CHAPTER - 4 : QUEUE

- Prof. Trupesh Patel.

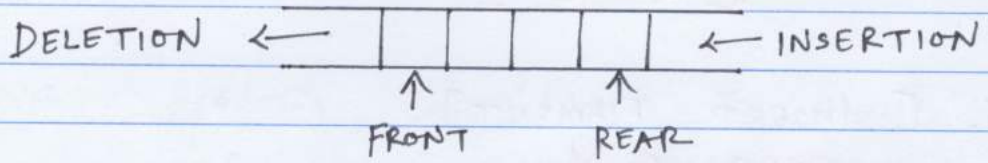
"A linear list which permits deletion to be performed at one end of the list and insertion at the other end is called Queue."

It follows FIFO (first in first out) or FCFS (first come first served) meechanism to process information.

FRONT - The end of queue from that deletion is to be performed.

REAR - The end of queue from that insertion is to be performed.

INSERTION = ENQUEUE
DELETION = DEQUEUE



Applications of Queue :-

- Queue of people at ticketing.
- Queue of processes in operating System.
- Job Scheduling in operating System use concept of Queue.
- Queue is used in shared resources.
- Queue is used in BFS (Breadth first Search) algorithm which eventually helps in tree traversal.
- Congestion in network can be resolved with Queue concept.

~ PROF. TRUPESH PATEL

Algorithm : Inserting an element in a queue.

QINSERT (Q, F, R, N, Y)

Here, Q - Queue is represented by vector Q.
F - pointer to the front element.
R - pointer to the Rear element.
Y - Y is to be inserted at Rear end of a queue.

1. [OVERFLOW ?]
   IF R >= N
   Then Write ('Queue Overflow')
   Return.

2. [Increment Rear Pointer]
   R ← R+1

3. [INSERT ELEMENT]
   Q[R] ← Y

4. [IS FRONT POINTER PROPERTY SET ?]
   IF F=0
   Then F ← 1
   Return

Algorithm : Deletes an element from a Queue.

QDELETE (Q, F, R)

Here, Q - Queue is represented by vector Q.
F - pointer to the front element of a Queue.
R - pointer to the Rear element of a Queue.

— PROF. TRUPESH PATEL

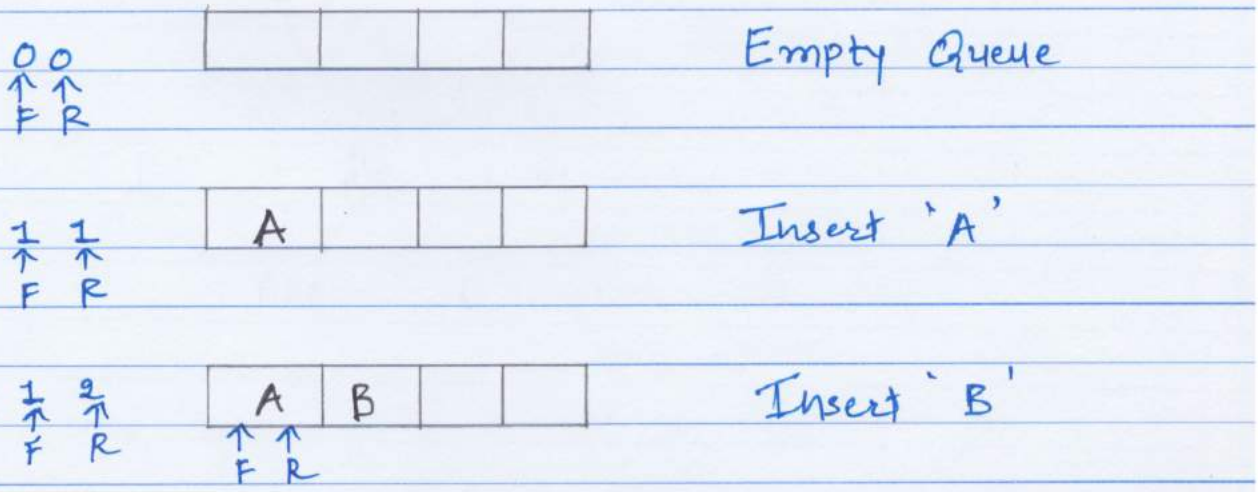
1. [Underflow ?]
   If F = 0
   Then Write (' UNDERFLOW ')
   Return 0 // 0 Denotes an empty queue

