* Queue :-

Queue is a linear list of elements in which deletion can take place only at one end called the "front" and insertion can take place only at the other end called the "rear".

Queue are also called FIFO lists.

* Representation of Queue (Array)

Queue may be represented as array. Queue can be maintained by a linear array "QUEUE" and two pointer variables FRONT containing the location of the first element of queue, and REAR, containing the location of the rear element of the queue.
FRONT = NULL indicate that Queue is empty.

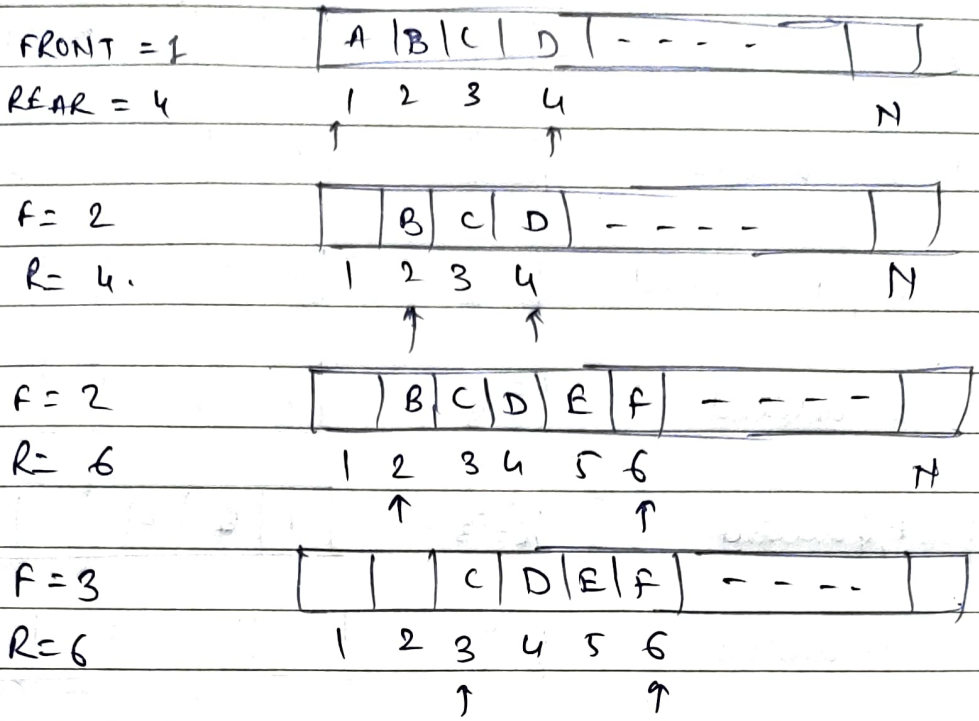
Queue → A → B → C → D.
'A' deleted B → C → D.
E, F inserted B → C → D → E → F
'B' deleted C → D → E → F.

whenever the element is deleted from the queue, the value of FRONT is increased by 1.

FRONT = FRONT + 1

Similarly, whenever an element is added to the queue the value of REAR is increased by 1.

REAR = REAR + 1



When Queue is full to right but space is available at left and we want to insert element, in this scenario, we will have to shift all elements to left. This is quiet expensive. Hence we consider that queue is circular, i.e. QUEUE[1] comes after QUEUE[N].

Ex:- N = 5 - Queue is initially empty.

a) Empty — F=0, R=0

b) A, B, C, inserted — F=1, R=3 — A B C

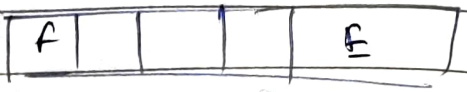
c) A deleted — F=2, R=3 — B C

d) D, E inserted — F=2, R=5 — B C D E

e) B, C, deleted — F=4, R=5 — D E

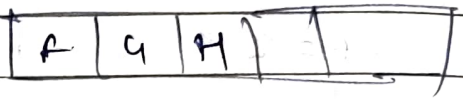
f) F inserted — F=4, R=1

g) D deleted — F=5, R=1



h) G, H inserted — F=5, R=3

i) E deleted — F=1, R=3



j) F, G deleted — F=3, R=3

k) H deleted — F=0, R=0

QINSERT (QUEUE, N, FRONT, REAR, ITEM)

① [Queue already filled?]
If FRONT = 1 and REAR = N or FRONT = REAR + 1
Write: OVERFLOW and Return.

