted node to start as

start = star – link

Since start points to the 1st element of linked list so, start→ link will point to the second element of linked list,l now we free the element to be deleted. ie.

Free (temp)

Deletion in between:-



If the element is other then the 1st element of linked list then we give the link part of the deleted node to the link part of the previous node & is given by

Temp = q → link;

q → link = temp → link

Free (temp);

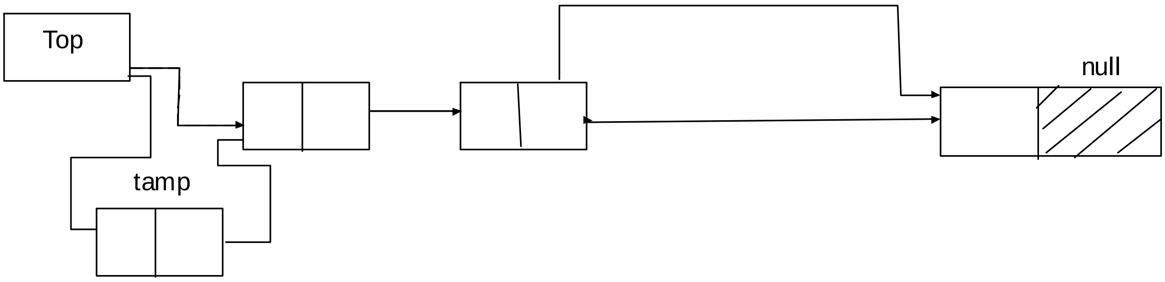
If the node to be deleted is last node of linked list then we can write

Temp = q → link;

q → link = NULL;

Free (temp);

Stack implementation using linked list:-



Push operation:-

For pushing the element on stack we follow the insertion operation of linked list. i.e. we add the element at the start of the list this can be written as

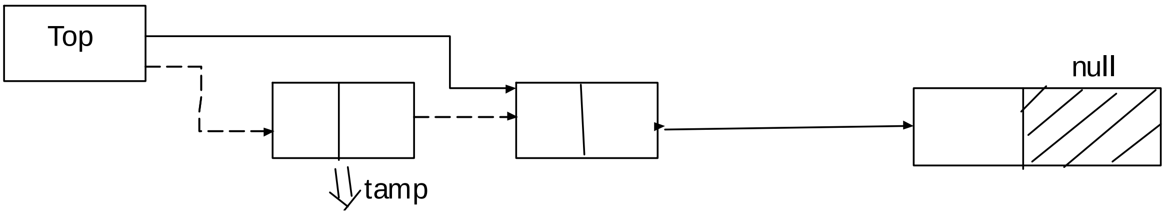
Temp = q → link = pushed item;

q → link = top;

top = temp;

Here, top always points to the 1st node of the linked list. Last pushed item will become 1st node of linked list. As there is use of linked list we don't check overflow condition.

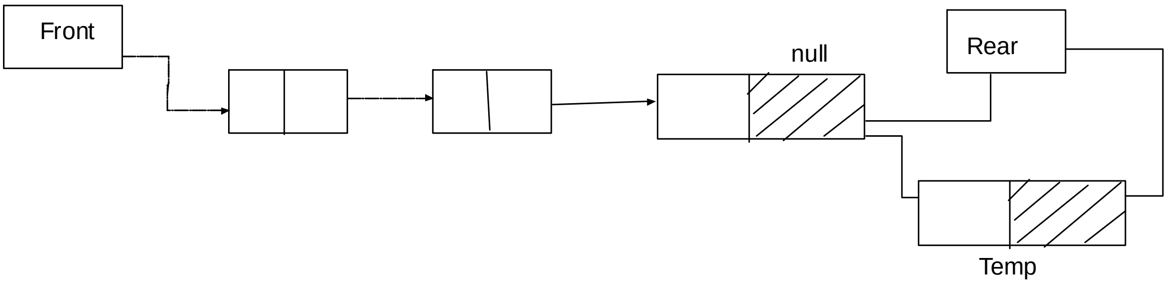
Pop operation:-



For this we delete the first element of linked list.

```
If (top = = NULL)
Printf("Stack is empty");
Else
Temp = top;
Printf("popped item is %d", temp→info);
Top = top→ link;
Free(temp);
```

Queue implementation using linked list:-



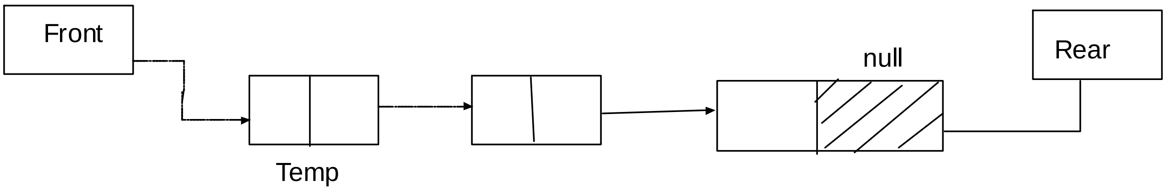
For adding the element in queue, we add the element at the end of the list. Here, front will point to the 1st node of the linked list & rear will point to the last node of the linked list this can be written as

```
Temp →  info = added_item;
Temp → link = NULL
If (front == NULL)
```

```
        Front = temp;
Else
        Rear → link = temp;
        Rear = temp;
```

Deletion operation:-

For deleting the element of a queue we delete the 1st node of linked list .

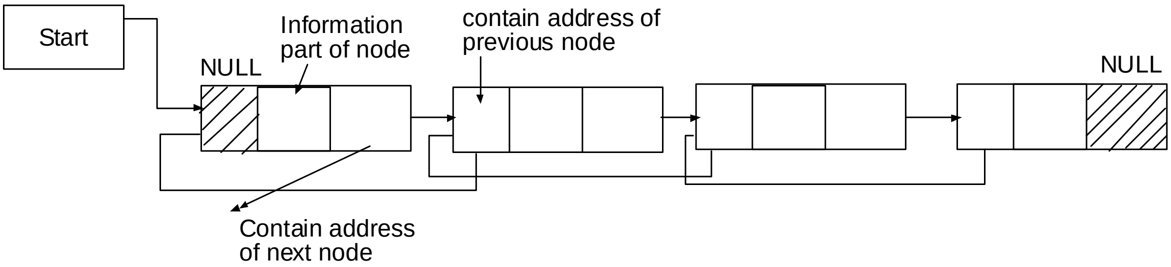


```
        If (front = = NULL)
        Printf("queue underflow");
        Else
        {
        Temp = fornt;
        Printf("Deleted element is %d", temp→ info);
        Front = front → link;
        Free (temp);
        };
```

Double linked list (DLL):-

In single linked list we can traverse only in one direction because each node has address of next node only. Suppose we are in the middle of linked list & we want to operation with just previous node, then we have no way to go in previous node. And we'll again traverse from starting node which is time consuming./ To overcome this problem we use double linked list.

In DLL each node has address of previous & next node along with data.


Start
NULL
Information part of node
contain address of previous node
NULL
Contain address of next node

Each node has to contain the address of information of previous node. The data structure for DLL is

```
Struct node
{
Struc node * previous;
Int info;
Struct node * next;
}
```

Here, struct node * previous is pointer to structure which will contain the address of previous node & struct node * next will contain the address of next node in list. Thus we can travel in both the direction.

Operation

1. **Traversing a double linked list:-**

   Start points to the 1st element of the list and we assign the value of start to ptr so ptr so ptr points to the 1st node of the list. For processing the next element we assign the address of next node to ptr as

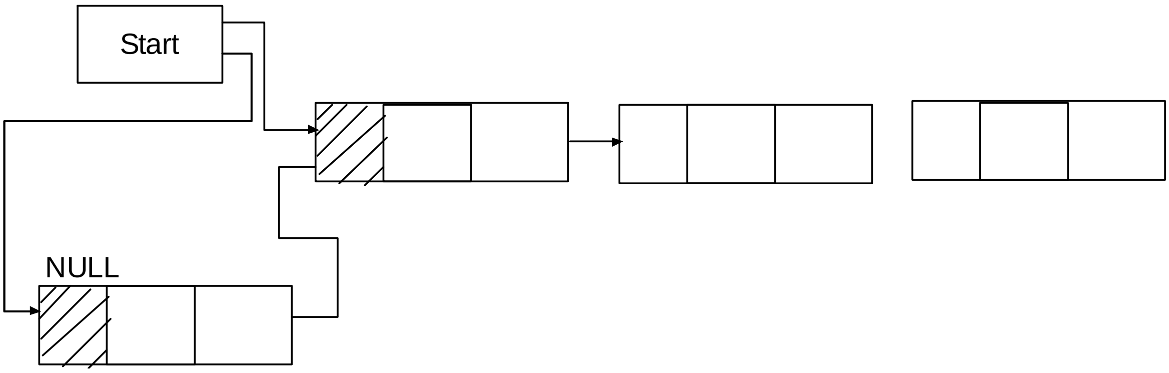
   Ptr = ptr → next

   Now, ptr has address of next node. Thus, we can traverse each element of list through this assignment until ptr has null value which the next part value of last element . it can traverse as:

   ```
   While (ptr!= NULL)
   Ptr = ptr→ next;
   ```

2. **Insertion:-**

   (a) **Insertion of beginning:**



For insertion at beginning we assign the value of start to the next part of inserted node and address of inserted node to the previous part of stat as:

temp→ next= start;

start → prev = temp

now, inserted node points to the next node which is beginning node of DLL and 'prev' part of second node will point to ht new inserted node. Now inserted node is the first node of DLL so start will be re assign as

start = temp;

now, start will point to the inserted node which is the 1st node DLL. Assign NULL to the previous part of inserted node as 'prev' node of the first node is null.

Temp → prev = NULL.

Thus whole process can re assign as

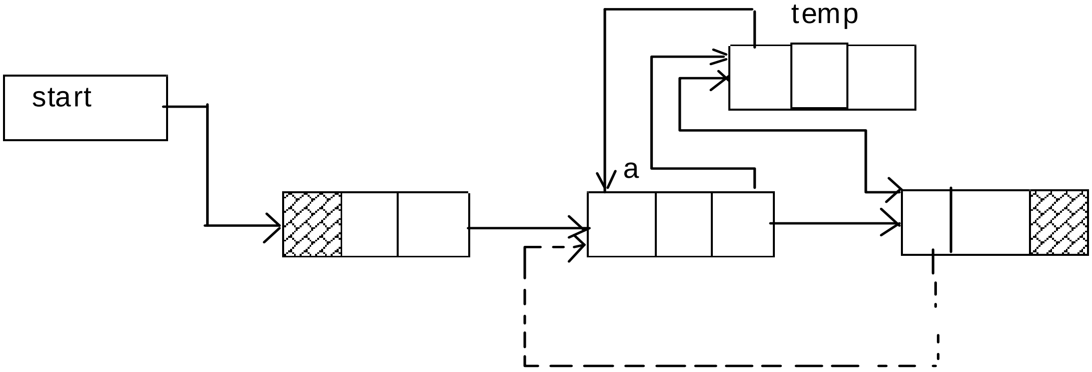
Temp → next = start;

Start → prev = temp;

Start → temp;

temp→ prev = Null.

Insertion in between:-



First we traverse the DLL for obtaining the node after which we want to insert the element. For inserting the element after the node we assign the address of inserted d node to the 'prev' part of next node. Then we assign the next part of previous node to next part of inserted node. Address of previous node will be assign to pre. Part of inserted node and the address of inserted node will be assigned to tyeh next part of prevous node. The whole process can be re assigned as

q→ next → prev = temp;

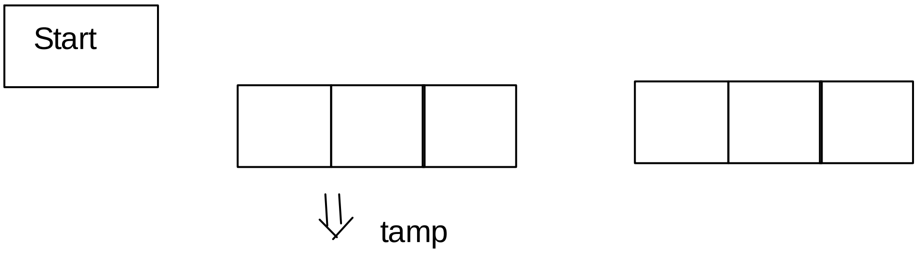
Temp → next = q → next;

temp→ prev = q;

q →next = temp;

Deletion

Deletion at beginning:-



Starts points to the 1st node of DLL. It node to be deleted is the4 1st of list then we assign the value of start to temp as

Temp = start ;

Assign next part of deleted node to start as

Start = start→ next;

Since, start points to the 1st node of linked list so start → next will point to the second node of list. Then null will be assigned to start point ot previous.

Start → previous = Null

Now, we can free the node to be deleted which is pointed by temp as;

Free(temp)

The whole process can be re assigned as

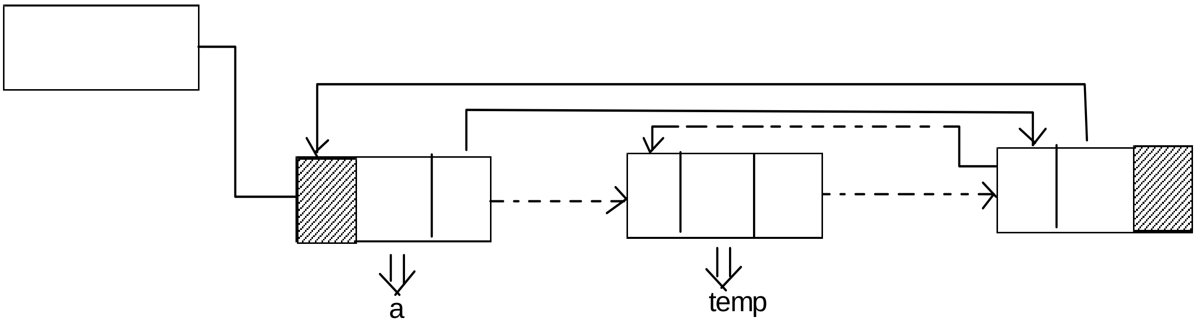
Temp = start;

Start = start → next;

Start → prev = NULL

Free (temp)

Deletion in between:-



If the element is other then the 1st element of linked list then we assign the next part of deleted node to next part of previous node & address of previous node to 'prev' part to next node this can be written as

Temp = q →next; //select node i.e. pointed.

a→ next → prev =q;

temp → next → prev = q;

free (temp);

Deletion of the last node:-

if node to be deleted is last node then we'll just free the last node & next part of second last will be null. This can be written as

temp =q → next;

free (temp);

q →next = Null.

Priority queue:-

In priority queue every of queue has some priority and based on that priority it wil be processed so the element of more priority will be process before the element which has less priority. If two elements have same priority. Then in this case it follows FIFO policy.

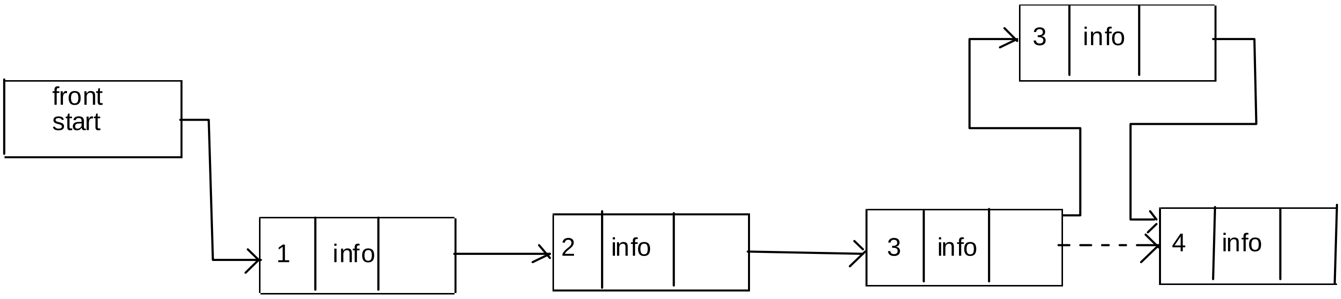
Structure:-

```
Struct pq {
        Int priority;
        Int info;
        Struct pq * link;
        }
```

Implementation:-

Temp → info = added – item

Temp → priority = item – priority;



```
If (front = = NULL || item – priority < front → priority)
{
Temp → link = front;
Front = temp;
Else
{
While(q → link ! = Null && q →link →priority < = item – priority)
q = q→ link;
temp → link = q → link;
q → link = temp;
}
```

Deletion:-

Delete operation wil be the deletion of 1st element of list because it has more priority then other elements of queue.

```
    If (front = = NULL)
    Printf("Queue underflow"))
    Else
    {
    Temp = front;
    Printf("deleted item is %d", temp →info):
    Front = front  → link
    Free (temp);
    }
```

Chapter – 6

Recursion :-

Recursion is defined as defining any thing interns of itself. Recursion is used to solve problems involving iteration s in reverse order.

Recursive functions:-

A function calls itself repeatedly.

Recursive program will not continue infinitely. A recursive procedure must have the following two properties.

a. There must be the certain criteria, called base criteria fro which the procedures does not call itself.
b. Each time the procedure does call itself, it must be closer to base criteria.

***Syntax**:-*

```
Main()
{
----------
-----------
        Function();
______________
}
        Function()
{__________________
        Function();
}
```

/* Find factorial of any number */

```
Main(()
{
Int no, fact;
Printf("Enter no");
Scanf("%d", &no);
Fact = factl(no);
Printf("Factorial no = %d", fact);
}
Int factl(int n)
{
If (n = = 0)
        Return(1);
Else
        Result = n* fa t (n-1));
        Return (result);
}
```

Output process:-

no =4

result = 24

after every call function will return the value to the previous function

fact l(u) = u* fact l(3)

fact l(3) = 3 * fctl(2)

factl(2) = 1 * fact(0)

fact(0) =1-

Here, steps for calling function are finished new each function will return the value to it's previous functions:

Factl(2) = 2 * fact (1) =2

Factl(3) = 3 * fact (2) = 6

` factl(4) = 4 * fct (3) = 4*6 = 24

		Fact(1) = 1				
		2 * fact (1)	2 * fact(1)	2 *1 = 2		
	3*fact(2)	4*fact(3)	4 * fact(2)	3 * fact (2)	3*2 = 6	
4*fact (3)	4*fct (3)	4*fact(3)	4*fact(3)	4* factl(3)`	4 * factl(3)	4*6 =24= 24

Need for recursion:-

There must be a terminating condition for problem which you want to solve with recursion. This condition is called the base criteria for that problem.

There must be an if condition in the recursion routing, this if clause specifies the terminating condition for the recursion.

Reversal in the order of execution is the characteristic of every recursion. Problem.

Every time a new recursive calls is made, a new memory space is allocate to each automatic variables used by the recursive routine.

The duplicated values of the local variables of a recursive call are pushed on to the stack with it's respective call & all these value are available to the respective function when it is popped off from the stack.

Disadvantage:-

It consume more storage space because the recursive calls along with automatic variable are stored in stack.

The computer may run out of memory if recursive calls are not cheked.

It is not more efficient in turms of speed & execution time.

Doesn't offer any concrete advantage over non recursive functions.

If proper precautions are not taken, recursion may result non terminating iterations.

Removal of recursion through iteration:-

Using recursion:-

```
Int factorial (int no)
{
Int fact = 1;
If (no>1)
Fact = no * factorial (n -1);
Return fact;
```

Without using recursion:-

```
Int factorial (int no)
{
```

```
Int fact =1
For(int i=no; i>1;I - -)
Fact = fact * I;
Return fact;
;
```

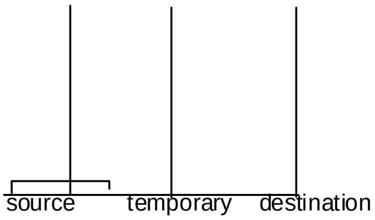
Tower of Hanoi (TOH):-

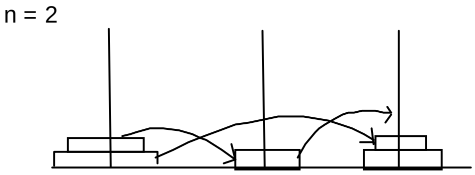
It is a game problem. In this case the disk is moved from one 0pillar to another pillar with the use of temporary pillar.