2. [Delete element]
   Y ← Q[F]

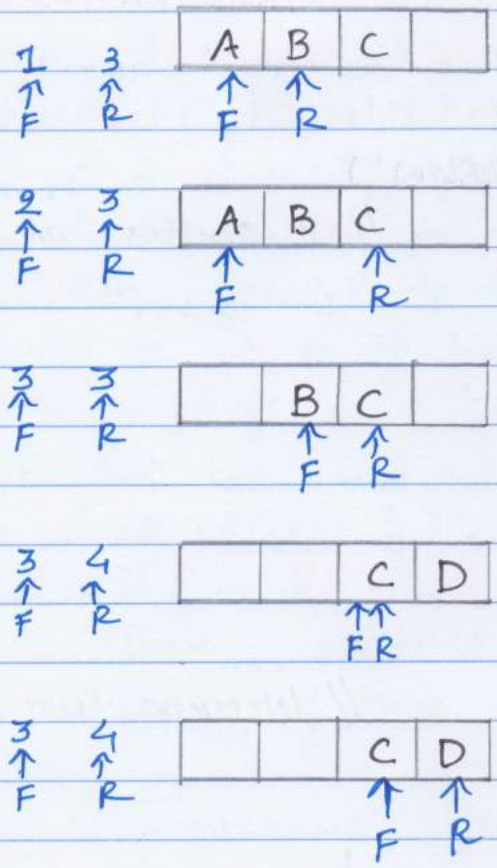
3. [Queue empty ?]
   If F = R
   then F ← R ← 0
   else F ← F + 1 // INCREMENT FRONT POINTER

4. [Return element]
   Return (Y)

EXAMPLE: perform operations on queue with size 4. show queue trace in each case.
Insert 'A', Insert 'B', Insert 'C', Delete 'A', Delete 'B', Insert 'D', Insert 'E'.



~ PROF. TRUPESH PATEL



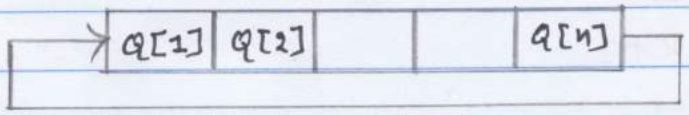
(R = 4) 7/ (N = 4) so Queue Overflow, but Space is there with Queue, this leads to the memory wastage.

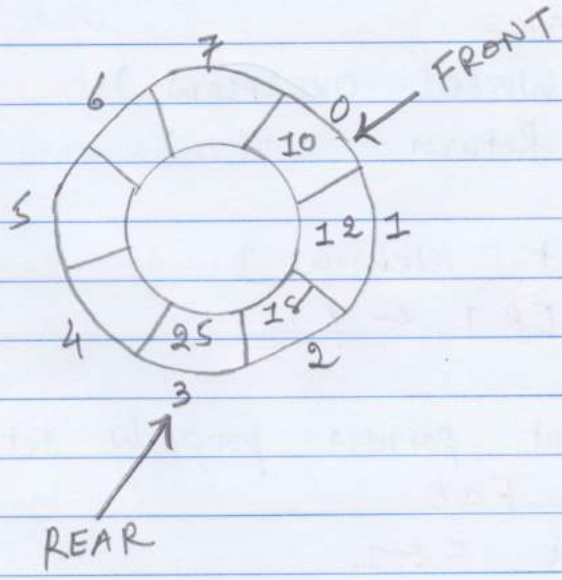
— PROF. TRUPESH PATEL

Circular Queue: A more suitable method of representing simple queue which prevents an excessive use of memory is to arrange the elements Q[1], Q[2], ... Q[n] in a circular fashion with Q[1] following Q[n], this is called Circular Queue.

In a circular Queue, the last node is connected back to the first node to make a circle.