② [Find new value of REAR]
If FRONT = NULL then
Set FRONT = REAR = 1
Else If REAR = N then
Set REAR = 1
Else
Set REAR = REAR + 1
[End of If].

③ Set QUEUE(REAR) = ITEM

④ Return.

QDELETE (QUEUE, N, FRONT, REAR, ITEM)

(1) [Queue already empty?].
If FRONT = NULL then
Write: UNDERFLOW and Return.

(2) Set ITEM = QUEUE[FRONT]

(3) [Find new value of FRONT]
If FRONT = REAR then [only one element in queue]
Set FRONT = REAR = NULL.
Else If FRONT = N then
Set FRONT = 1
Else
Set FRONT = FRONT + 1
[End of If structure].

(4) Return

* Linked List Representation of Queue.

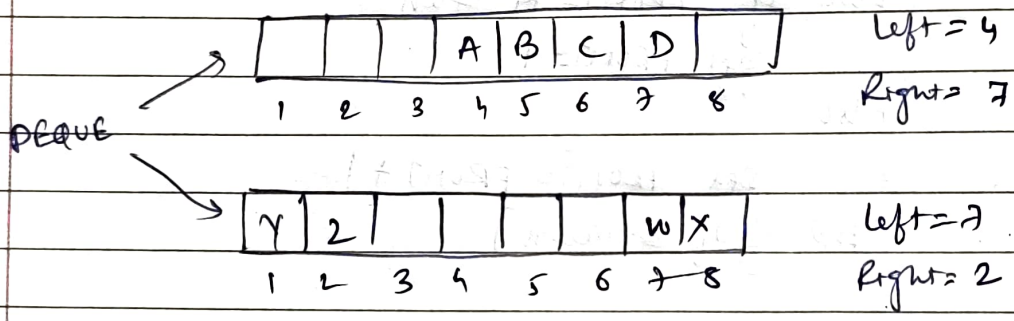
This will be covered later when we finish studying linked list.

Please go to Page No_____ for this topic.

DEQUES

A deque is a linear list in which elements can be added or removed at either end but not in the middle. The term is contraction of name "Double Ended Queue".

Unless and until implied we assume our deque is maintained by a circular array DEQUE with pointers LEFT and RIGHT, pointing to the two ends of deque.



Two variations of a deque -

(1) Input Restricted dequeue:-

It is a deque which allow insertions at only one end of the list but allows deletions at both end of the list.

(2) Output Restricted deque.

It is a deque which allows deletions at only one end of the list and allows insertions at both end of the list.

Assignment 1:-

3) Write algos to insert element from front & rear in DEQUE and for delete from front & rear in Deque.

PRIORITY QUEUES

A priority queue is a collection of elements such that elements has been assigned a priority and such that the order in which elements are deleted and processed" comes from the following rules :-

① An element of higher priority is processed before any element of lower priority.

② Two elements with same priority are processed according to the order in which they were added to the queue.

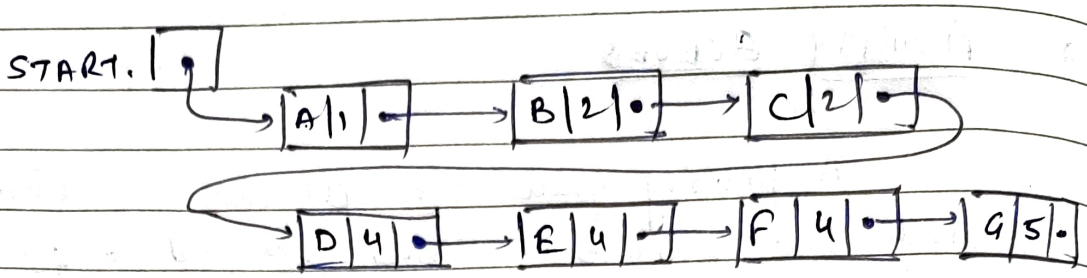
We will see two ways of maintaining a queue in memory. One that uses a one-way list and the other uses multiple queues.