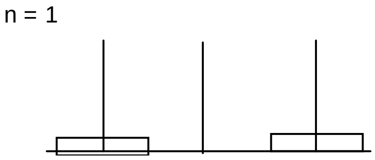
Rule:-

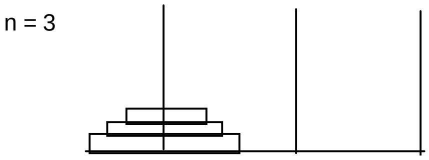
We can move only one disk from one pillar to another at a time.

Large disk can ‘t be can not be placed on smaller disk.

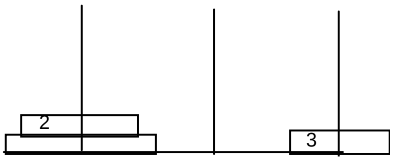




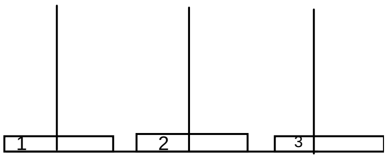




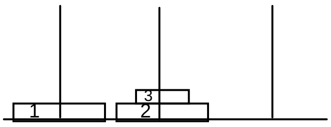
Step:- I



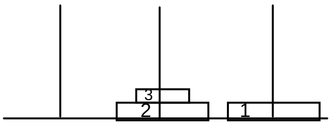
Step:- II



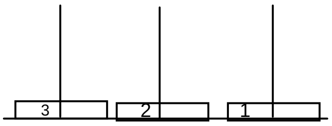
Step:- II



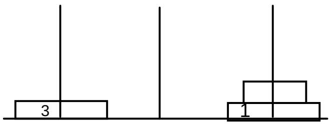
Step:- IV



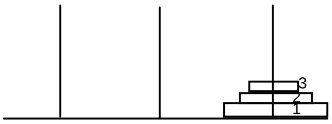
Step:- V



Step:- VI



Step:- VII



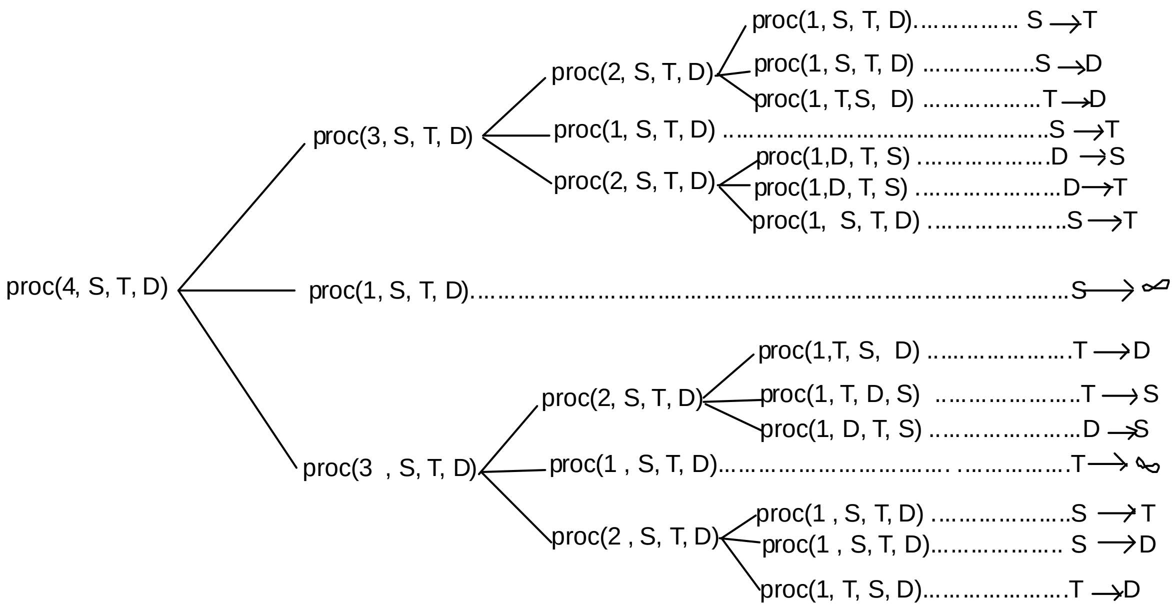
Algorithm:-

Move upper 0-1 disk from source to temporary.
move largest disk from source to destination.
move n -1 disk from temporary to destination.

Problem:-

Proc(N -1, S, D, T)
Proc(1, S, T, D)
Proc(N-1, T, S, D)]
For n = 1

		Proc(1, S, T, D)
		Proc(1, S, D, T)
Proc (3, S,T,D) then,	proc(2,S,D,T)	proc(1,D, S,T)
	Proc(1,S, T, D)	S →D
	Proc(2, S, T, D)	proc(1, T, D,S)
		Proc(1, T, S, D)
		Proc(1, T, D, S)


proc(4, S, T, D)
proc(3, S, T, D)
proc(2, S, T, D)
proc(1, S, T, D)…………… S ⟶T
proc(1, S, T, D) ……………..S ⟶D
proc(1, T,S, D) ………………T⟶D
proc(1, S, T, D) ………………………………………..S ⟶T
proc(2, S, T, D)
proc(1,D, T, S) ………………..D ⟶S
proc(1,D, T, S) …………………D⟶T
proc(1, S, T, D) ………………..S⟶T
proc(1, S, T, D)……………………………………………………………………………S⟶ ∞
proc(3 , S, T, D)
proc(2, S, T, D)
proc(1,T, S, D) …………………T ⟶D
proc(1, T, D, S) …………………T ⟶S
proc(1, D, T, S) …………………..D ⟶S
proc(1 , S, T, D)…………………………. ……………T⟶ ∞
proc(2 , S, T, D)
proc(1 , S, T, D) ………………..S ⟶T
proc(1 , S, T, D)……………… S ⟶D
proc(1, T, S, D)…………………T ⟶D

Translation from prefix to postfix using recursion:-

Algorithm:-

⇒ In the prefix string is a single variable, it is it's own postfix equivalent.

⇒ Let operator be the first operator of the postfix string.

⇒ Find the first operand opend1, of the string, convert it to postfix call it post 1.

⇒ Find the second and 2, convert & call it post 2.

⇒ Concatenate post 1, post 2 & op (operand)

Convert into postfix from the given string ** + AB - - + CDEF

Step	**Symbol**	**operator**	**postfix**	
1	*	OP1		
2	*	OP2		
3	+	OP3		
4	A		Post + AB + ⇒post 21	
5	B		Post 32	
6	-	OP4		
7	+	OP5		
8	C	Post 51 ⇒ CD +	Post 41 ⇒	⇒post 22
9	D	Post 52		
10	E		Post 42	
11	F			post 12

Chapter :-7

Definition:-

A tree is as non linear data structure in which items are arranged in a sorted sequence, It is used to represent hierarchial relationship existing among several data items. Each node of a tree may or may not pointing more than one node.

Theoretic definition of tree:-

It is a finite se of one or more data items (nodes) such that

there is a special data item called the root.

It's remaining data items are portioned into no of mutually exclusive subsets, each of which is it self a tree and they are called subtrees.

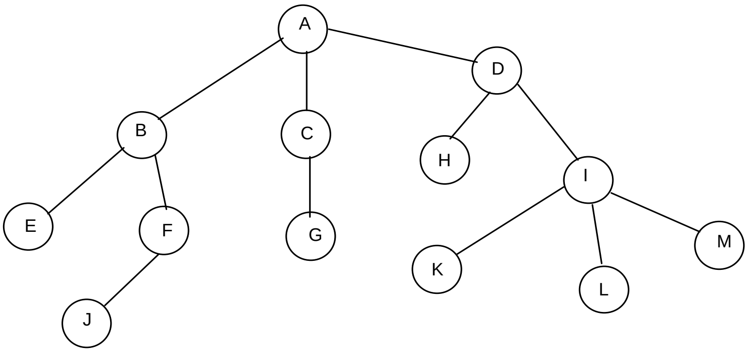


Fig. Tree

Tree terminology:-

Root:-

The first node in a hierarchial arrangement of data items is root.

Node:-

Each data item in a tree is called a node.

Degree of a node:- It is the no. of subtrees of a node in a given tree from fig.

Here, degree of node A = 3

Degree of node D = 2

Degree of node F = 1

Degree of node I = 3

Degree of a tree:-

It is the maximum degree of nodes in a given tree here degree of tree is 3.

Terminal node:-

A node with degree 0 is terminal node here, SIGHXLM are terminal node.

Non terminal node:-

Any node except the root whose degree is not zero is call non terminal node.

Siblings:-

The children nodes of a given parent nodes are called siblings. E and F are siblings of parent node B and K, L, M are siblings of parents node I & so on.

Level:-

The entire tree structure is leveled in such a way that the root rode is always at level zero. Then it's immediate children are at level 1 and there immediate children are at level 2 and so on up to the terminal nodes. Here, the level of a tree is 3.

Path:-

It is a sequence of consecutive edge from the source node to the destination node.

Edge:-

Is it is a connecting line of two nodes i.e. line drawn from one node to another.

Depth or height:-

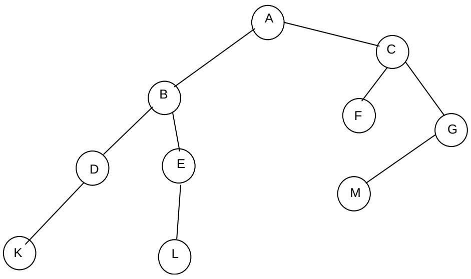
It is the maximum label of may node in a given tree here, the depth of a tree is 4.

Forest:-

It is a set of disjoint trees. In a given tree if we remove it's rot then it becomes forest.

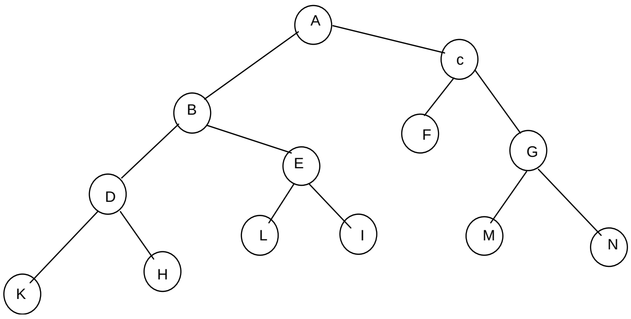
Binary tree:-

A binary tree is a finite set of data items which is either empty or consist of a single item called the root and two disjoint binary trees the lefts subs tree and right subtree. In binary tree the maximum degree of any node is almost 2 i.e. each node having 0, 1. or 2 degree node.



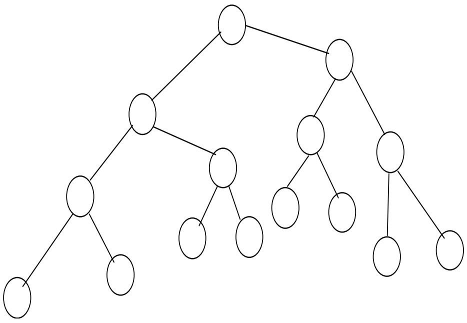
Strictly Binary tree:-

If every non terminal node in a binary tree consist of anon empty left sub tree and right sub tree then such tree is called strictly binary tree.

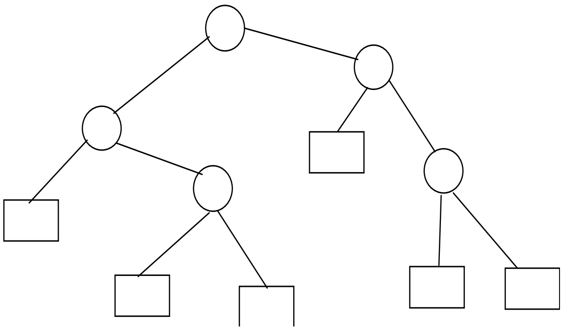


Completely binary tree:-

Completely binary tree of depth ‘d’ is the binary tree of depth d that contains exactly 21` nodes at each level l between zero and d i.e. in a complete binary tree there is exactly one node at level zero, 2 node at level 1, 4 node at level 2 and so on.

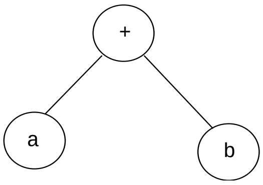


Extended binary tree:- (2-tree)



A binary tree is called extended binary tree if every node of tree has zero or two children. It is called 2- tree.

Algebraic expression representation tree.



Any algebraic expression can be represented in binary two in which left and right child represent operand of he expression and parent of the child repressing the operator.

e.g, a + b

Algorithm:-

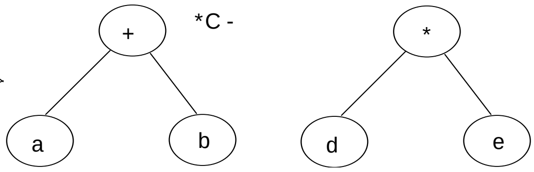
Note the order of precedence. All expression in parenthesis one to be ealuated first.

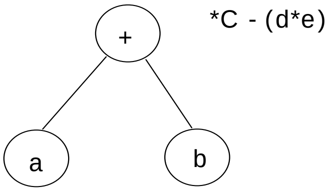
Exponential will come next.

Division & multiplication will be the next in order of precedence.

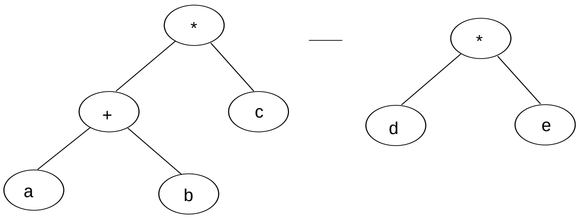
Subtraction and addition will be the last to be proceeded.

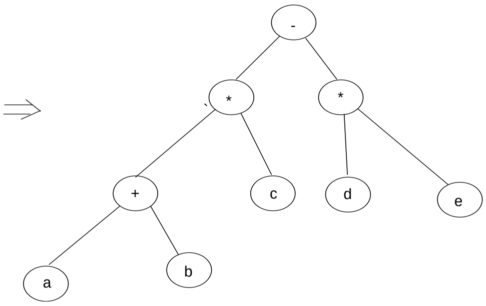
⇒





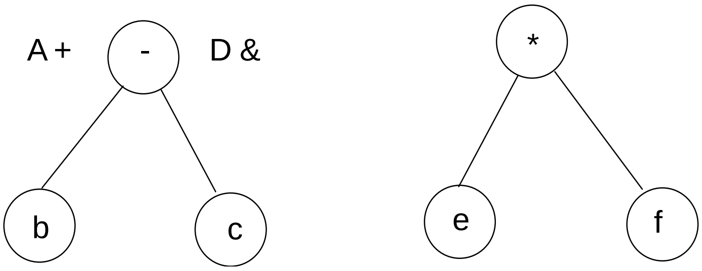
(a+b)* C – (d * e)

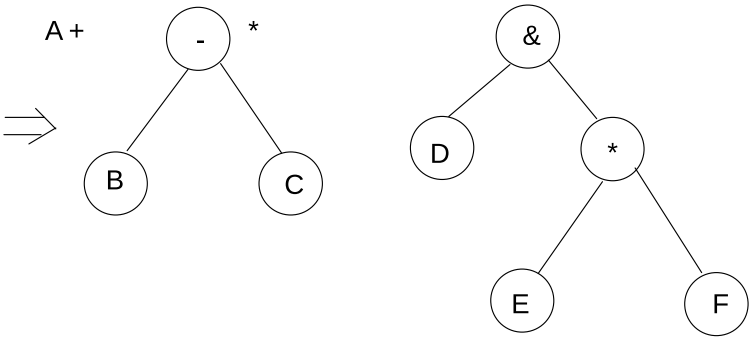


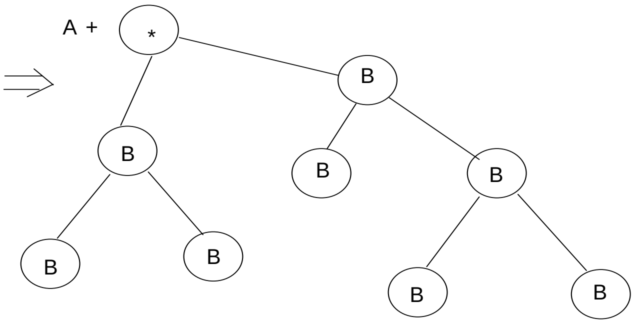


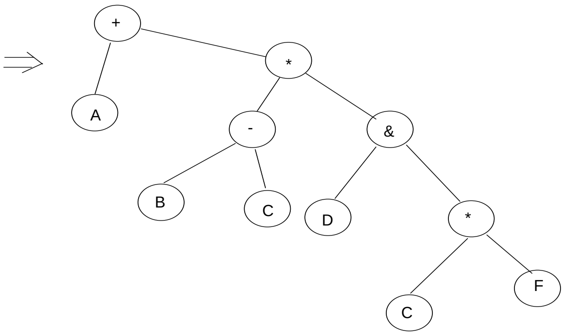
(ii) A + (B-C) * D & (E * F)

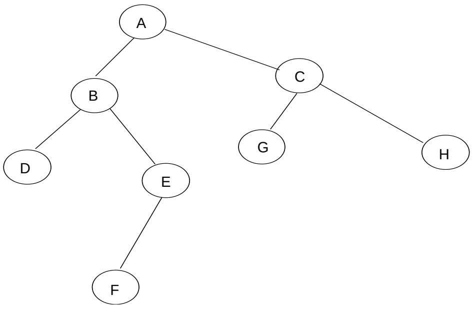
(iii) (A+B*C) & ((B+ B) * C)



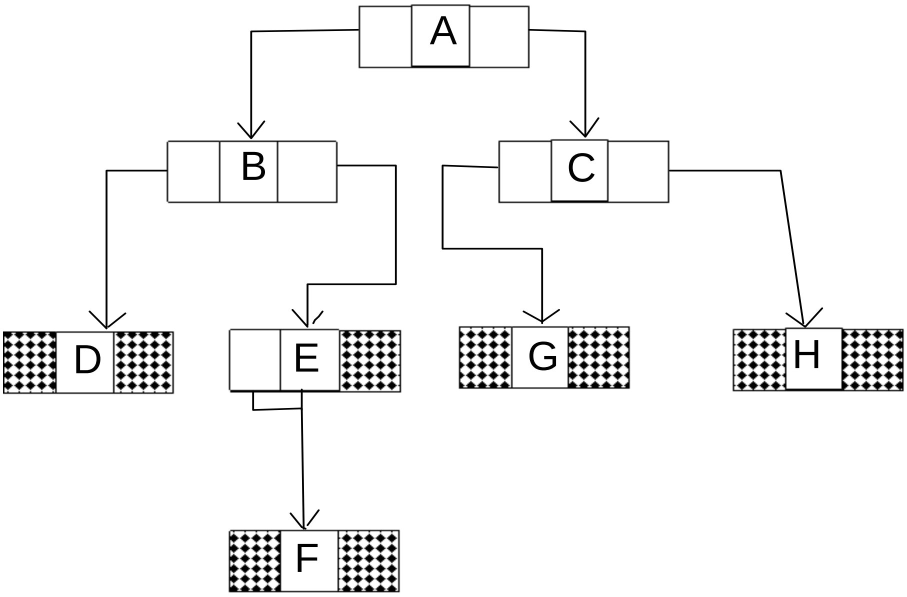








Representation of binary Tree:-



Structure:-

```
        Struc node
                {
                Char data:
                Node * lchild;
                Node * rchild;
                }
```

Binary Tree Traversal:

While traversing a tree root is denoted by 'N' left subtree as 'R'. Then there will be six combination of traversal i.e. NRL, NLR, LRN< LNR< RNL & RLN

NLR	pre ordered traversal
LAN	post order traversal
LNR	Inorder

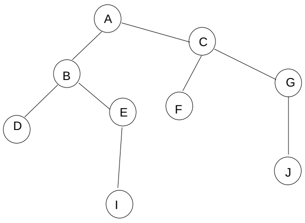
Pre ordered traversal:- NLR

Visit the root node.

Traverse the left sib tree of root in pre order,"

Traverse the right subtree of root in pre order.

```
Pre order (ptr)
Struct node * ptr;
{
If (ptr ! = NULL)
{
Printf("%d", ptr→ data);
Pre order (ptr → lchild);
Pre order (ptr → rchild);
}}
```



Preorder:-

A B D E I C F G I

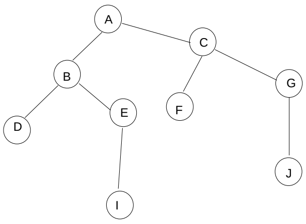
(ii) in order Traversal (LNR)

Travere the left subtree of root in order.