Circular Queue follows FIFO principle.
It is also known as RING BUFFER.





— Prof. Trupesh Patel

Algorithm: Insert Y at the REAR end of circular queue.

CQINSERT (F, R, Q, N, Y)

Here, Q - Vector Q to represent queue.
F - Front pointer to the front element of a queue.
R - Rear pointer to the rear element of a queue.
Y - element to be inserted.

1. [Reset rear pointer ?]
   If R=N
   then R ← 1
   else R ← R+1

2. [overflow ?]
   If F=R
   then Write ('OVERFLOW')
   Return

3. [Insert element]
   Q[R] ← Y

4. [Is front pointer properly set ?]
   If F=0
   then F ← 1
   Return

~ Prof. Trupesh Patel

Algorithm: Delete last element of the queue.

CQDELETE (F, R, Q, N)

Here, F - pointer to the front element
R - pointer to the Rear element
Q - Vector Q represents queue.
N - Vector Q consist N elements.

1. [Underflow ?]
   IF F=0
   Then Write('Underflow')
   Return (0)

2. [Delete Element]
   Y ← Q[F]

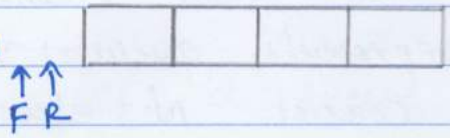
3. [Queue empty ?]
   If F=R
   then F ← R ← 0
   Return (Y)

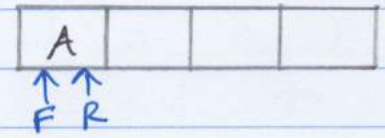
4. [Increment front pointer]
   IF F=N
   Then F ← 1
   Else F ← F+1
   Return (Y)

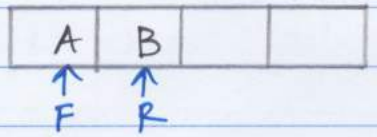
~ Prof. Trupesh Patel

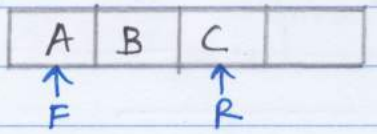
perform following operations on circular queue.

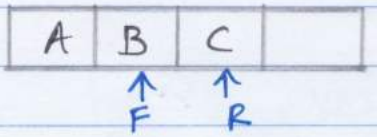
Insert 'A', Insert 'B', Insert 'C', Delete 'A', Delete 'B', Insert 'D', Insert 'E'.

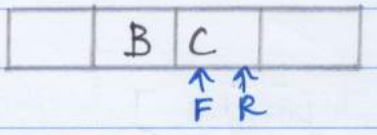
F=0, R=0  Empty Queue

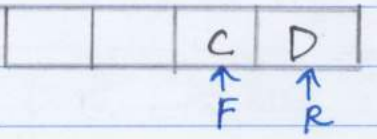
F=1, R=1  Insert 'A'

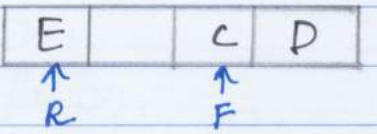
F=1, R=2  Insert 'B'

F=1, R=3  Insert 'C'

F=2, R=3  Delete 'A'

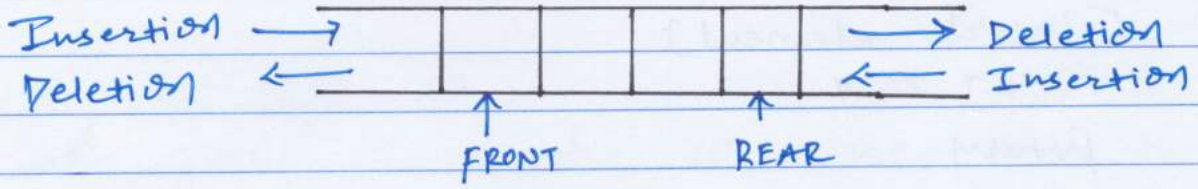
F=3, R=3  Delete 'B'

F=3, R=4  Insert 'D'

F=3, R=1  Insert 'E'

DQueue : A DQueue (double ended queue) is a linear list in which insertion and deletion are performed from either end of the structure.