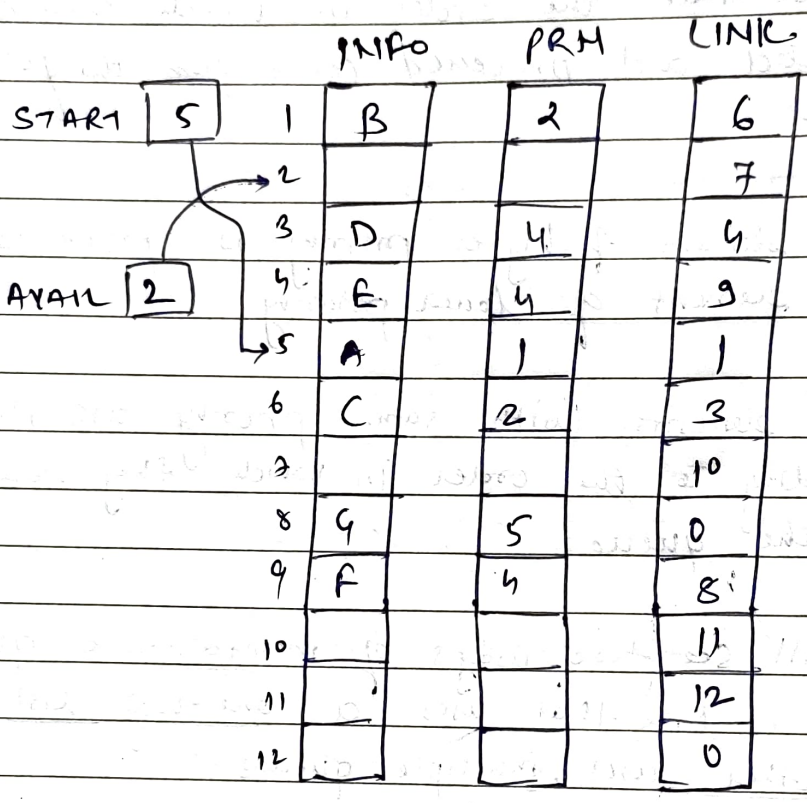
One-way list Representation of Priority Queue.

a) Each node in the list contains three items of information : an information field INFO, a priority number PRN and a link no. LINK.

b) A node 'x' precedes a node 'y' in the list
1) when 'x' has higher priority than 'y' or
2) both have same priority but 'x' was added to the list before 'y'.

Note : Lower the priority number, higher the priority.





Array Representation of PRIORITY QUEUE

Another way to maintain a priority Queue in a memory is to use separate queue for each level of priority (or each priority number).

Each such queue will appear in its own circular way and must have its own pair of pointers FRONT and REAR.

Thus, we can use a Two dimensional array to represent PRIORITY QUEUE instead of multiple linear arrays.

Example :-

	FRONT
1	2
2	1
3	0
4	5
5	4

	REAR
1	2
2	3
3	0
4	1
5	4

	1	2	3	4	5	6
1			A			
2	B	C	X			
3						
4	F				D	E
5				G.		

NOTE :- FRONT [K] and REAR [K] contains respectively the front and the rear elements of row 'k' of QUEUE, the row that contains the queue of elements with priority 'k'.

Algo to Delete

1. Find the smallest 'k' such that FRONT [K] ≠ NULL.
2. Delete and process the front element in row 'k' of QUEUE.
3. Exit.

Algo to INSERT

1. Insert ITEM as the rear element in row 'M' of QUEUE, where M = priority of ITEM.
2. Exit