Visit the root.

Traverse the right subtree of root in order.

```
In order (ptr)
Struct node * ptr;
{
If (pre! = NULL)
{
In order (pre→ lchild)
Printf("%C", ptr →data);
In ordre (ptr→ rchild);
}}
```



Post order traversal:- (LRN)

Traverse the left subtree of root in post order.

Traverse the right subtree of root in post order.

Visit the root.

```
Post order (ptr)
Strct node * ptr;
{
If(pt! = NULL)
{
Post order (ptr →l child);
Post order (ptr → r child);
Printf("%c", ptr→ data);
}
```

}

Post order:-

D I E B F J G C A

Question:- construct a binary tree from given algebraic expression and show it's traversals.

(a – b * c) / (a + e/f)

A - /(a + e/f)

Preorder:-

/ - a * bc + d/e f

Inorder: a – b * / d + e – f

Post order: a b c * - d e f/ + /

Creating the tree from preorder and in order traversal:-

In pre order traversal, scan the nodes one by one and keep them inserting in tree.

In order traversal, put the cross mark over the node which has been inserted.

To insert a node in it's proper position in the tree we look at that node the in order traversal & insert it according to it's position with respect to the crossed nodes:-

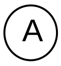
Preorder:- A B D G H E I C F J K

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

Step:-1 **Inset A**

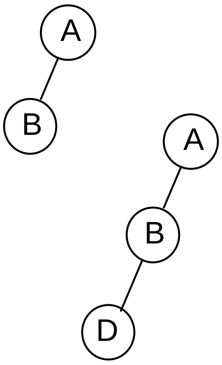
Pre: A B D G H E I C F J K

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



Step:-2 **Inset B**

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



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

Step:-2 **Inset D**

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

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

Step:-3 **Inset G**

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

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

Step:- 4 **Inset H**

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

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

Step:- 5 **Inset E**

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

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

Step:- 6 **Inset I**

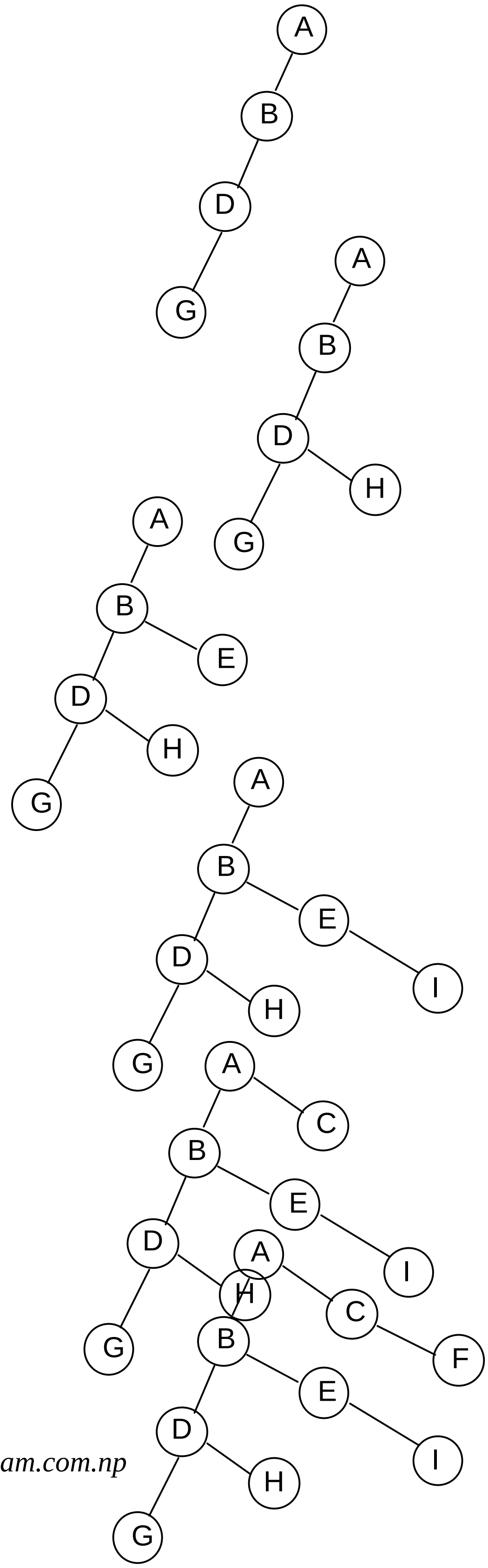
Preorder:- A B D G H E I C F J K

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

Step:- 7 **Inset C**

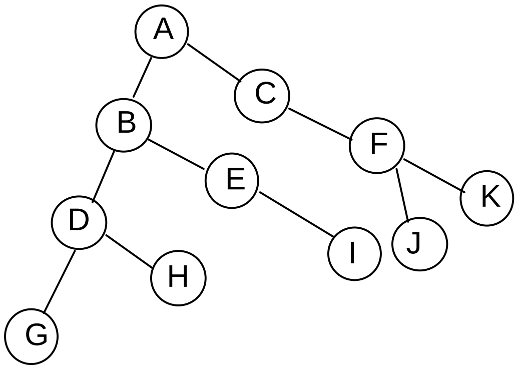
Preorder:- A B D G H E I C F J K

Inordre:- G D H B E I A C I F K



Step:- 8 **Inset C**
Preorder:- A B D G H E I C F I K
Inorer:- G D H B E I A C I C K

Step:- 9 **Inset I**
Preorder:- A B D G H E I C F J K
Inorer:- G D H B E I A C I F K

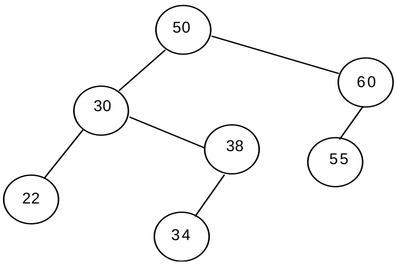


Binary search tree:-

A binary search tree is a binary tree in which each node has value greater than every node of left sub tree and les than every node of right sub tree.

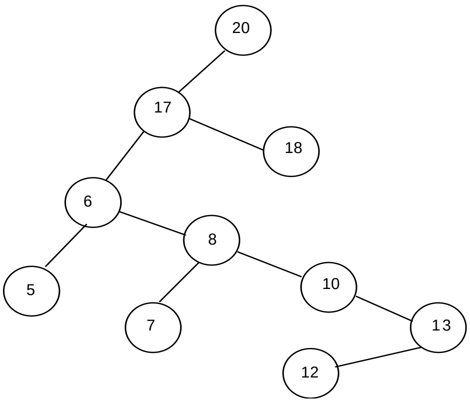
Question:- construct the binary tree from given data item.

50 30 60 22 38 55 34



Construct the binary tree:-

20 17 6 8 10 20 7 18 13 12 5
6



Delete operation in binary tree:-

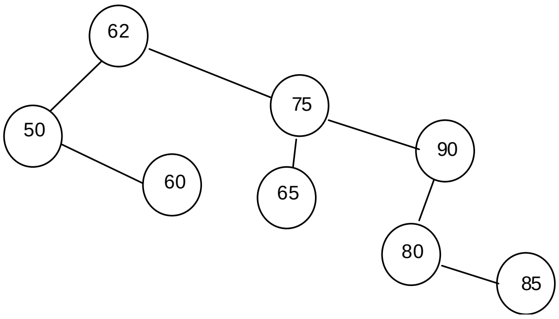
Leaves:- easiest to delete.

- set pointer from it's parent (if any) to NULL.

Node with one child; fairly easy.

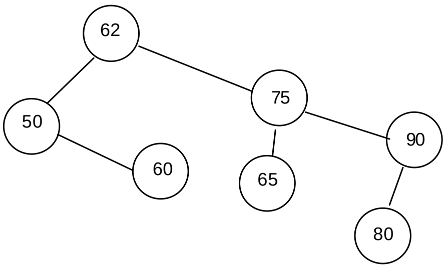
- Replace the node being deleted by it's in order successor.

e.g



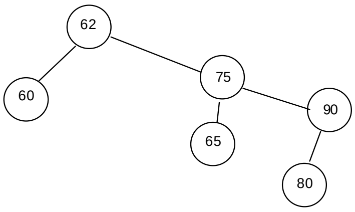
Delete 85:-

Since 85 has no children so, we can delete it., simply by giving NULL value to it's parents' right pointer thus the tree becomes



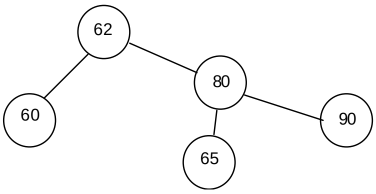
Delete item 50:-

Since, 50 has only one child so we can delete it by giving the address of right child to it's parents left pointer thus the tree becomes.



Delete items 75:-

In order : 60 62 65 75 80 90



Since, 75 has two children so 1st we delete the item which is in order successor of 75. Here, 80 is the in order successor of 75 we delete so by giving NULL value to it's parents left pointer after that we replace item 75 with item 80. We give the address of the left and right pointer of 75 to the left and right pointer of 80 & address of 80 to the right pointer of parent 75 which is 62. Thus the required tree is shown above.

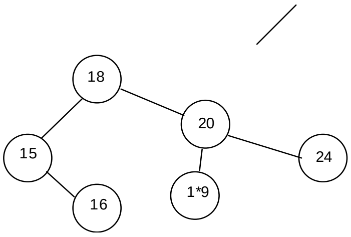
AVL Tree "Balanced Tree"

"G.M adel's son, vel' skill & E.M land is"

A AVL tree is a binary search tree where, height of left & right subtree at any node will be with maximum difference 1 each node of AVL tree has a balanced factor. Balance factor of a node is defined as the difference between the left subtree & right subtree of a node.

i.e. balanced factor = Height of left subtree – height of right sub tree.

A node is called right heavy or right high if height of it's right sub tree is one more than height of it's left sub tree. A node is called left heavy or high if height of it's left sub tree is one more than height of it's right sub tree. A node is called balanced, if heights of left or right sub tree are equal. The balanced factor will be 1 for left high & -1 for right high & 0 for balanced node. Thus in an AVL tree each node can have only three values of balance factor which are 1, 0 & -1.



Procedure for insertion of a node in AVL tree:-

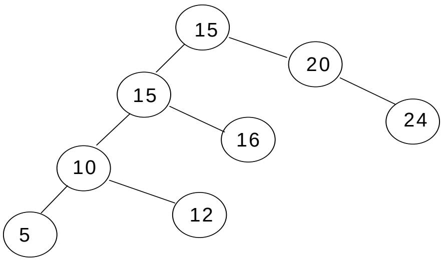
insert the node at it's proper place as in binary search tree.

Calculate the balanced factors of all the nodes on the path starting from the inserted nodes to the root node. If the value of balance factor of each node on this path is -1 or 1 then, the tree is balanced. If the balanced factor of nay node is more than 1 than the tree becomes unbalanced . The nod3e which is nearest to the inserted node 4 has absolute value of balance factor greater than 1, then it is marked as pivot node.

If the tree has become unbalanced after the insertion, then the tree is rotated about the pivot node to achieve all the properties of AVL treep.

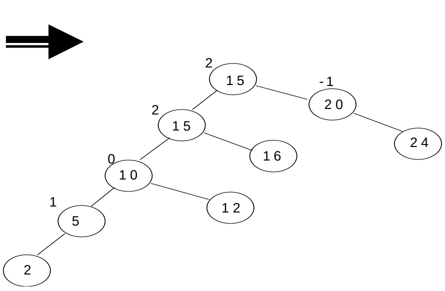
AVL rotations:-

Left to left rotation:-

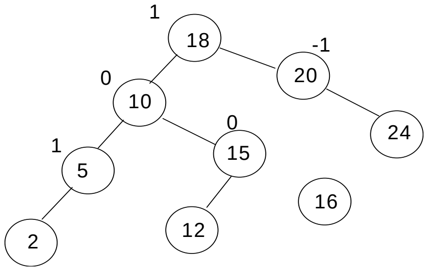


Insert 2.

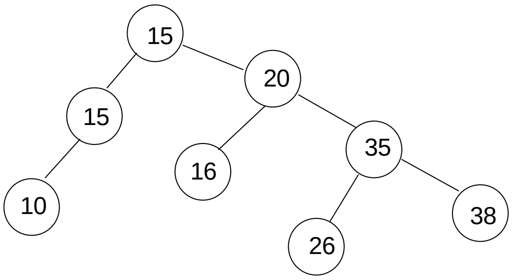
Pivot node



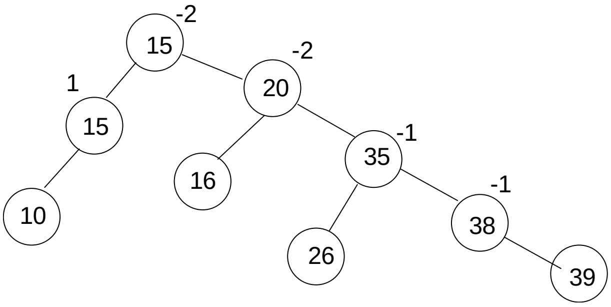
If the node is inserted on the left side of left subtree than we perform left to left rotation.



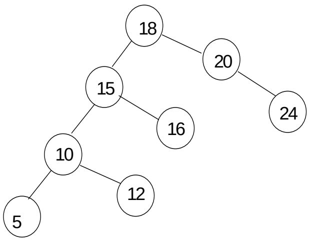
Right to right rotation:-

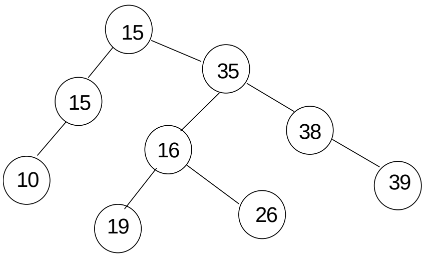


Insert 39



If the node is inserted on the right side of right sub tree than we perform right to right rotation.

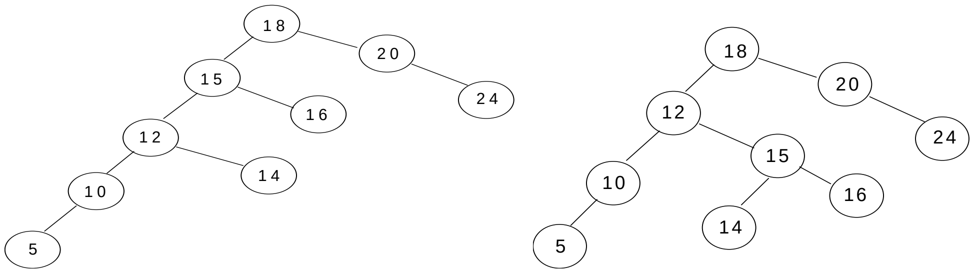




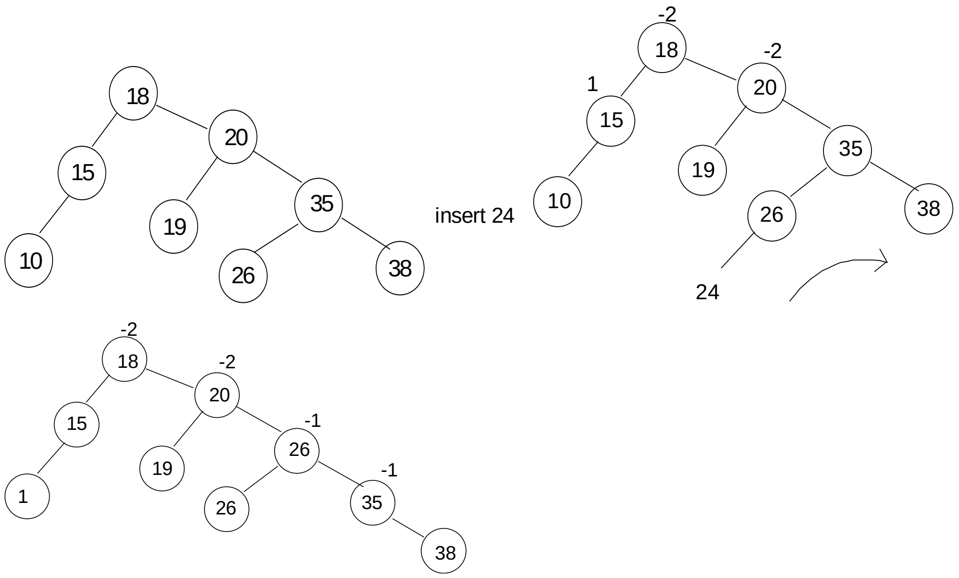
Left to right rotation:-

If the node is inserted on the right side of left sub tree then we perform left to right rotation.

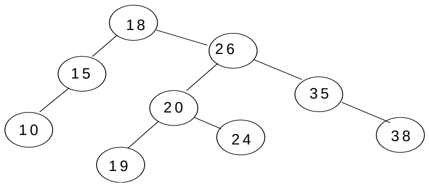
In this left to right rotation, the operation involves two steps 1st we perform right to right from next to pivot node then perform left to left.



Right to left rotation:-



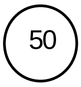
Rotate right to Right:-



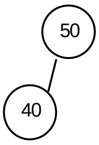
Construct an AVL tree from given data.