Input Restricted dqueue : allows insertion at only one end
Output Restricted dqueue : allows deletion from only one end



Algorithms :

- DQINSERT_REAR is same as QINSERT (Enqueue)
- DQDELETE_FRONT is same as QDELETE (Dequeue)
- DQINSERT_FRONT
- DQDELETE_REAR

Algorithm : Inserts Y at front end of the circular queue.

DQINSERT_FRONT (Q, F, R, N, Y)

Here, Q = Queue is represented by a vector Q containing N elements.
F = pointer to front element of a queue.
R = pointer to rear element of a queue.

```
1. [Overflow ?]
   IF F=0
   Then  Write('Empty')
```

Return
If F = 1
Then Write ('Overflow')
Return

2. [Decrement front pointer]
F ← F-1

3. [Insert element]
Q[F] ← Y
Return

Algorithm: Delete and return an element from rear end of the Queue.

DQDELETE_REAR(Q,F,R)
Here, Q = Queue is represented by a Vector Q containing N elements.
F = pointer to the front element of a queue.
R = pointer to the Rear element of a queue.

1. [Underflow ?]
IF R=0
Then Write ('Underflow')
Return (0)

2. [Delete Element]
Y ← Q[R]

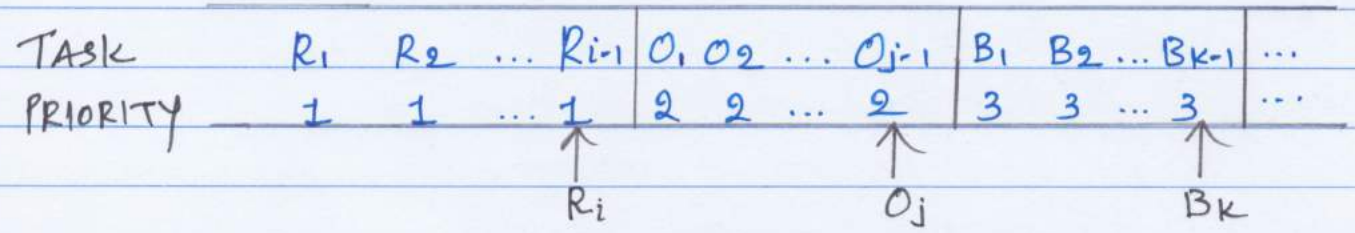
~ PROF. TRUPESH PATEL

3. [Queue Empty ?]
   IF R = F
   Then R ← F ← 0
   Else R ← R-1

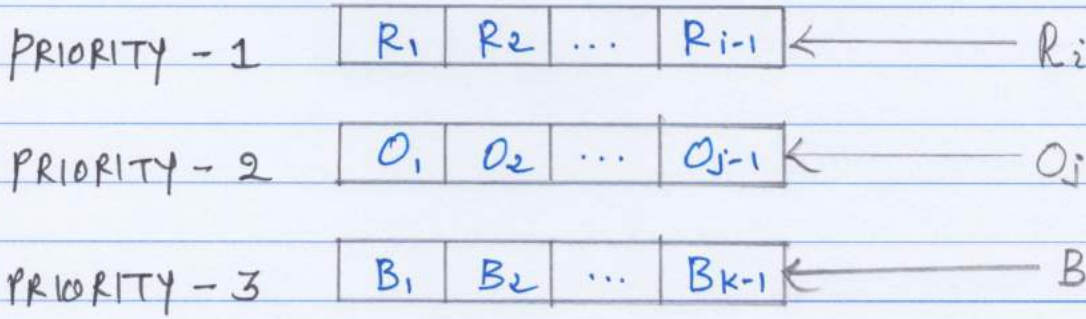
4. [Return Element]
   Return (Y).

~ PRIORITY QUEUE :-

66 A queue in which we are able to insert & remove items from any position based on some property (such as priority of the task to be processed) is often referred as PRIORITY QUEUE. 99



priority queue with insertion allowed at any position.



priority Queue viewed as a set of queue