I/Ps : 50, 40, 35, 58, 48, 42, 60, 30, 33, 32

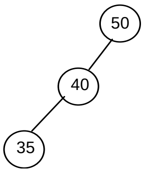
Insert 50



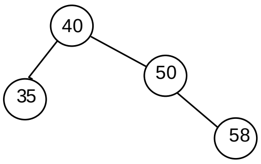
Insert 40



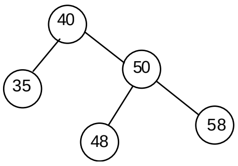
Insert 35



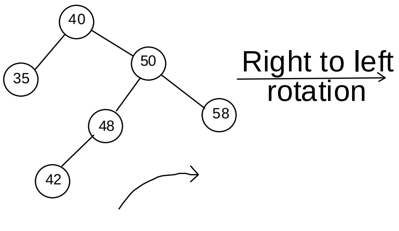
Insert 58

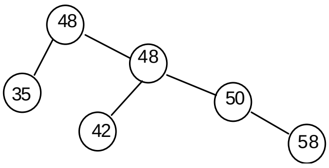


Insert 48

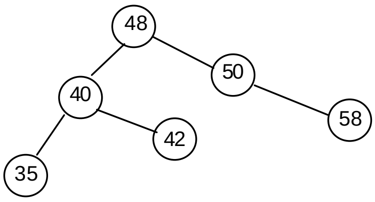


Insert 42

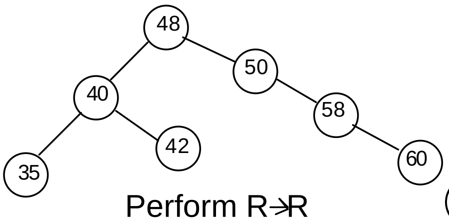




Step – I

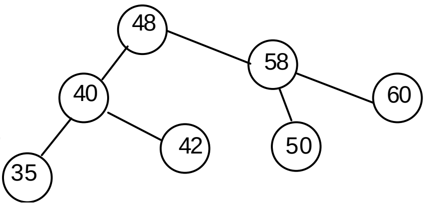


Step –ii

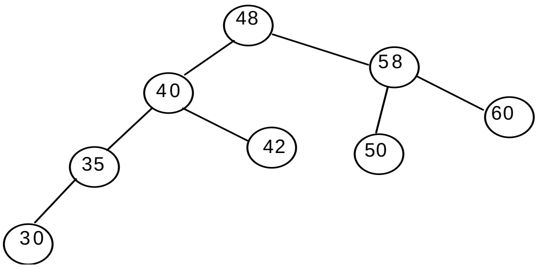


Insert 60

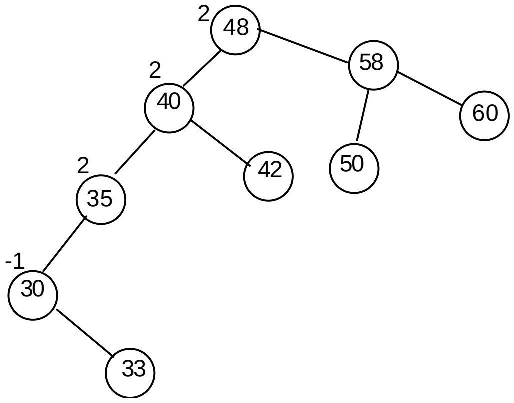
Perform R→R



Insert 30

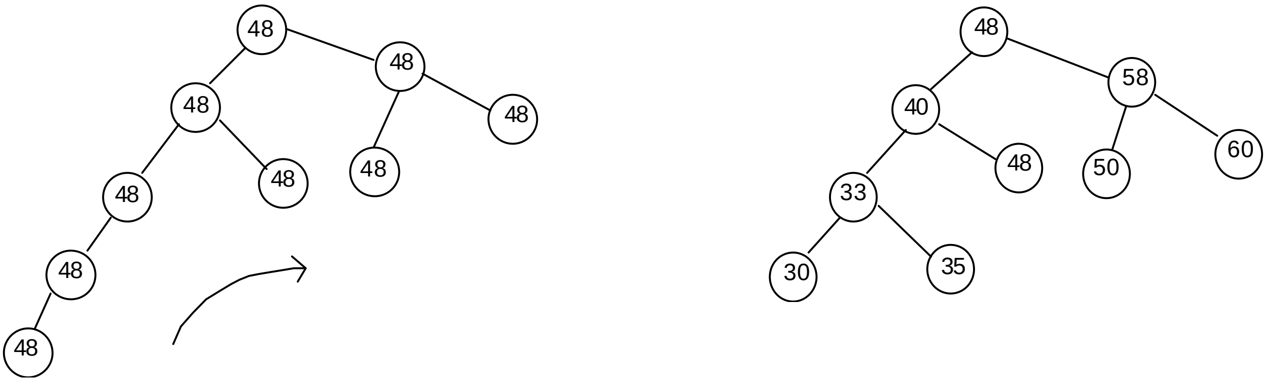


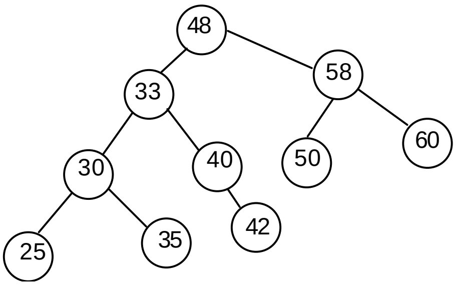
Insert 33



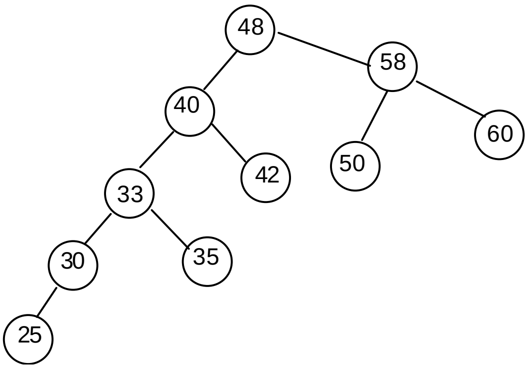
Step I

Step II





Insert 25



Huffman Tree:-

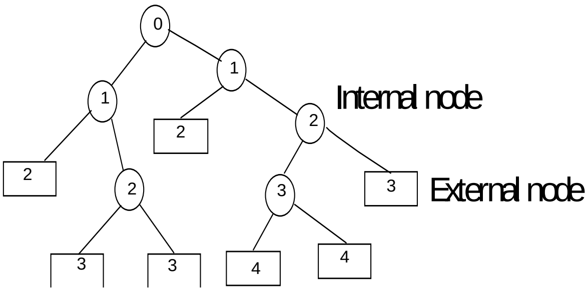


Fig:- extended binary tree

Total path length of internal nodes:-

PI = 0 + 1 + 2 + 1 + 1 + 2 + 3 = 9

Total path length of external nodes:

PE = 2 + 3 + 3 + 2 + 4 + 4 + 4 + 3 = 21

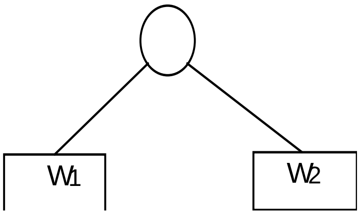
If external node has weight w, then weighted path length for external node is

$P = W_1P_1 + W_2 P_2 + \ldots\ldots\ldots W_nP_n$

Algorithm:-

Let us take there are n weights W_1, W_2, ………., ………., W_n

Take two minimum weights & create a subtree, suppose W_1 & W_2 are first two min weight then subtree will be



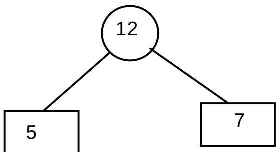
Now the remaining weight will be $W_1 + W_2$, W_3, W_4……W_n

Create all sub tree at the last weight.

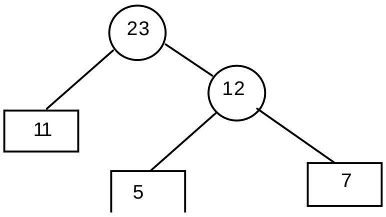
Q. Construct an extended binary tree by huff man method.

	A	**B**	**C**	**d**	**E**	**F**	**G**
Weight	**16**	**11**	**7**	**20**	**25**	**5**	**16**

Step:1 Taking two nodes with minimum weight 7 & 5,

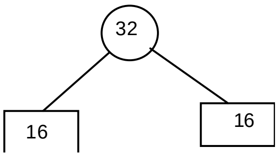


Step:2 taking two nodes with minimum weight 11 & 12



Now the elements are 16, 23, 20, 25, 16

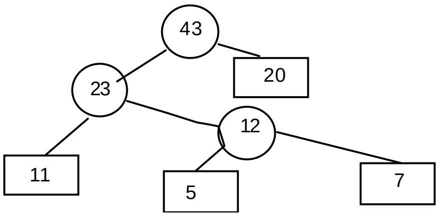
Step:3 Taking two nodes with minimum weight 16 & 16



Now,

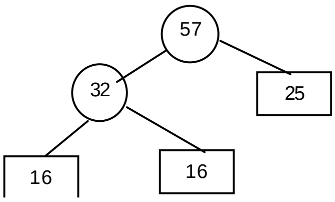
The elements are 32, 23, 20, 25

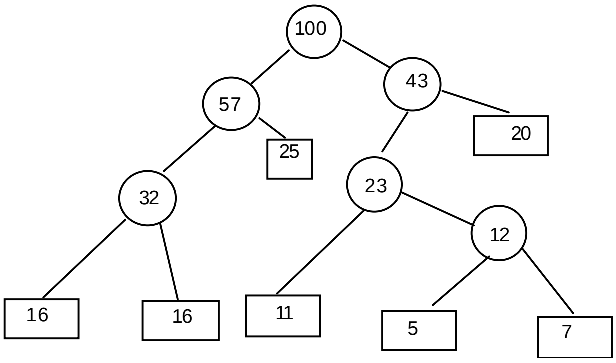
Step:4



Now the elements are 32, 43, 25

Step:5





Total path length:-

PI = 0 + 1 + 2 + 2 + 3 + 4 = 12

PE = 2 + 3 + 3 + 2 + 4 + 4 + 3 = 21

B – Tree "Balanced Tree"

It is also known as balanced sort tree.

The height of the tree must be kept to a minimum.

There must be no empty sub trees. Above the leaves of the tree.

The leaves of the tree must all be the same level.

All nodes except the leaves must have at least some minimum no of children.

B – Tree of order n can be defined as:-

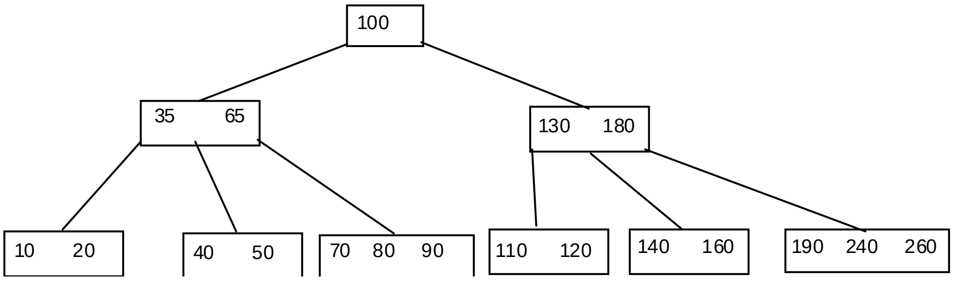
Each node has at least n+2 & maximum 'n' non empty children.

All leaves nodes will be at the same level.

All the leaf nodes contain minimum n-1 keys.

Keys are arranged in a defined order with in the node. All keys in sub tree to the left of the key are the procedure of the key and that on the right are successors of the key.

When a new key is to be inserted in to a full node, then split the nodes with the median value is inserted in the parent node. In case the parent node is root, a new node is created.



Each node at same leaves.

All non- leaf nodes have no empty subtree.

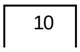
Keys 1 less than no. of their children.

Q. Construc a B – tree of order 5 inserting the keys:-

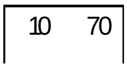
10, 70, 60, 20, 110, 40, 80, 130, 100, 50, 190, 90, 180, 140, 280.

Sol^{n}:-

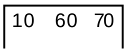
- Insert 10



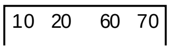
- Insert 70



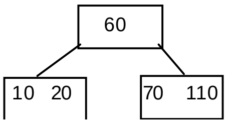
- Insert 60



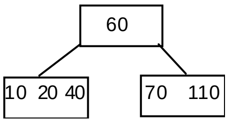
- Insert 20



- Insert 110

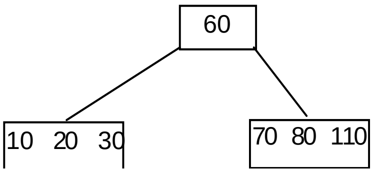


- Insert 40

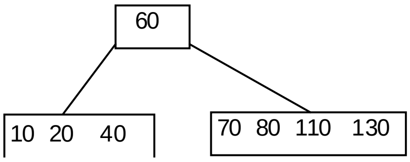


80, 130, 100, 150, 90, 180, 240, 30, 120, 140, 160

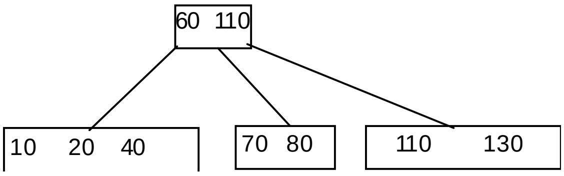
Insert 80



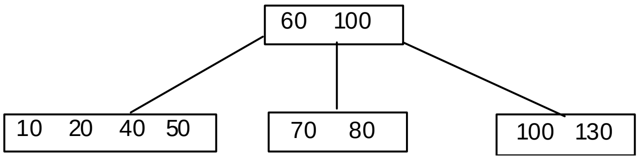
Insert 130

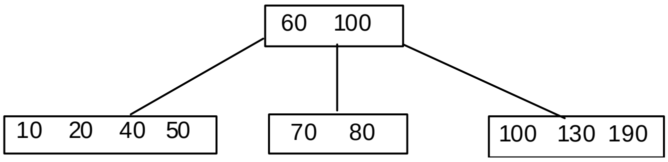


Insert 100

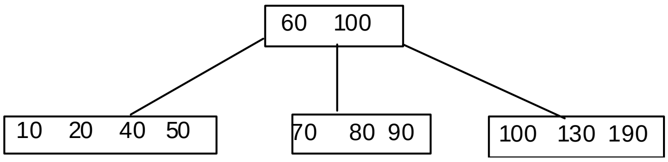


Insert 50

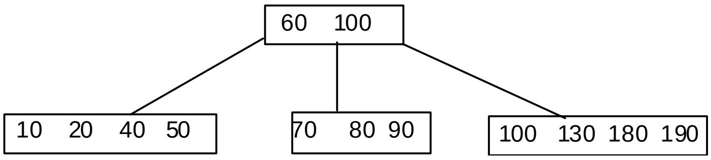




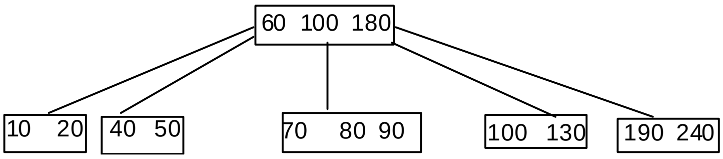
Insert 190



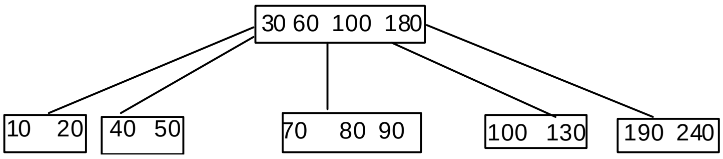
Insert 90



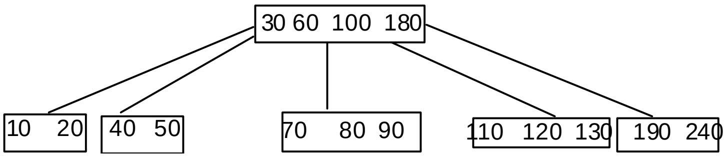
Insert 180



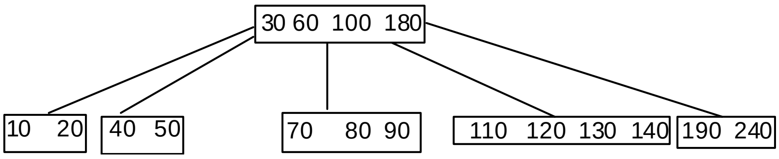
Insert 240



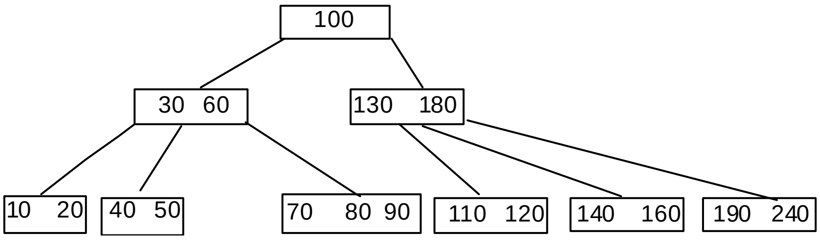
Insert 30



Insert 120



Insert 140



Insert 160

Deletion in B- tree:-

node is leaf node.

node is non leaf

-if node has more than minimum no. of keys than it can be easily deleted.

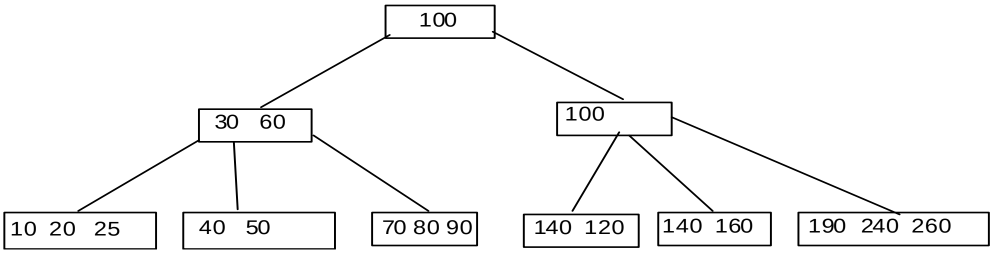
-if it has only minimum no. of keys, than first we see the no. of keys in adjacent leaf node. If it has more than minimum no. of keys then first key of the of the adjacent node will go to the parent node & key in parent node will be combined together in one node.

-If now parent has also less than minimum no. of keys then the same thing will be repeated until it will gate the node which has more than the minimum no. of keys.

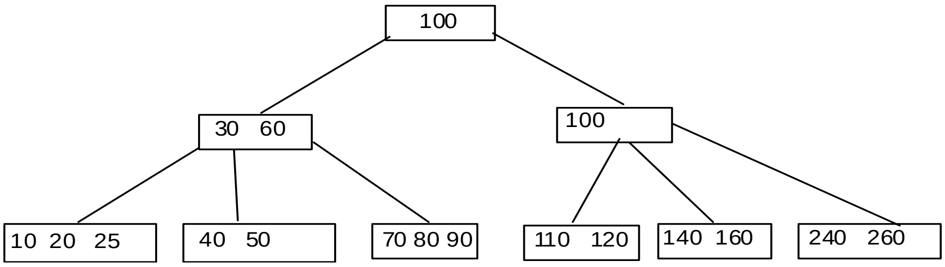
Node is non leaf:-

-In this case key will be deleted & it's predessor or successor key will condition it's place.

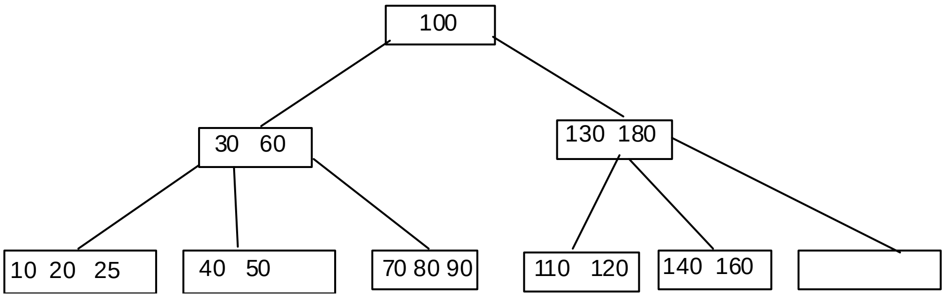
-If both nodes of predessor or successor key have minimum no. of keys then the rates of predessor & successor keys will be combine.



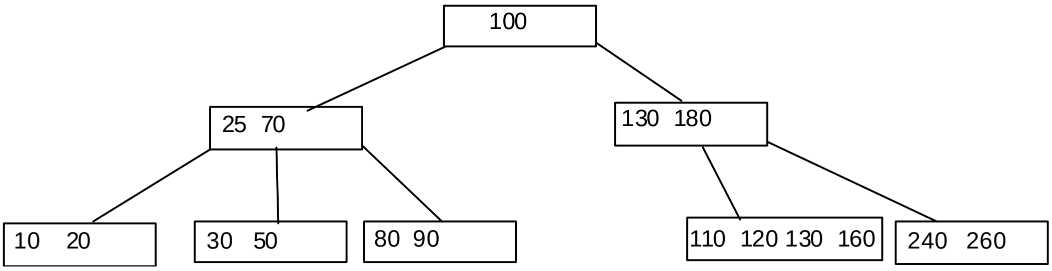
Delete 190



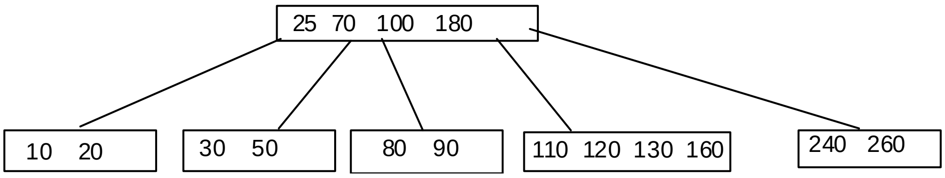
Delete 60



Delete 40



Delete 140



Question:- Construct a B- tree of order 5 I/P the element when keys are 659, 767, 702, 157, 728, 102, 461, 899, 920, 44, 744, 264, 384, 344, 973, 905, 999
Perform delete operation fro 44, 344, 920

Chapter – 8

Game tree:- 659, 767, 702, 157, 728, 102, 461, 899, 920, 44, 744, 264, 344, 973, 905, 999

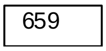
Given,

Maximum no. of key = n – 1 = 4 element

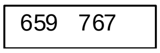
Minimum no. of element = n/2 = 2 element in each node.

Maximum no of children = 5

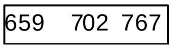
Insert 659



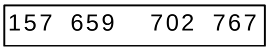
Insert 767



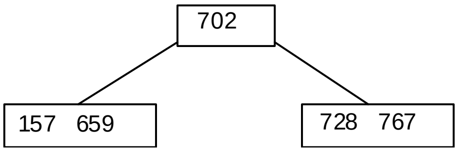
Insert 702



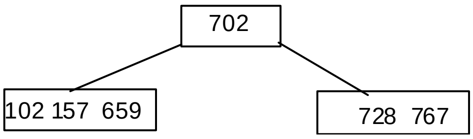
Insert 157



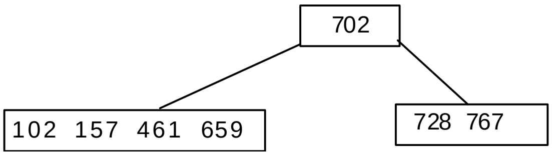
Insert 728



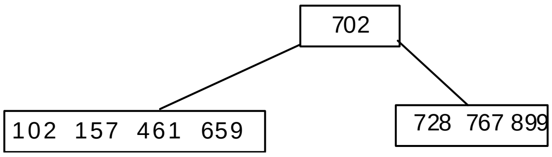
Insert 102



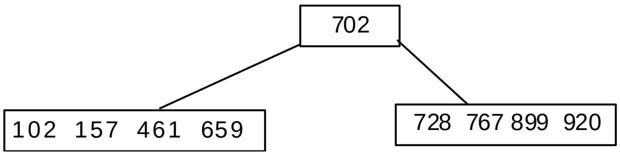
Insert 461



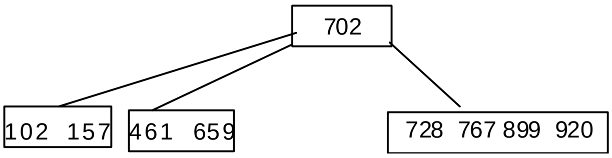
Insert 899



Insert 920



Insert 44



Insert 744

Insert 264 & 384

Insert 344

Insert 973 & 905 & 999

delete 44

delete 344 direct

Sorting:-

Sorting is storage of data in sorted order it can be in ascending or descending order.

Type

Internal sort

External sort

In internal sorting data i.e. going to be sorted will be in many memory. In external …….will be on auxiliary storage , tape floppy disk etc.

Sorting Technique:-

Insertion Sort:-

The insertion sort inserts each element in proper place if there are n-element in array and we place each element of array at proper place in the previously sorted element list.

Algorithm:-

Consider N elements in the array are

Pass 1 arr[0] is already sorted because of only one element.

Pass 2 arr[0] is inserted before or after arr[0]. So arr[0] & arr[i] are sorted.

Pass 3 arr[2] is inserted before arr[0], in between arr[0] & arr[1] or after arr[0] . so arr[0] , arr[1] & arr [2] are sorted.

Pass 4 arr[3] is inserted in to it's proper place in array arr[0], arr[1], arr[2], arr[3] & are sorted.

…………………

……………………

Pass N arr[N-1] is inserted in to it's proper place in array arr[0], arr[1], ……. Arr[N-1]. So arr[0], arr[N-1] are sorted.

Q. Trace algo. With the given data using insertion sort.

	82	42	49	8	92	25	59	52
Pass1	82	42	49	8	92	25	59	52
Pass 2	82	42	49	8	92	25	59	52
Pass 3	42	82	49	8	92	25	59	52
Pass 4	42	49	82	8	92	25	59	52
Pass 5	8	42	49	82	92	25	59	52
Pass 6	8	42	49	82	92	25	59	52
Pass 7	8	25	42	49	82	92	59	52
Pass 8	8	25	42	49	59	82	92	52
Pass 9	8	25	42	49	52	59	82	92

Selection Sort:-

Selection sort is the selection of an element & keepingit in sorted order. Let us take an array arr[0]……..arr[N-1]. First find the position of smallest element from arr[0] to arr [n-1]. Then interchange the smallest element from arr[1] to arr[n-1], then interchanging the smallest element with arr[1]. Similarly, the process will be for arr[0] to arr[n-1] & so on.

Algorithm:-

Pass 1:- search the smallest element for arr[0] ……..arr[N-1].
- Interchange arr[0] with smallest element
  Result : arr[0] is sorted.

Pass 2:- search the smallest element from arr[1],……….arr[N-1]
- Interchange arr[1] with smallest element
  Result: arr[0], arr[1] is sorted.
  …………………
  …………………

Pass N-1:-
- search the smallest element from arr[N-2] & arr[N-1]
- Interchange arr[N-1] with smallest element
  Result: arr[0]………… Arr[N-1] is sorted.

Q. Show all the passes using selecting sort.

	75	35	42	13	87	27	64	57
Pass 1	75	35	42	13	87	27	64	57
Pass 2	13	35	42	75	87	27	64	57
Pass 3	13	27	42	75	87	35	64	57
Pass 4	13	27	35	75	87	42	64	57
Pass 5	13	27	35	42	87	75	64	57
Pass 6	13	27	35	42	57	75	64	87
Pass 7	13	27	35	42	57	64	75	87

Buibble sort:- (exchange sort)

In bubble sort each element is compared with I't adjacent element. If the 1st element is large than the second one then the position of the elements are interchanged otherwise it is not changed.

Alg.

If N elements are given in memory then for sorting we do the following steps:-

Pass:-

1. first compare the 1st element and 2nd element of array. If 1st < 2nd then compare the 2nd with 3rd.

2. if 2nd > 3rd.

Then interchange the value of 2nd & 3rd.

now compare the value of 3rd with 4th.

similarly compare until N-1th element is compared with nth element.

Now, highest value element is reached at the Nth place.

Elements will be compared until N-1 elements.

Or

Repeat through step 1 to 5 for N-1 elements.

Q. Show all the passes using bubble sort.

13, 32, 20, 62, 68, 52, 38, 46

Pass 1.	13	32	20	62	68	52	38	46	
Pass II	2nd > 3rd		-interchange.						
	13	32	62	68	52	38	46		
Pass III	32<62		- no change.						
	13	32	20	62	68	52	38	46	
Pass IV	62<68		- no change.						
	13	32	20	62	68	52	38	46	
Pass V	68< 52		- interchange.						
	13	32	20	62	52	68	38	46	
Pass VI	68 <38		- interchange						
	13	32	20	62	68	38	68	46	
Pass VII	68<46; interchange								
	13	32	20	62	52	38	46	68	

Pass 1 (i)	13	20	32	62	52	38	46	68
	13< 20 ; no change							
(ii)	13	20	32	62	52	38	46	68
(iii)	13	20	32	62	52	38	46	68
(iv)	13	20	32	52	62	38	46	68
(v)	13	20	32	52	38	62	46	68
(vi)	13	20	32	52	38	46	62	68
(vii)	13	20	32	52	38	46	62	68

Pass:- 3

(i) 13	20	32	52	38	46	62	68	;13<20 no change
(ii) 13	20	32	52	38	46	62	68	;20<32 “
(iii)13	20	32	52	38	46	62	68	;32<52
(iv)13	20	32	38	52	46	62	68	;52< 38 change
(v)13	20	32	52	38	46	62	68	

Pass :- 4

O/P 13 20 32 38 46 52 62 68

Q. Quick sort (partition exchange sort)

Quick sort is base on divide and conquer algorithm . In thi9s we divide the original list in to two sub list . we choose the item from list called key or pivot from which all the left side of elements are smaller & all the right side of elements are greater than that element. Thus we can create two list, one list is an the left side of pivot and 2nd list is an right side of pivot. Similarly, we choose the pivot for dividing the sub-list until there are two or more elements in the sub list.

Algorithm:-

(i) take the first element of list as pivot.

(ii) Place pivot at ht proper place in list.

For placing pivot at it's proper place it follows following steps

→ Compare the pivot element one by one from right to left for getting the element which has value less than pivot element.

→ Interchange the element with element.

→ Now, comparison will start from the interchange element position from left to right for getting the element which has higher value than pivot.

→ Repeat the same process until pivot is at it's proper position.

(iii) Create two subsists left & right side of pivot.

(iv) Repeat the same process until all elements of first are at proper position in list.

Q. Show all the same process until all elements quick with the following.

48 29 8 59 72 88 42 65 95 19 82 68

Soln:- Pass(i)

48 29 8 59 72 88 42 65 95 19 82
68

(ii) interchange

19 29 8 59 72 88 42 65 95 48 82
68

(iii) large element interchange

19 29 8 48 72 88 42 65 95 59 82
68

(iv) interchange

19 29 8 42 72 88 48 65 95 59 82
68

(v) interchange

19 29 8 42 48 88 72 65 95 59 82
68

For sub list 1.

(i) 19 29 8 42 search small element

(ii) 8 29 19 42

Search large element

(iii) 8 19 29 42

Sublist sublist 22

Now, consquaring sublist 11 & sublist 12 with pivot (19) we get,

8, 19, 29, 42

For sublist 2

88 72 65 95 59 82 68
68 72 65 88 59 82 88
68 72 65 88 59 82 95
68 72 65 82 59 88 95

Sub list 21

For sub list 21.

Similarly, perform above process

Merge Sort:-

IN merge sort we take a pair of conjugative array elements then merge them in sorted array & take adjacent pair of array elements & so on. Until all elements of array are in single list.

e.g.	5	8	89	30	92	64	4	21	56
pass1	5	8	89	30	92	64	4	21	56
pass2	5	8	30	89	64	92	4	21	56
pass3	5	8	30	89	4	21	64	92	56
	4	5	8	21	30	56	64	89	92

Algorithm :- at last:-

Shell sort (Diminishing increment sort)

Shell sort is an improvement an insertion sort. In this case we take on itme at a particular distance (increments) than we compare items which are far apart and then sort them. After- words we decrease the increments & repeate this process again. At last we take the increment one & sort them with insertion sort.

Procedure:-

Let us take an array from arr[0], arr[1]……… arr[N-1] & take the distance 5 (say) for grouping together the items then in terms will grouped as:

First: arr[0], arr[5], arr[10],……….

Second: arr[1], arr[6], arr[11],………

Third: arr[1], arr[7], arr[12],……..

Fourth: arr[9], arr[8], arr[13]…….

We can see that we have to make the list equal to the increments & it will cover all the items from arr[0]………arr[N-1].

First sort this list with insertion sort then decrease the increments & repeat this process again.

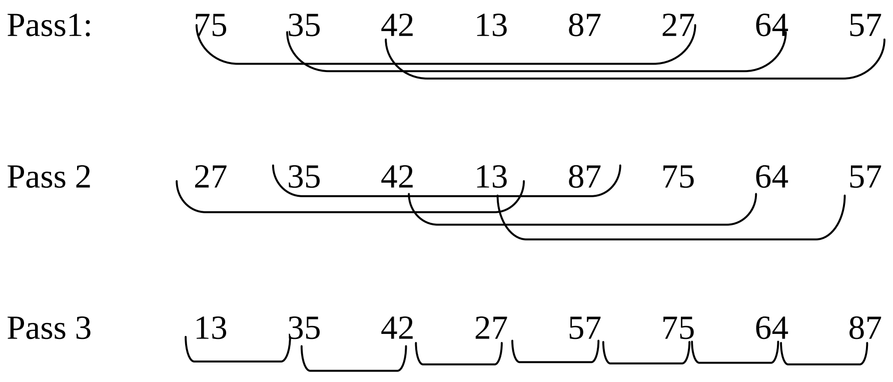
At ht end, list is maintained with increment 1 & sort them with insertion sort.

Question:- Show all the passes using shell sort with following list:-

75 35 42 13 87 27 64 57

Sol^{n}:-

Consider increments 5


Pass1: 75 35 42 13 87 27 64 57
Pass 2 27 35 42 13 87 75 64 57
Pass 3 13 35 42 27 57 75 64 87

⇒ 13 27 35 42 57 64 75 87

Explanation:-

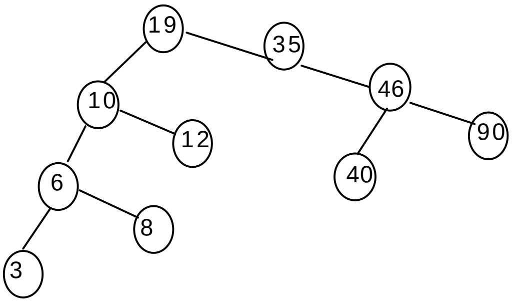
Binary Tree sort:-

Create binary search tree.

Find in order traversal of binary tree.

Question:-

19 35 10 12 46 40 6 90 3 8



[do step by step]

In order traversal

3 6 8 10 12 19 35 40 46 90

Heap Sort:-

Heaps:-

A heap is a binary tree that satisfied the following propertiers:-

→ shape property

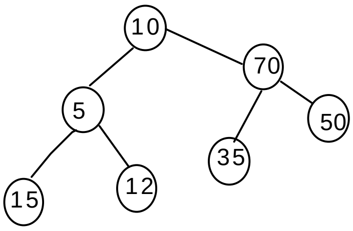
→ Order property.

By the shape property we mean that heap must be a complete binary tree where as by order property we mean that for every node in the heap the value store in the heap node is greater than or equal to the value to the value an each of its' children. A heap that satisfied this property is known as max heap.

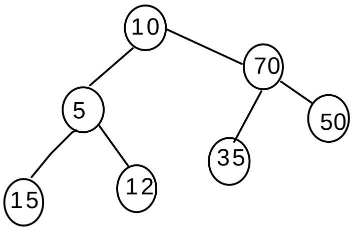
However, If the order property is such that for every node. In the heap the value stored in that node is less than or equal to the value in each of it's children. That heap is known as minimum heap.

e.g **create the heap tree.**

10 5 70 15 12 35 50

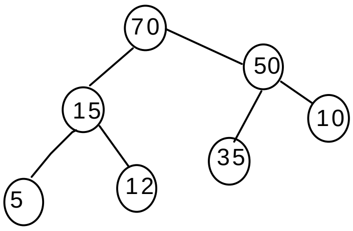


- Here root no must be greater than children here 5< 15 so, we inter change it's position.



- Now, in position 2, 5, & 6 the 70 is greater than 35 & 50. so we can't interchange.

- On the other hand the.



Heap sort:-

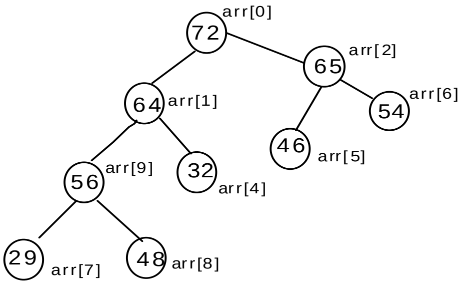
The elements of the heap tree are represented by an array . The root is the larget element of heap tree. As it is maintained in the array. The largest element of heap tree. As it is maintained in the array. The largest value should be the last element of array. For heap sorting root is deleted till there is only one element in the tree.

Steps:-

Replace the4 root with the last node of heap tree.

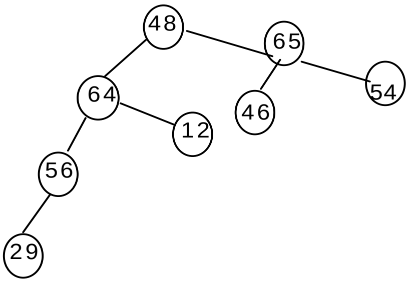
Keep the last node at proper position.

Repeat step 1 & 2 until there are only one root node in the tree.



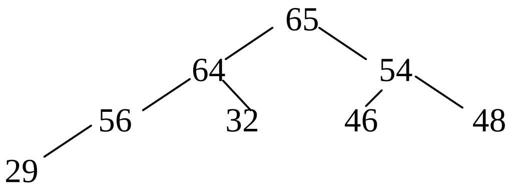
	0	1	2	3	4	5	6	7	8
arr	72	64	65	56	32	46	54	29	48

Step:-1 Root node is delete and this root node is replaced by last node and the previous value of root node is placed in proper place in array.

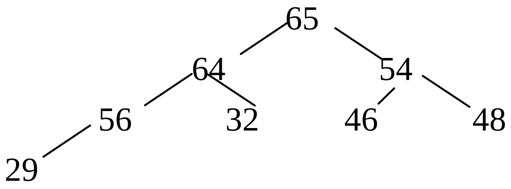


48	64	65	56	32	54	29	72

Here, root node is less than 65. so we interchange the position to make heap tree.



Again , 48< 54



Now, which is in Heap tree form.

65	64	54	56	32	46	48	29	72

Again , delete root node & put 29 in root node.

29	64	54	56	32	32	46	48	65	72

29	64	54	56	32	32	46	48	65	72

65	64	54	56	32	46	48	29	72

Radix sort:-

In radix sort, we sort the item in terms of it's digits. If we have list of nos. then there will be 10 parts from 0 to 9 because radix is 10.

Algo

consider the list n digit of nos, then there will be 10 parts from 0 to 9.

in the first pass take the nos in parts on the basis of unit digits.

In the second pass the base will be ten digit.

Repeat similarly for n passes for n digits.

Show all the passes using radix sort.

233 124 209 345 498 567 328 163

Pass 1

Numbers	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
233				233						
124					124					
209										209
345						345				
498									498	
567								567		
328									328	
163				163						

Pass 2

	233	163	124	345	567	498	328	209		
Number	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
233				233						
163				1			163			
124			124							
345					345					
567							567			
498										498
328			328							
209	209									

209, 124, 328, 233, 345, 163, 567, 498.

Taking hundred position:-

Number	0	1	2	3	4	5	6	7	8	9
209			209							
124		124								
328				328						
233			233							
345				345						
163		163								
567						567				
498					498					

Hence, required sequence is

124 163 209 233 328 345 498 567

Chapter – 9

Searching

Definition:-

Searching is used to find the location whether element is available or not. There are various kinds of searching techniques.

sequential search:-

simplest technique for searching on unordered table for particular record is to scan each entry in sequential manner until the desired record is found.

If search is successful then it will return the location of element otherwise it will return failene notification.

Consider, sequential search in array.

Arr	[0]	[1]	[2]	[3]	[4]	[5]	[6]
	10	20	30	40	50		

Algorithm:-

Put a unique value at the end of array. Then.

```
(i)    index =0
(ii)   scan each element of array one by one.
(iii)  (a) If match occurs then return the index value.
       (b) otherwise index = index +1
(iv)   Repeat the same process until unique value comes in scanning.
(v)    Return the failure notification.
```

Consider sequential search in linked list.

Algorithm:-

Take a pointer of node type and initialize it with sort

Ptr = start

Scan each node of the linked list by traversing the list with the help of ptr.

Ptr = ptr → link;

If match occur then return.

Repeat the same process until null comes in scanning.

Return the failure notification.

Performance analysis:-

Number of key comparison taken to find a particular record

Average case :	(n+1)/2
Warpe case:	n+1

Best case:- if desired record is present in the 1st position of search table i.e. only one comparison is made.

Binary search:-

The sequential search situation will be in worse case i.e. if the element is at the end of the list. For eliminating this problem one efficient searching technique called binary search is used in this case.

the entries are stored in sorted array.

The element to be searched is compared with middle element of the array.

a. If it is less than the middle element then we search. It in the left portion of the array.
b. If it is greater than the middle element then search will be in the right portion of the array.

The process will be in iteration till the element is searched or middle element has no left or right portion to search.

Algorithm:-

```
Binary search (K.N, x)
→ entries in ascending order.
  (i)    [initialize]
     Start ← 0, end ← N
  (ii)   Perform search
     Repeat through step (iv) while  low ≤ high
  (iii)  [obtain index of mid point of interval]
     Middle ← [ (start + end)/2]
  (iv)   Compare
         If X<K (middle)
             Then end ← middle -1
         Else
             If X>K(middle);
```

```
            Then start ← middle +1
            Else
                  Write (“Successful search”)
            Return (middle)
   (v)    [unsuccessful search]
      Return (0)
```

Question:- Tress a Binary search algorithm I/P data:-

75, 151, 203, 275, 318, 489, 524, 591, 647, 727

Search Pop x = 275, 727, 725.

Sol^n:-

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
75	151	203	275	318	489	524	591	647	727

Iteration:1

Start = 0 end = 9 middle = $\frac{0+9}{2} = 4$

Since middle element [4] = 318

Since, the middle element is greater than search element, so we assign,

Iteration:-2

Start = 0 end = 3 middle = $\frac{0+3}{2} = 1$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
75	151	203	275	318	489	524	591	647	727

151 < 275

Now,

Start = middle + 1 = 2 end = 3

Iteration:- 3

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
75	151	203	275	318	489	524	591	647	725

203 < 275 end = 3

Iteration :- 4

Middle = $\frac{3+3}{2}=3$

275 = 275

Middle [3] = 275

For 725

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
75	151	203	275	318	469	524	591	647	727

Iteration:-1

Start = 0 end = 9 middle = $\frac{0+9}{2}=4$

725> 318

Iteration:-2

Start = 5 end = 9 middle = $\frac{5+9}{2}=7$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
75	151	203	275	318	489	524	591	647	727

725 < 591

Iteration:-3

Start = 8 end = 9 middle = $\frac{8+9}{2}=8$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
75	151	203	275	318	489	524	591	647	727

725 < 647

Iteration:- 4

Start = 9 end = 9 middle = $\frac{9+9}{2}=9$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
75	151	203	275	318	489	524	591	647	727

725 < 727

Binary search tree:- (BST)

Algorithm:-

```
[initialize]
    Read (no)
    Root node content = 0
    Right subtree = NULL
    Left sub tree = NULL
while there is data
    Do
            Begin
                    Read(no)
compare no. with the content of root.
Repeat
    If match then declare duplicates
    Else
            If no < root, then
            Root = left sub tree root
       Else
            Root = right sub tree root.
    Until
            Duplicate found or (root = = NULL)
    If(root =  = NULL) then place  it as root.
```

Application of BST:-

sorting a list

→ construct a BST &

→ Traverse in in order

For conversion prefix, infix & posfix expression.

→ construct a BST &

→ traverse in preorder, post order, inorder

Algorithm to build a binary search tree (BST) from post fix expression

→ Read symbol at a time.

→ If symbol is a operand.

* create one node tree push pointer to stack.

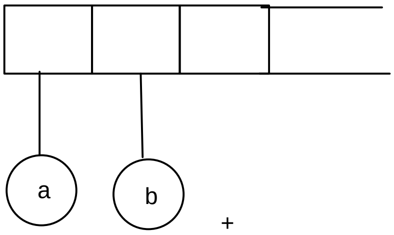
→ if symbol is operator

pop pointer to two trees T! and T2 & form a new tree whose root is children point to T1 & T2 respectively.

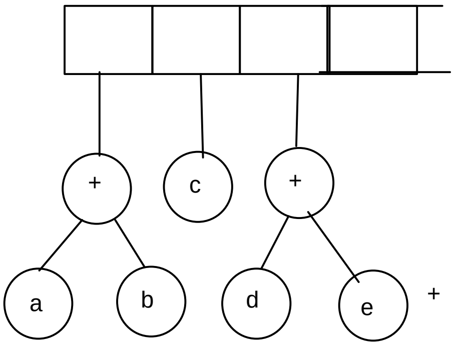
Pointer to this new tree is then pushed to stack.

Ab+ cde + **

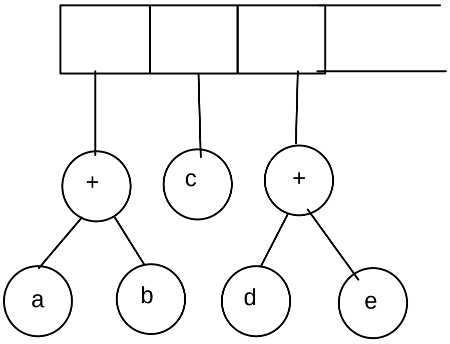
1.



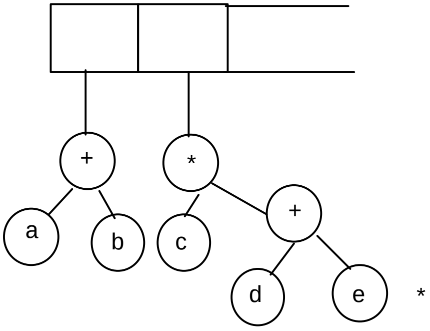
2.



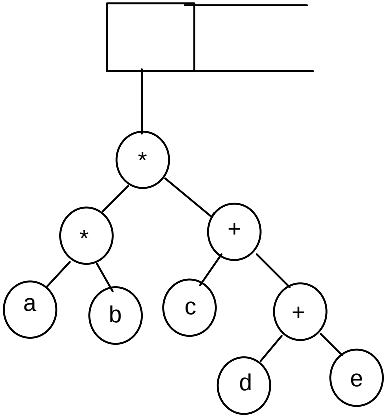
3.



4.



5.



Inorder:-

Preorder:-

Post order:-

“Hashing”

Sequential search, binary search and all the search trees are totally dependent on no. of element and may key comparisons are involved. Now, our need is to search the element in constant time and loss key comparisons should be involved.

Suppose all the elements are in array of size ‘N’. Let us take all the keys are unique and in the rage ‘o’ to N-1. Now we are sorting the record in array based on key, where array index and keys are same. Then we can access the record in constant time and no key comparisons are involved.

Consider 5 records where keys are :-

9, 4, 6, 7, 2

The keys can be stored in array up

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

Here, we can see the record which has key value can be directly accessed through array index.

In hashing key is converted into array index and records are kept in array. In the same way for searching the record, key are converted into array index and get the records from array.

For storing records:-

Key
↓
Generate array index
Key
↓
Stored the record on that array index.

For accessing record:-

Key
↓
Generate array index
Key
↓
Get the records from the array index

The array which supports hashing for storing records or searching records is called hash table. Each key is mapped on a particular array index through hash function. If each key is mapped on a unique hash table then this situation is ideal but there may be possibility that hash function generating some hash table address for different keys & this situation is called collision. For missing collision we have to perform:

Choose a good hash table which perform minimum collision.
Resolving the collision.

Technique for choosing hash function.

Trunction method:-

In this method a part of hey is considered as address, it can be some rightmost digit or leftmost digit.

Q. apply the truncation method to get the hash index of table size of 100 for following keys.

82394561, 87139465, 83567271, 85943228
Sol^n:
Table size = 100
Then,

Take 2 rightmost digits for getting the hash table address.

Key value	address
82394561	61
87139465	65
83567271	71
85943228	28

Example. h(key) = a
h(823994561) = 61

Mid square method:-

In this method we square the key , after getting number we take some middle of that number as an address, suppose keys are 4 digits and maximum address = 100.

Key value	**square**	**address**	
1456	02119936	19	h(1456) = 19
1892	03579664	79	h(1892) = 79

3. folding method:-

In this method we break the key in to pieces & add them and get the address. Suppose 10 digit key 1234567890 may be reduced to key 50, 000 storage location by carrying out the steps.

1 2 3 4 5 6 7 8 9 0

Steps:-

1 2 3 4 5 6 7 8 9 0
3 4 5 6 7
+ 1 2 8 9 0
4 7 4 5 7

3. number > 50, 000?
4. if no; number = address
5. if yes, subtract 50,000 with the no until no<50,000.

H(1234567890) = 47457

Modulus method:-

Modular method is best way for getting address from key take the key, do they modulus operation & get the remainder as address for hash table in order to minimize the collision table size should a prime number consider keys.

82394561, 87139465, 83567271, 89943228
& Table size = 97
Then, address are
82394561 % 97 = 45
87139465 % 97= 0
83567271% 97=25
89943228% 97= 64

82394561 = 82 = 3945 + 61= 4088
87139465 = 87 + 1394 + 65=
83567271= 83 + 5672 +71=
89943228 = 89 + 9432 + 28=

Applying modulus method we get,
4088 % 97 = 14.

Hash function for floating point numbers:-

The operations can be performed as

i. Check the fractional part of key.
ii. Multiply the fractional part with the size of the hash value.
iii. Take the integer part of the multiplication result as a hash address of key.

e.g. Question:- Consider the key as floating point number as
123.4321, 19.469, 2.0289, 8.9956
Hash size = 97

Sol^{n}:-
0.4321 x 97 = 41. 9139

0.463 x 97 = 44.911
0.0289 x 97 = 2.8906
0.9956 x 97 = 96.5732

Here, hash address will be integer part of these numbers

i.e. H(123.4321) = 41
H(19.463) = 4
H(2.0289) = 2
H(8.9956) = 96

Hash function for string:-

iv. In this case we add the ASCII value of each character and then apply modulus operation on this value.

e.g consider the table size = 97

PUSET

PUSET = P + U + S + E + T
= 112+ 117 + 115 + 101 + 116
= 561

Now,

Applying modulus operation, we get

H(PUSET) = 561% 97
= 76

Hence, puset lies on the 76th address of hash table.

v. In second method ASCILL value of each character is multiplied by 127 i.e. maximum value of ASCII character and then perform modulus operation.

e.g.

table size = 997

PUSET =

PUSET = P + U + S + E + T
= (112 + 117+ 115 + 101 + 116) x 127
= 71247

Now, applying modulus operation, we get

H(PUSET) = 71247 % 997
= 460

Q. **1.** Apply the folding and modulus method to get the hash index of table size 11 for following keys.

8925, 23197,37565, 48693, 46453

Q. 2. :- Get the hash index in the table size 29 for the following floating point number.

56.9281, 145.0092, 28.45, 28.45 , 89.3967, 2.877

Q 3. :- Get the hash index in the table size 19 for the following keys.

PUSET, eastern, khoppa, acme, novel

Q 4. :- apply the truncation method to get the hash index of the table size 997 for following keys.

699934, 674352, 632433, 678433, 678668, 629871,

653420

Q 5:- apply folding method to get the hash index of table size 79 for the following keys.

56497,79256,27143,49239,18942,77722

Q 6:- apply the mid- square method to get the index of table size 97 for the following keys.

1123, 1234 1012, 1034, 1103, 1005

# collision Resolution technique:-

open chaining : separate chaining

closed chaining.

i. Linear probing

ii. Quadratic probing

iii. Double hashing

iv. Rehashing.

Open chaining (separate chaining):-

This method maintains the chain of elements which have same hash address. We can take hash table as an array of pointers. In this method each pointer will point to one linked list and the elements which have same hash address will be maintained in the linked list basically it involves two operations.

(i) creation of god hash function for getting hash key value in the hash table.

(ii) Maintain the elements in the linked list which is pointed by the pointer available in hash table.

e.g.

I/P keys:- 13, 15, 6, 24, 23 & 20

Table size :- 7

Hash function h(x) = x %7

h(13) = 13%7 = 6

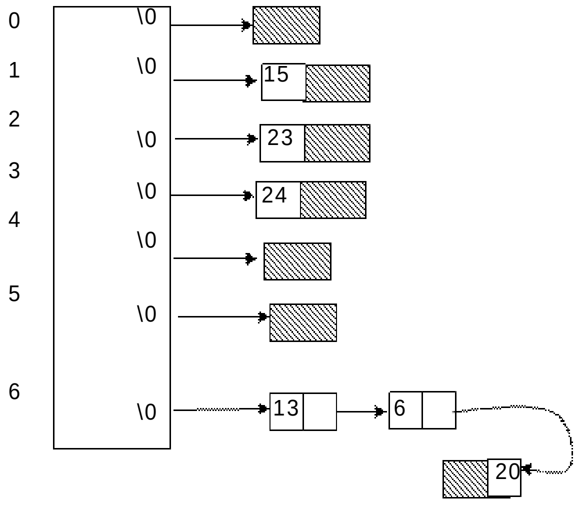
h(15) = 15%7 = 1

h(6) = 6%7 = 6

h(24) = 24%7 = 3

h(23) = 23 % 7 = 2

h(20) = 20%7= 6



Closed chaining (open addressing)

(iii) Linear probing:-

This hashing technique finds the hash key value through hash function and maps the key on the particular position in hash in hash, table. In case if key has same hash address then it will find the next empty position in the hash table we take the hash table as circular array. If table size in n then after n-1 position it will search form zeroth position in the array.

e.g. consider table size 11 & elements are

29, 18, 23, 10, 36, 26, 46, 43

Sol^{n}:-

H(29) = 29%11 = 7

H(18) = 18 %11= 7

H(23) = 23%11 = 1

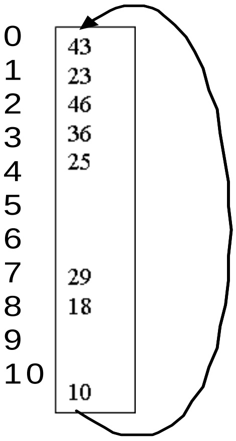
H(10) = 10%11= 10

H(36) = 36%11= 3

H(25) = 25%11= 3

H(46) = 46%11= 2

H(43) = 43%11= 10



0	43
1	23
2	46
3	36
4	25
5	
6	
7	29
8	18
9	
10	10

Disadvantage of linear probing:-

→ dustering problem.

→ searching is slow.

Quadratic probing:-

In case of collision

Rehash functions : (Hash value + 1^2) % size

(Hash value + 2^2) % size & so on.

This technique decrease the problem of dustering but can't search all the locations, if hash table is prime then it will search at least half of the locations of hash table.

e. g.

table size = 11

given elements = 29, 18, 43, 10, 46, 54

H(29) = 29% 11= 7

H(18) = 18%11 = 7

(hash value + 1^2) %11

(7 + 1) % 11 = 8

H(43) = 43%11= 10

H(10) = 10 % 11 = 10

(Hash value + 1^2) %11

H(46) = 46%11 = 2

H(54) = 54%11 = 10

(10 + 1^2) %11 = 0

(10 + 2^2) % 11= 3

0	10
1	
2	46
3	54
4	
5	
6	
7	29
8	18
9	
10	43

Double hashing :-

This technique requires hashing second time in case of collision. Suppose h is a hash key then in case of collision we will again do the hashing of this hash key.

i.e. Hash(h) = h’

now,

we will search the hash key location as h, h+h’, h+2h’ & h+3h’ & so on.

Consider, the table size = 13 then two hash functions are

H = key % 13

& h’ = 11(key % 11)

At the time of collision, hash address for next probability is (h+h’) % 13

((key % 13) + (11 – (key %11)))) % 13

Or, h+ 2h’

Or, (key % 13) + 2(1) – key % 11))

Question:- consider elements 8, 55, 48, 68 table size = 13

h = key % 13

h’ = 11 – (key % 11)

now, applying modulus operation,

h(8) = 8%13= 8

h(55) = 55%13= 3

h(48) = 48%13 = 9

h(68) = 68 % 13= 3

hence, collision occurs at table location 3.

so, applying double hashing, we get

(key % 13) + (11- key %11)) % 13

(3 + 11 – 2) % 13 = 12 % 13 = 12

0	
1	
2	
3	55
4	
5	
6	
7	
8	8
9	48
10	
11	
12	68

Advantages:-

→ we do operation faster.

Draw backs:-

Requires two times calculation of hash function, which creates if complex & searching will be slower than linear & quadratic probing.

Rehashing:-

There are chances of insertion failure when hash table is full, so the solution for this particular case is to create a new hash table with the double size of previous hash table. Here, we will use new hash function and we will insert all the elements of the pervious hash table. So, we will scan the elements of previous hash table one by open & calculate the hash key with new hash function & insert them into new hash table.

Consider the table size 11 & elements are

7, 18, 43, 10, 36, 25………..

0	10
1	
2	
3	36
4	25
5	
6	
7	7
8	18
9	
10	43

Sol^{n}:- Size = 11

Applying linear probing we get

h(7) = 7%11= 7
h(18) = 18%11= 7
h(43) = 43%11= 10
h(10) = 10%11= 10
h(36) = 36%11= 3
h(25) = 25%11= 3

now, if we want to insert six more element then size will not be sufficient . In order to fit all the elements or key with in a table we take new table of size more than double with prime number. Thus total size is 23.

Applying linear probing, we get,

h(7) = 7%23= 7
h(18) = 18%23 = 18
h(43) = 43%23 = 20
h(10) = 10%23 = 10
h(36) = 36%23= 13
h(25) = 25%23 = 2
.. ….
.. ..
.. ..

0	
1	
2	25
3	
4	
5	
6	
7	
8	
9	
10	10
11	
12	
13	36
14	
15	
16	
17	
18	18
19	
20	43
21	
22	

Disadvantage:-

This method is more expression.

CHAPTER :- 10

Graph:-

A graph B is a collection of two sets V & E where V is the collection of vertices . $V_0, V_1 \ldots\ldots\ldots V_{n-1}$ also called nodes & E is the collection of edges $e_1, e_2, \ldots\ldots\ldots e_n$ .where an edges is an arc which connects two nodes. This can be represented as

$G = (V,E)$

$V(G) = (V_0, V_1 \ldots\ldots\ldots V_{n-1})$ or set of vertices.

$E(G) = (e_1, e_2, \ldots\ldots\ldots e_n)$ or set of edges.

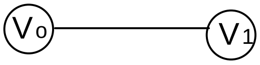
Types:-

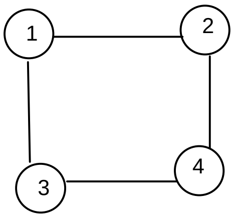
undirected

directed.

Undirected:-

A graph which has unordered pair of vertices is called undirected graph. Suppose there is an edge between V_0 & V_1 then it can be represented as (V_0, V_1) or (V_1, V_0).



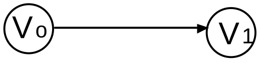


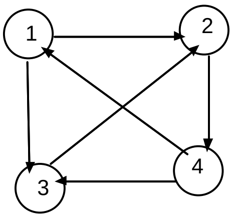
$V(G) = \{ 1, 2, 3, 4\}$

$E(G) = \{(1, 2), (1, 4), (1, 3), (2, 3), (2, 4), (3, 4)\}$

Directed:-

A directed graph or digraph is a graph which has ordered pair of vertices (V_1, V_2) where V_1 is the tail & V_2 is the head of the edge.





(V_1, V_2)

$V(G) = \{ 1, 2, 3, 4\}$

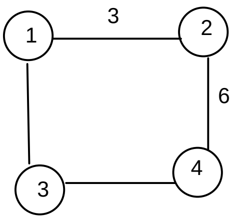
$E(G) = \{(1, 2), (1, 4), (1, 3), (2, 3), (2, 4), (3, 4)\}$

Weighted graph:-

A graph is said to be weighted if it's edges have been assigned with some non negative as weight.

Adjacent nodes:-

A node u is adjacent to another node or is a neighbor of another node V if there is an edge from u to node V. In undirected graph if(V_o, V_1) is an edge then V_o is adjacent to V_1 & V_1 is adjacent to V_o. In a digraph if (V_o, V_1) is an edge then V_o is adjacent to V_1 & V_1 is adjacent from V_o.



Path:-

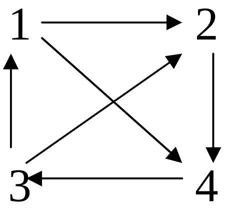
A path from node u_0 to node u_n is a sequence of nodes $u_{0,}$ u1 , u2......... u_{n-1} , u_n such that u_0 is adjacent to $u_{1,}$ u_1 is adjacent to u_2 …………….. u_{n-1}. is adjacent of u_n

Length of Path:-

Length of a path is the total number of edges included in the path.

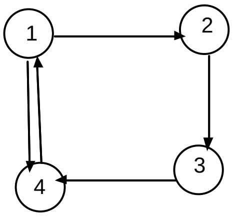
Closed path:-

A path is said to be closed if first & last node of the path are same.



Simple Path:-

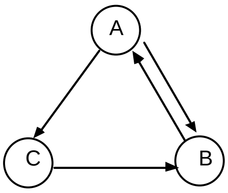
It is a path in which all the nodes are distinct with an exception that the first & the last nodes of the path can be same.



Cycle:-

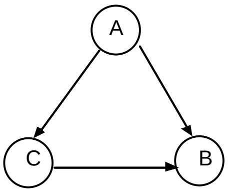
Cycle is a simple path in which first & last nodes are the same.

A



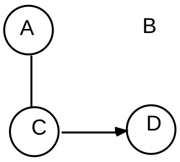
Cyclic graph:-

A graph that has no cycles is called Acyclic graph.



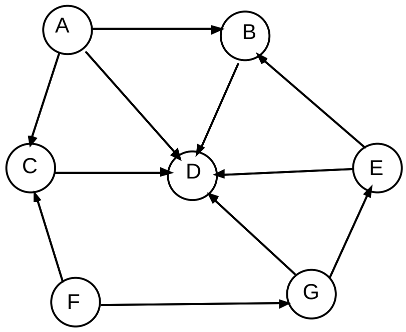
Degree:-

IN an undirected graph the number of edges connected to a node is called the degree of that node or degree of a node is the no. of edges incident on it.



Indegree:-

The indegree of a node is the number of edges coming to that node.



Here, in degree of nodes:-

A = 0
B = 2
C = 2
D = 6
E = 1
F = 0
G = 1

Outdegree:-

The out degree of a node is the number of edges going outside form the node.

Out degree of A = 3
B = 1
C = 1
D = 0
E = 2
F = 3
G = 2

Isolated node:-

If a node as no edges connected with any other node then it's degree will be zero & is called as isolated node

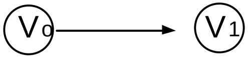
A C

B D

C is isolated node.

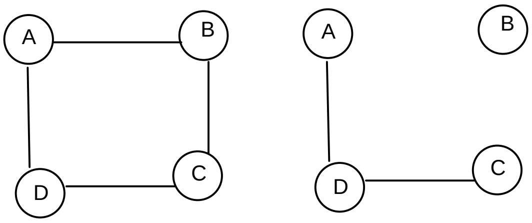
Successor or predessor:-

In digraph if a node V_o is adjacent to node V_1 then V_o is the predessor of V_1 & V_1 is the successor of $V_{o,}$



Connected graph:-

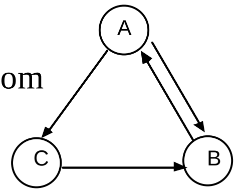
An undirected graph is connected if there is a path from any node of a graph to any other node.



Here, G_1 is connected Graph while G_2 is unconnected graph.

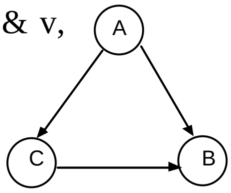
Strongly Connected:-

A digraph is strongly connected if there is a directed path from any node of graph to any other node.



Weakly connected:-

A digraph is called weakly connected for any pair of nodes u & v, there is a path form u to v r a path from v to u but not both.



Maximum, edges in graph:-

In undirected graph maximum edge = $\frac{n(n-1)}{2}$ & in digraph.

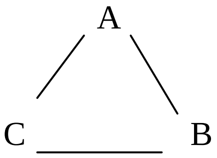
Maximum edges = n(n -1)

Where, n is the total no. of nodes in the graph.

n(n-1)/2

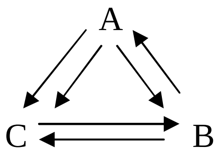
= 3(8 -1)/2

= 3



n(n -1)

3(3 -1) = 6



Representation of Graph

Two ways:-

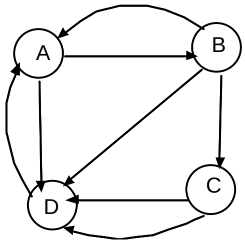
→ Sequential representation (adjacency matrix)

→ linked list representation (adjacency list)

Adjancy matrix:-

Adjancy matrix is the matrix which keeps information of adjacent nodes i.e. keeps the information weather node is adjacent to any other node or not. Suppose there are four nodes in a graph then row one reprew3ents the node 1, row 2 represents the node 2 & so on . similarly, column 1 represents node 1 & column 2 represents node 2 & so on. The entry of the matrix will be.

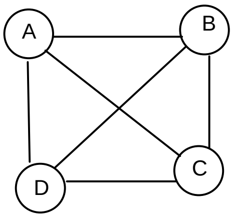
Arr[i][j] = 1   if there is an edge from node I to node j
= 0 if there is no edge from node I to node j.



$$\begin{array}{c|cccc|c} & A & B & C & D & \text{outdegree} \\ \hline A & 0 & 1 & 0 & 1 & 2 \\ B & 1 & 0 & 1 & 1 & 3 \\ C & 0 & 0 & 0 & 1 & 1 \\ D & 1 & 0 & 1 & 0 & 2 \\ \text{indegree} & 2 & 1 & 2 & 3 & \end{array}$$

again,

For undirected graph:-



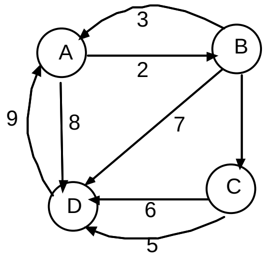
$$\begin{array}{c|cccc|} & A & B & C & D \\ \hline A & 0 & 1 & 1 & 1 \\ B & 1 & 0 & 1 & 1 \\ C & 1 & 1 & 0 & 1 \\ D & 1 & 1 & 1 & 0 \end{array}$$

It has no in degree & out degree because it has no direction of node.

Representation of weighted graph in matrix form:-

If graph has some weight on it's edge then,

Arr[i][j] = weight on edge (if there is an edge from I to node j
= 0   (otherwise)



Indegree =

Weighted adjacency matrix

$$\begin{array}{c|cccc|c} & A & B & C & D` & \underline{\text{out degree}} \\ \hline A & 0 & 2 & 0 & 8 & 10 \\ B & 3 & 0 & 4 & 7 & \\ C & 0 & 0 & 0 & 5 & \\ D & 9 & 0 & 6 & 0 & \end{array}$$

Warshall Algorithm:-

Used for finding path matrix of a graph

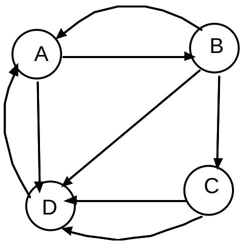
Algorithm:-

```
Initialize
    P ← A
```

```
 [perform a pass]
     Repeat through step 4 fro K = 1, 2, ……….n
process row]
     Repeat step 4 for I = 1, 2, ………..n
process columns]
     Repeat for j= 1, 2, ……….n
     $P_{ij}$ U ($P_{ik} \cap P_{kj}$)
[finish]
     Return
```

Q:- From the given graph find out the path matrix by warshal algorithm.

$$P_0 = \begin{array}{c|cccc} & A & B & C & D` \\ \hline A & 0 & 1 & 0 & 1 \\ B & 1 & 0 & 1 & 1 \\ C & 1 & 1 & 0 & 1 \\ D & 1 & 1 & 1 & 0 \end{array}$$



$P_{ij} \cup (P_{ik} \cap P_{kj})$

For k = 1

$P_{ij} \cup (P_{i1} \cap P1j)$

$$P_1 = \begin{array}{c|cccc} & A & B & C & D \\ \hline A & 1 & 0 & 1 & 0 \\ B & 1 & 1 & 1 & 1 \\ C & 0 & 0 & 0 & 1 \\ D & 1 & 1 & 1 & 1 \end{array}$$

Similarly taking k = 2 $\quad P_{ij} = P_{ij} \cup (P_{ik} \cap P_{kj})$

$P_{ij} = P_{ij} \cup (P_{i2} \cap P_{2j})$

$$P_2 = \begin{array}{c|cccc} & A & B & C & D` \\ \hline A & 1 & 1 & 1 & 1 \\ B & 1 & 1 & 1 & 1 \\ C & 0 & 0 & 0 & 1 \\ D & 1 & 1 & 1 & 1 \end{array}$$

Now taking k = 3,

$$P_3 = \begin{array}{c|cccc} & A & B & C & D` \\ \hline A & 1 & 1 & 1 & 1 \\ B & 1 & 1 & 1 & 1 \\ C & 0 & 0 & 0 & 1 \\ D & 1 & 1 & 1 & 1 \end{array}$$

Again taking k = 4

$$P_4 = \begin{array}{c|cccc} & A & B & C & D` \\ A & 1 & 1 & 1 & 1 \\ B & 1 & 1 & 1 & 1 \\ C & 1 & 1 & 1 & 1 \\ D & 1 & 1 & 1 & 1 \end{array}$$

Here, P_0 is the adjency matrix & P_u is the path matrix of the graph.

Q:- Modified warshal's algorithm:-

Warshall's algoritham give the path matrix of graph. By modifying this algorithm, we will find out the shortest path matrix Q. Q_{ij} represent the length of shortest path from V_i to V_j. Here, we consider the matrices q_0, q_1, q_2, ………. q_n.

Thus, length of shortest path from V_i to V_j using nodes $V_{j,}V_{2,}$ ….V_n

$Q_{k[i][j]}$ =

∞ [i] there is no path from V_j to V_j using nodes $V_{1,}$ V_2................ V_n

Procedures:-

→ In this algorithm length of 1st path will be Q_{k-1} [i][j]

→ length of 2nd path will be Q_{k-1} [i][j] + Q_{k-1} [i][j].

Now, select the smaller one from these two path length so value of

Q_k [i][j] = Minimum [Q_{k-1} [i][j], Q_{k-1} [i][j] + Q_{k-1} [i][j]]

Algorithm:-

```
→   Q ← A
      - adjacency matrix with 0 replaced by ∞
→   [Perform a pass]
      - Repeat through step 4 for k = 1,2, ………..0.
→   [process rows]
      - Repeat step 4 fro j = 1,2, …………n
→   [process column]
      - Repeat for j = 1,2, ……….n
  $Q_k$ [i][j] ← Min [$Q_{k-1}$ [i][j], $Q_{k-1}$ [i][j] + $Q_{k-1}$ [i][j] ]
→   [finish]
      - Return
```

Case :-1

Q_k [i][j] = ∞ & Q_{k-1} [i][j] + Q_{k-1} [i][j] = ∞

Then, Q_k [i][j] =min (∞,∞) = ∞

Case :-2

Q_k [i][j] = ∞ & Q_{k-1} [i][j] + Q_{k-1} [i][j] = b

Then, Q_k [i][j] =min (∞,b) = b

Case :-3

Q_k [i][j] = a & Q_{k-1} [i][j] + Q_{k-1} [i][j] = ∞

Then, Q_k [i][j] =min (a,∞) = a

Case :-4

Q_k [i][j] = a & Q_{k-1} [i][j] + Q_{k-1} [i][j] = b

Then, Q_k [i][j] =min (a,b)

Traversal in graph:-

There are two efficient techniques for traversing the graph.

→ 1) depth first search (DFS)

→ 2) Breadth first search (BFS)

Difference between traversal in graph & traversal in tree or us

There is no 1st node or root node in graph . Hence the traversal can start from any node

In tree or list when we start traversing from the 1st node, all the nodes are traversed which are reachable from the starting node. If we want to traverse al the reachable nodes we again have to select another starting node for traversing the remaining nodes.

In tree or list while traversing we never encounter a node more then once but while traversing graph,. There may be a possibility that we reach a node more than once.

In tree traversal, there is only one sequence in which nodes are visited but in graph for the same technique of traversal there can be different sequences in which node can be visited.

Breadth first search:-

This technique uses queue for traversing all the nodes of the graph. In this we take any node as a starting node than we take all the nodes adjacent to that starting node. Similar approach we take for al other adjacent nodes which are adjacent to the starting node & so on. We maintain the start up of the visited node in one array so that no node can be traversed again.

Algorithm:-

```
[initialize]
    Mark all vertex unvisited
begin with any node.
    Insert it into queue (initially queue empty)
Remove node from queue.
    append it to traversal list.
    2   Mark it visited.
```

4. insert al the unvisited or node snot an queue in to the queue.
2. Repeat step 3 to 5 until queue is empty.
3. [Finish]
   Return.

1						
	2	8	3	5		
		8	3	5	6	7
			3	5	6	7
				5	6	7
					6	7
						7

Thus the traversal list is

1→ 2 → 8 → 3 → 4 → 5 → 6 → 7

Depth first Search:-

This technique uses stack for traversing all the nodes of the graph in this we take one as starting node then go to the path which is from starting node & visit all the nodes which are in that path. When we reach at the last node then we traverse another path starting from that node. If there is no path in the graph from the last node then it returns to the previous node in the path & traverse another & so on.

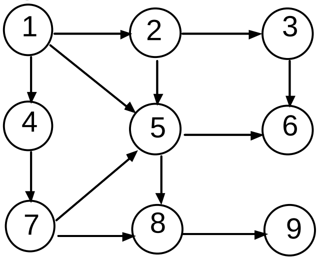
Algorithm:-

$\overset{*}{1}$	$\overset{*}{2}$	8	3		
2	$\overset{*}{1}$	4	5		
3	1	$\overset{*}{6}$	$\overset{*}{7}$		
4	$\overset{*}{2}$	$\overset{*}{8}$			
5	2	8			
6	$\overset{*}{3}$	8			
7	$\overset{*}{3}$	$\overset{*}{8}$			
8	$\overset{*}{4}$	$\overset{*}{5}$	$\overset{*}{1}$	$\overset{*}{6}$	7

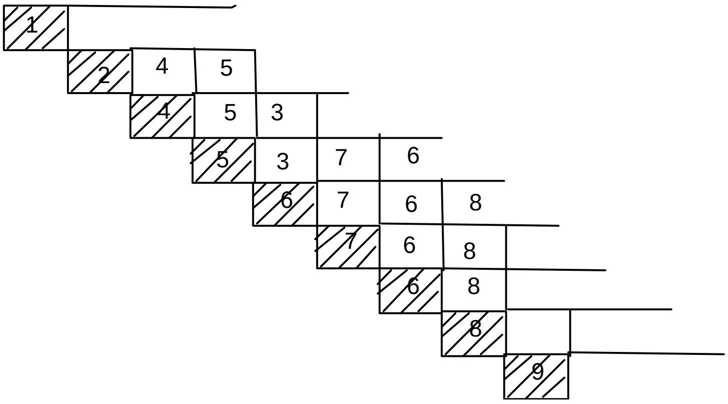
Traversal:-

1→ 2 → 8 → 3 → 4 → 5 → 6 → 7

Question :-



Breadth first search:-



Thus traversal list is

1→ 2 → 4 → 5 → 3 → 7 → 6 → 8 → 9

Depth first search:-

→1 $\overset{*}{4}$ 5 2

2 $\overset{*}{5}$ 3

3 1 6

→4 $\overset{*}{7}$

5 $\overset{*}{8}$ $\overset{*}{6}$

6

7 $\overset{*}{5}$ 8

8 $\overset{*}{9}$

$\overset{*}{9}$

Traversal is

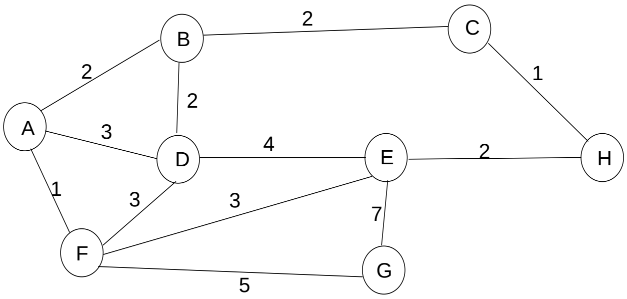
1→ 4 →7 → 5 → 8 → 9 → 6 → 2 → 3

Shortest path algorithm:-

→ used to find the shortest path form one node to another node.

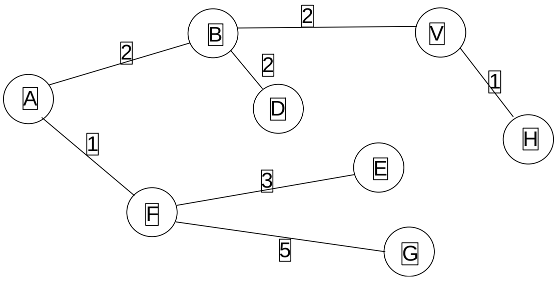
→ A single source vertex & seek shortest path to all other vertices.

→ shortest path is that path in which the sum of weight of included edges is minimum.



From A

Shortest path to B = 2
C = 2 + 2 = 4
D = (2 + 2) or (1 + 3) = 4
E = 4
F = 1
G = 6
H = 5



Dis Kstra algorithm:-

In this technique each node is labeled with distance (dist) predecessor & status, distance of node represents, the shortest distance of that node from the source node & predecessor of a node represents the node which precedes the given node in shortest path form source. Status of a node can be permanent or temporary.

→ shaded circle represent permanent nodes which indicates that it has been included in the shortest path.

→ Temporary nodes can be relabeled if required but once node is made permanent, it can't be rebelled.

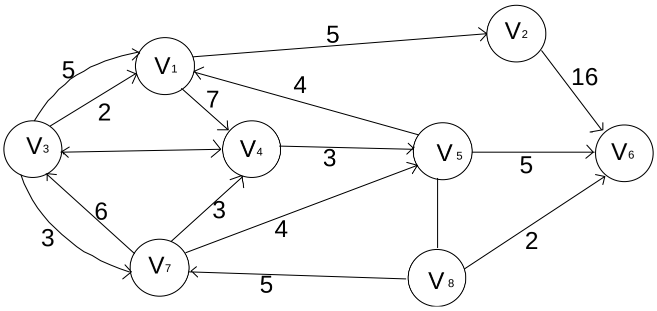
Procedure:-

→ Initially make source node permanent and make it the current working node . al other nodes are node temporary.

→ Examine all the temporary neighbors of the current working nodes & after checking the condition for minimum weight reliable the require node.

→ From all the temporary nodes find out the node which ahs minimum value of distance, make the node permanent & now this is the current working node.

→ Repeat step 2 & 3 until destination node is made permanent.



Let , V_1 = source node

Node	dist	pred	status
V_1	0	0	permanent
V_2	∞	0	temp
V_3	∞	0	temp
V_4	∞	0	temp
V_5	∞	0	temp
V_6	∞	0	temp
V_7	∞	0	temp
V_8	∞	0	temp

Check adjacent node of V_3

V_4 Dis > V_3 dis + distance (V_3,V_4) 7<2+ 4 relable.

V_7 Dis > V_3sis + sistance (V_3, V_7) ∞> 2 + 3 relable

Node	dist	pred	status
V_1	0	0	permanent
V_2	∞	0	temp
V_3	∞	0	temp
V_4	∞	0	temp
V_5	∞	0	temp
$V_{6]}$	∞	0	temp
V_7	∞	0	temp
V_8	∞	0	temp

Check adjacent node of V_7

V_4 Dis > V_{37} dis + distance (V_7,V_4) 7<2+ 4 relable.

V_5 Dis > V_7 sis + sistance (V_7, V_5) ∞> 2 + 3 relable V_5

Node	dist	pred	status
V_1	0	0	permanent
V_2	8	V_1	temp
V_3	2	V_1	permanent
V_4	6	V_3	permanent
V_5	9	V_7	temp
$V_{6]}$	∞	0	temp
V_7	5	V_3	permanent
V_8	∞	0	temp

Check adjacent node of V_4

V_5 Dis > V_4 dis + distance (V_4,V_5) 9<6+9 leave.

Node	dist	pred	status
V_1	0	0	permanent
V_2	0	V_1	permanent
V_3	2	V_1	permanent
V_4	6	V_3	permanent
V_5	9	V_7	temp
$V_{6]}$	∞	0	temp
V_7	5	V_3	permanent
V_8	∞	0	temp

Check adjacent node of V_2

V_6 Dis > V_2 dis + distance (V_2,V_6) ∞>8 + 16 relable.

Node	dist	pred	status
V_1	0	0	permanent
V_2	8	V_1	permanent
V_3	2	V_1	permanent
V_4	6	V_3	permanent
V_5	9	V_7	permanent
$V_{6]}$	24	0	temp
V_7	55	V_3	permanent
V_8	∞	0	temp

Check adjacent node of V_5

V_6 Dis > V_5 dis + distance (V_5,V_8) $\infty > 9 + 8$

V_6 Dis > V_5 dis + distance (V_6, V_5) $24 > 9+5$

Node	dist	pred	status
V_1	0	0	permanent
V_2	8	V_1	permanent
V_3	2	V_1	permanent
V_4	6	V_3	permanent
V_5	9	V_7	permanent
V_6	14	V_5	permanent
V_7	5	V_3	permanent
V_8	17	0	temp

Here, V_6 is smallest & make it permanent since. V_6 is the destination node make it permanent & stop.

Now, start from destination node V_6 & keep on seeing it's perdecessors until we get source node as predecessor.

Predecessor of V_6 is V_5

" : " V_5 is V_7

" : " V_7 is V_3

" : " V_3 is V_1

$V_1 \rightarrow V_3 \rightarrow V_7 \rightarrow V_5 \rightarrow V_6$

Spanning tree:-

A spanning tree of a connected graph G contains all the nodes and has the edges which connects all the nodes so that number of edges will be one less than the no. of nodes.

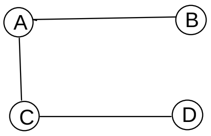


Fig a

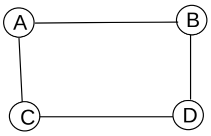


Fig a

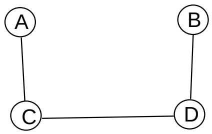


Fig b

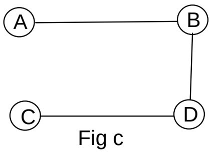


Fig c

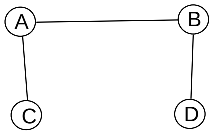


Fig d

Here, A B C D is a spanning tree & D is the minimum spacing tree.

Node:-	1	2	3	4	5	6	7	8	9
Father:-	0	0	0	0	1	0	0	0	0

Step :2 = selected is 4 -5 wet = 3.
n_1 = 4 root n_1 = 4 n_2 = 5   root n_2 = 1
Roots are different, so edge is inserted.
Father [1] = 4

Node:-	1	2	3	4	5	6	7	8	9
Father:-	4	0	0	0	1	0	0	0	0

Step:- 4 edge selected is 3 – 6   wt = 5
n_1 = 3   root – n_1 = 3 n_2= 6   root – n_2= 6
father [6] = 3
roots are different so edge is inserted.

Node:-	1	2	3	4	5	6	7	8	9
Father:-	4	0	0	0	1	3	0	0	0

Step: 5   edge selected is 5-6 wt = 6
n_1 = 5   root – n_1 = 4 n_2= 6   root – n_2= 3
Roots are different so edge is inserted in spanning tree. Father of [3] = 4

Node:-	1	2	3	4	5	6	7	8	9
Father:-	4	0	0	0	1	3	0	0	0

Step: 6   edge selected is 3 -5 wt = 7
n_1 = 3   root – n_1 = 4 n_2= 5   root – n_2= 4
Roots are same so edge is not inserted i

Node:-	1	2	3	4	5	6	7	8	9
Father:-	4	0	4	0	1	3	0	0	0

Step: 7   edge selected is 2-5 wt = 8
n_1 = 2   root – n_1 = 2 n_2= 5   root – n_2= 4
Roots are different so edge is inserted Father of [4] = 2

Node:-	1	2	3	4	5	6	7	8	9
Father:-	4	0	4	2	1	3	0	0	0

Step: 8   edge selected is 1-2 wt = 9
n_1 = 1   root – n_1 = 2 n_2= 2   root – n_2= 32
Roots are same so edge is not inserted

Node:-	1	2	3	4	5	6	7	8	9
Father:-	4	0	4	2	1	3	0	0	0

Step: 9 edge selected is 2-3 wt = 10

n_1 = 2 root – n_1 = 2 n_2= 3 root – n_2= 2

Roots are same so edge is not inserted

Node:-	1	2	3	4	5	6	7	8	9
Father:-	4	0	4	2	1	3	0	0	0

Step: 10 edge selected is 5-7 wt = 11

n_1 = 5 root – n_1 = 2 n_2= 7 root – n_2= 7

Roots are different so edge is inserted in spanning tree. Father of [7] = 2

Node:-	1	2	3	4	5	6	7	8	9
Father:-	4	0	4	2	1	3	2	0	0

Step: 11 edge selected is 5-8 wt = 12

n_1 = 5 root – n_1 = 2 n_2= 8 root – n_2= 8

Roots are different so edge is inserted in spanning tree. Father of [8] = 2

Node:-	1	2	3	4	5	6	7	8	9
Father:-	4	0	4	2	1	3	2	2	0

Step: 12 edge selected is 7-8 wt = 14

n_1 = 7 root – n_1 = 2 n_2= 8 root – n_2= 2

Roots are same so edge is not inserted

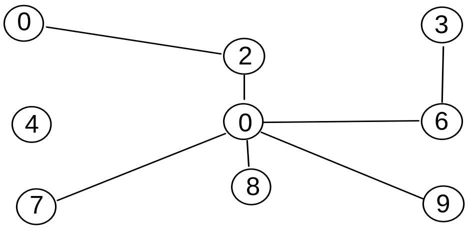
Node:-	1	2	3	4	5	6	7	8	9
Father:-	4	0	4	2	1	3	2	2	0

Step: 13 edge selected is 5- 9 wt = 15

n_1 = 5 root – n_1 = 2 n_2= 9 root – n_2= 9

Roots are different so edge is inserted in spanning tree. Father of [8] = 2

Node:-	1	2	3	4	5	6	7	8	9
Father:-	4	0	4	2	1	3	2	2	2



Since, minimum spacing tree should contain n-1 edges where n is the no. of nodes in the graph. This graph contains nine nodes so after inserting 8 edges in the spanning tree we will not examine other edges & stop the process.

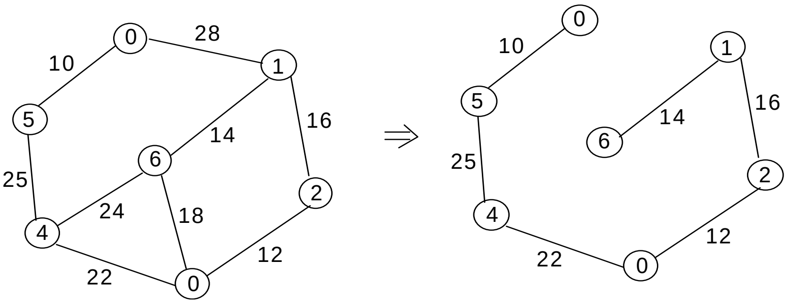
Here, edge included in the spacing tree are (1, 5) (4, 5) (3, 6), (5, 6), (2, 5) ,(5, 7), (5, 8) & (5, 9) & weight of spanning tree = 2 + 3+ 5+ *+ 11+12 + 15

` = 62

Round- Robin algorithm:-

Initialize the spanning forest to contain the node but not edges.

1. the node but not edges.
   a. Each vertex is in It's own set called partial tree & maintained in aqueue arbitrarily.
2. maintain each edges associated with each node or partial tree in priority queure ordered by the weight of the edges.
3. Select a partial tree from queue & find the minimum weight edge incident to the partial tree from priority queue.
4. find the practical tree that is connected by minimum path edge. Remove two sub tree joining by the edge & combine in to a single new tree & add to rear of the queue.
   b. Combine two priority queue of the node A & B (partial tree) & delete the edge connecting them from queue.
5. Repeat the algorithm unit a queue contains a single tree which is called minimum cost spanning tree.



Soln:-

Priority queue.	Partial tree weight of edges.
{4}	22, 24, 25
{3}	12, 18 22
{5}	10, 25
{6}	14, 18, 24
{0}	10, 28
{2}	12, 16
{1}	14, 16, 28

⇒

{3, 4}	12, 18, 24, 25
{0, 5}	25, 28
{1, 6}	16, 18, 24, 28
{2, 3, 4}	16, 18, 24, 25
{0, 2,3, 4, 5}	16, 18, 24, 28
{0, 1, 2, 3, 4, 5, 6}	all cancel

Here, weight of spanning tree = 10+25+22+12+16+14 =99

Greedy algorithm:-

Consider the problem of making changes Assume cosines of value 25ϕ (quarter), 10ϕ & (dine), 5ϕ (nick) , & 1ϕ (penny) & suppose we want to return 63 ϕ in change, almost without thinking , we convert this amount to two quarters, one dine & 3 pennies. Not only where we able to determine quickly lost of coins & the correct value but we produce certain list of value with that coin.

The algorithm probably used to select the largest will whose value was not greater than 63, & add it to the list of subtract it's value from 63 getting 38[(63-25) = 38]

We then select the largest coi9n whose value is not greater than 38 and add it to the list & so one. According to this

63-25 = 38
38-25 = 13
13-10 = 3
3-1 =2
2-1 =1
1-1 = 0

This method of making charge is a greedy algorithm.

At any individual stage a greedy algorithm selects that option which is locally optimum in same some particular sence. not that the greedy algorithm for making change produces on over al optimum solution only because of special properties of the coins. If the coins had value 1, 9 & 11 & we first select an 11 coin & then four 1 coins total of 5 coins.

We have seen several greedy algorithm such as Dijkstrals shortes path algorithm & kruskal's minimum cost spanning tree algorithm.

Kruskal's algorithm is also greedy as it picks from remaining edges the shortest among these that do not create a cycle.

```
/*Program of stack using Array */
# define max 5
Int top =-1;
Int stack_arr[Max];
Main()
{
Int choice;
While(1)
{
Printf("1.psh"\n");
Printf("2.pop");
Printf(3. display");
Printf( 4. quit");
Printf("Enter your choice")
```

```
Scanf(“%d”, choice);
Switch(choice)
{
Case 1:- psh();
Break;
Case 2: pop();
Break;
Case 3:- display()l;
Break;
Case 4: exit(4);
Default:
Printf(“wrong choice”);
}
}
}
Void push()
{
        Int pushed_item;
If(top== (max -1)
        Printf(“stack overflow”);
Else
{
        Printf(“Enter that item to be pushed in stack”);
        Scanf(“%d”, & pushed_item);
Top =top +1;
Stack_arr[top] = pushed_item;
}
}
Void pop()
{
If (top = =-1)
        Printf(“stack uinderflow”);
Else
{
        Printf(“popped element is %d”, stack_arr[top];
        Top = top -1;
}
}
Void display()
{
Int I;
If(top == -1)
        Printf(“Stack empty”);
Else
{
        Printf(“Stack element”);
For(i=top; I > = 0; i- -)
Printf(“%d”, stack_arr[i]);
}
}
```

/*Program of circular queue */

```
# define Max 5
Int cqueue_arr[Max];
Int front =-1;
Int rear =-1;
Main()
{
Int choice;
While (1)
{
Printf(“1. insert”);
Printf(2.delete”);
Printf(3.dispalay”);
Printf(4.quit”);
Printf(“Enter your choice”);
Scanf(“%d”, & choice);
Switch(choice)
{
Case 1: insert();
        Break;
Case 2: del();
        Break;
Case 3: display();
        Break;
Case 4: exit(1);
Default:
        Printf(“Wrong choice”);
}}}
Insert()
{
Int added_item;
If((front = = 0 & & rear = = Max -1)) || (front = = rear +1))
{
Printf(“Queue overflow”);
Return;
}
If (front = = -1)
{
Front = 0;
Rear = 0;
}
Else
If (rear = = max -1)
Return =0;
}
else
Reae = rear +1;
Printf("I/P element for insertion in queue");
Scanf("%d", & added_item);
Cqueue_arr[rear] = added item;
Dle()
{
If(front ==-1)
{
Printf("Queue underflow");
Return;
}
Printf("Element deleted from queue is %d", cqueue_arr[front]);
If (front == rear)
{
Front =-1;
Rear =-1;
}
Else
If(front = max-1)
Font =0;
Else
Front = front +1;
}
Display()
{
Int fornt _pos = front; rear_pos = rear;
If (front == -1)
{
Printf("queue is empty");
Return;
}
Printf("Queue elements");
```

```
If(front_pos<=rear_pos)
While(front_pos<=rear_pos)
{
Printf(%d", cqueue_arr[front_pos]);
Front_pos + +;
}
Font_pos= 0;
While (front_pos<= rear_pos]);
Front_pos =0;
Whiel(frot_pos<=rear_pos)
{
Printf(%d", cqueue arr[front_pos]);
Front_pos ++;
}}}
```

Output:-

```
1.  insert
delete
 display.
quit
enter Your choice:-1
input the element for insertion in queue   7
insert.  2. delete.  3. delete.  4. quit
enter your      choice 1. input = 8
                 "            input = 9
                +             input = 10
               "              input = 11
```

```
# Define Max 5
Int deque_arr[max];
Int left =-1;
Int right =-1;
Main()
{
Int choice;
Printf("1. I/P restricted dequeue");
Printf("2. O/P restricted dequeue");
Printf("Enter your choice");
Scanf("%D", & choice);
Switch(choice)
{
Case 1: input_que();
```

```
        Break;
Default: printf("Wrong choice");
}}
Input_que()
{
Input_que()
{
Int choice;
While(1)
{
Printf(1. insert at right");
Printf(2. delete from left");
Printf(3. delete from right");
Printf(4. display");
Printf(5. quite");
Switch(choice)
{
Case 1: insert_right();
        Break;
Case 2: delete_left();
        Break;
Case 3: delete_right();
        Break;
Case 4: display_queue();
        Break;
Case 5: exit();
Default: printf("Enter wrong choice");
}}}
Insert_right()
{
Int added_item;
If((left == 0 && right == max -1) || (lseft == right +1))
{
Printf("Queue overflow");
Return;
}
If(left ==-1)
{
Left =0;
Right =0;
}
Else
(f(right = max-1)
Right =0;
Else
Right = right +1;
Printf("I/P the element for adding in queue");
Scanf("%d", & added_item);
Deque_arr[right] = added_item;
}
Insert_lfet()
{
Int added_item;
If(left== 0&& right = max -1) || (left = right +1));
{
Printf("Queue overflow");
Return;
}
If(left ==-1
{
Left = 0;
Right =0;
}
Else
If(left = 0)
Left = max -1;
Else
Left = left-1;
Printf("I/P thelement for adding");
Scanf("%d", & added_item);
Deque_arr(left) = added_item;
}
Delete_left()
{
If(left ==-1)
{
Printf("Queue underflow"):
Return;
}
Printf("Element deleted from queue is %d", deque_arr[left]);
If(left == right)
{
Left =-1;
Right =-1;
}
Else
```

```
If()left== max-1)
Left =0;
Eles
Left = left +1;
}
Delete_right()
{
If(left ==-1)
{
Printf("Queue under flow");
Return;
}
Printf("Element deleted  from queue is %d", deque_arr[right]);
If(left = = right)
{
Left = -1;
Right =-1;
}
Else
If(right = =0)
Right = max -1;
Else
Right = rightg -1;
}
Display_queue()
}
```

Program of list using array

```
#defince Max 10
Int arr[max];
Int n;
Main()
{
Int choice, item_pos;
While(1)
{
Print("1. input list");
Printf("2. insert");
Printf("3. search");
Printf("4. display");
Printf("5. quit");
Printf("Enter your choice");
```

```
Scanf("%d", & choce);
Switch(choice)
{
Case 1:
        Printf("Enter the no. of element to be inserted");
        Scanf("%d", &n)
        Input(n);
        Break;
Case 2:
        Break;
Case 3. insert();
        Break;
Case 3: printf("Enter elements to be serched");
Scanf("%d", & item);
Pos = search(item);
If(p0os>=1)
Printf("%d found at postion %d", item, pos);
Else
Printf("element not found");
        Break;
Case 4:
        Del();
        Break;
Case 5: display();
        Break;
Case 6: exit();
        Break;
Default:
        Print("Wrong choice");
}}}
Input()
{
Int I;
For (I =0; i<n; i++)
{
Printf("I/P value for element %d",j+1);
Scanf(%d", &arr[i]);
}}
Int search *(int item)
{
Int I;
For (I =0; i<n;i++)
{
```

```
If (item == arr[i])
Return(i+1);
}
Return(0)                       /*if element not found */
}
Insert ()
{
Int temp, item, position;
If(n == max)
{
Printf("list overflow");
Return;
}
Printf("enter positionfor insertion")
Scanf("%d", & position);
Printf("Enter the value");
Scanf("%d", & item);
If position > n+1)
{
Printf("Enter position less than or equal to n+1);
Return;
}
If position = n+1;
}
Arr[n] = item
n = n+1
return;
}
/*insertion in between */
Temp  = n-1;
While (tem> = posiotn-1)
{
Arr[tem +1] = arr[temp];
Temp --;
}
Arr[position  -1] = item;
n = n+1;
}
Del()
{
Int tem , position, item;
If(n==0)
{
```

```
Printf("list underflow");
Return;
}
Printf("Enter the element to be deleted");
Scanf("%d", & item)
If (item = arr[n-1])
{
 N = n-1;
Retun;
}
Position = search (item);
If (position == 0)
{
Printf("Element not present in array");
Return;
}
//Deletion in between
Temp = position -1;
While (tem<=n-1)
{
Arr[temp] = arr[temp +1];
Temp ++
{
N = n-1;
}
Display()
{
Int I;
If (n = = 0)
{
Printf("List is empty");
Return;
}
For (I =0; i<n; i++)
Printf("value at position  % %d", i+1, arr[i]);
}
```

/* program of single linked list */

```
#include <stdio.h>
#include<malloc.h>
Struct node
        Int info;
        Struct node * link;
}* start;
Main()
{
{
Int choice n, m, position, I;
Start = NULL;
While(1)
{
Printf("1. create list \n");
Printf("2. ad at beginning \n");
Printf("3. add after \n");
Printf("4. delete \n");
Printf("5. display \n");
Printf("6. count \n");
Printf("7. reverse \n");
Printf("8. search \n");
Printf(" 9. quit \n");
Printf("Enter your choice");
Scanf(%d", & choice);
Switch (choice)
{
Case 1:  printf("How many nodes you want");
        Scanf( "%d", & n);
        For(i=0; i<n;i++)
        {
        Printf("Enter the element :");
        Scanf("%d"<&m);
        Create – list (m);
        }
        Break
Case 2:
        Printf("Enter the element:-");
        Scanf("%d", & m);
        Addetheg(m);
                Break;
Case 3:
        Printf("Enter the element:-");
        Scanf("%d", &m);
        Printf("Enter the position after which this element is );
        Scanf("%d", & position);
        Add after (m, position);
                Break;
Case 4:
        If(start = = NULL)
        {
        Printf("List is empty \n");
        Continue;
        }
        Printf("Enter the element for deletion ");
        Scanf('%d", &m)
        Del(m);
        Break;
Case 5: display();
        Break;
Case 6:  count();
        Break;
Case 7:  rev();
        Break;
Case 8:
        Printf("Enter the element  to be searched");
        Scanf("%d", &m);
        Search(m);
        Break;
Case 9:
        Exit()
Default: printf("Wrong choice \n");
}                /* end of switch  */
}                /* end of main() * /
Create – list (int data)
{
Struct node  *q, * temp;
Temp = malloc (size of (struct node));
Temp –→info = data;
Temp –→link = NULL;
If(start == NULL)              /*if list is empty*/
q =  start;
While(q → link ! = NULL)
        q = q → link;
        q → link = temp
}
Return;
}                        /* end of create – list is */
Addatbeg (int data)
{
Strct node * temp;
```

```
Temp = malloc (size of (struct node));
temp→ infor = data;
temp → link = start;
start = temp ;
return;
}                              /* End of addatbeg() */
Add after (int data, int pos)
{
Struct node * temp *q;
Int I;
Q start;
For (i=0; i<pos-1; i++)
{
Q = q→ link;
If(q == NULL)
{
Printf("There are less than %d element", pos);
Return;
}}                  /* end of for */
Temp = malloc (size of (struct node)
Temp → link = q → link;
Temp →info = data;
Q → link = temp;
Return;
}                              /*end of add after () */
Del (int data)
{
Struct node * temp, * q;
If(start → info = = data)
{
Temp = start;
Start start → link;                  /* first element deleted */
Free (temp);
Return;
}
Q = start;
While (q→ link→ link ! NULL)
{
If (q→ link → info = =  data)
{
Temp = q → link;
Q →link  = temp → link;
Free (temp);
Return;
}
Q = q→ link ;
/*end of while */
If (q→link →info = data)                    /* last element deleted */
{
Temp = q → link ;
Free (temp);
Q → link = NULL
Return;
}
Printf(“element %d not found \n”, data);
Return;
}                                  /* End of del() */
Display()
{
Strcut node  * q;
If (start == NULL)
{
Printf(“List is empty \n”);
Return;
}
q = start
printf(“List is: \n”)
while (q ! = NULL)
{
Printf(“%d”, a →info);
Q = q→ link;
}
Printf(“\n”);
Return;
}                                  /*End of display */
Count()
{
Struct node * q = start;
Int cnt = 0;
While (q ! = NULL)
{
Q = q →link;
Cnt ++ ;
}
```

```
Printf("No. of element are %d \n", cnt);
}                         /* End of count () */
Rev()
{
Struct node * P1, *P2, *P3;
If (start → link =  = NULL)                          / *only one element
*/
Return;
P1 = start;
P2 = P1 → link
P3 = P2 → Link;
P1 → Link = NULL
P2 →link = P1;
While (P3 ! = NULL)
{
P1 = P2;
P2 = P3;
P3 = P3 → link
P2 → link = P1;
}
Start = P2;
}                                        /*End of rev() */
Search(int data)
{
Struct node * ptr = start;
Int pas = 1;
While (ptr ! = NULL)
{
If (ptr → infor == data)
{
Printf("Item %d found at position %d", data, post);
Return;
}
Ptr = ptr → link;
Pos + + ;
}
If (ptr = = NULL)
Printf("Item %d not found in list \n", data);
}                                                  /* end of search() */
```