هياكل البيانات

المرحلة الثانية

محاضرة(6)

م.م فرح معاذ جاسم

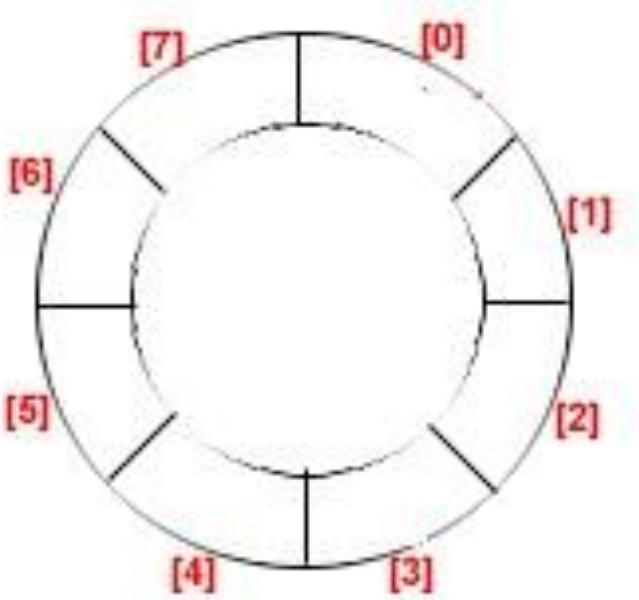
Farahmaath86@uoanbar.edu.iq

data st

1. CIRCULAR QUEUE DATA STURCTURE

Circular Queue is also a linear data structure, which follows the principle of **FIFO**(First In First Out), but instead of ending the queue at the last position, it again starts from the first position after the last, hence making the queue behave like a circular data structure.

In the other word the queue is considered as a circular queue when the positions 0 and MAX-1 are adjacent. Any position before front is also after rear.



Note:

Note that the container of items is an array. Array is stored in main memory. Main memory is linear. So this circularity is only logical. There cannot be physical circularity in main memory.

The drawback of circular queue is difficult to distinguished the full and empty cases. It is also known as **boundary case problem**.

In circular queue it is necessary that:

- Before insertion, fullness of Queue must be checked (for overflow).
- Before deletion, emptiness of Queue must be checked (for underflow).

Solution of the drawback of circular Queue

Use count variable to hold the current position (in case of insertion or deletion).

OPERATION OF CIRCULAR QUEUE

Some of the basic operations of the circular queue are as follows:

Initialize Operation.

1.Front: Returns the front (first,head) position in the circular queue

2.Rear: Returns the rear (last,tail) position in the circular queue

3. Insert: (value) is used to insert an element in the circular queue. The element is always inserted at the rear end of the queue.

4. Delete: (value) is used to delete an element from the circular queue

We follow the following sequence of steps to insert a new element in the circular queue.

1#Check if the circular queue is full:

test ((rear == MAXSIZE-1 && front == 0) || (rear == front-1)),

where 'SIZE' is the size of the circular queue.

2# If the circular queue is full then it displays a message as "Queue is full". If queue is not full then,

check if (rear == MAXSIZE – 1 && front != 0).

If it is true then set rear=0 and insert element.

For delete operation in a circular queue.

to delete an element from the queue. In the circular queue, the element is always deleted from the front end. Given **below is the sequence**

Steps:

#1) Check if the circular queue is Empty: check if (front==-1).

#2) If it is empty then display the message "Queue is empty". If queue is not empty then perform step 3.

#3) Check if (front==rear).

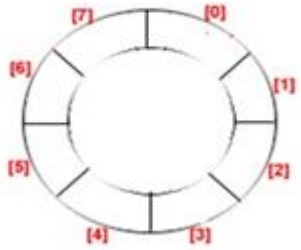
If it is true then set front=rear= -1

else check if (front==size-1),

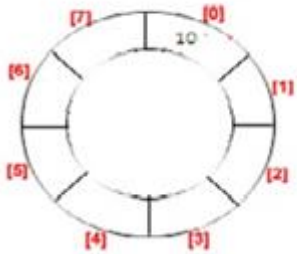
if it is true then set front=0 and return the element.

Example1: Consider the example with Circular Queue implementation

1) Initially: **Front =-1** and **rear = -1**

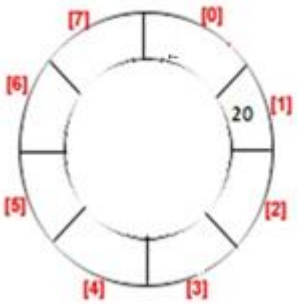


2) Add item 10 then **front = 0** and **rear =0.**



3) now insert 20 **front = 0** and **rear =1.**

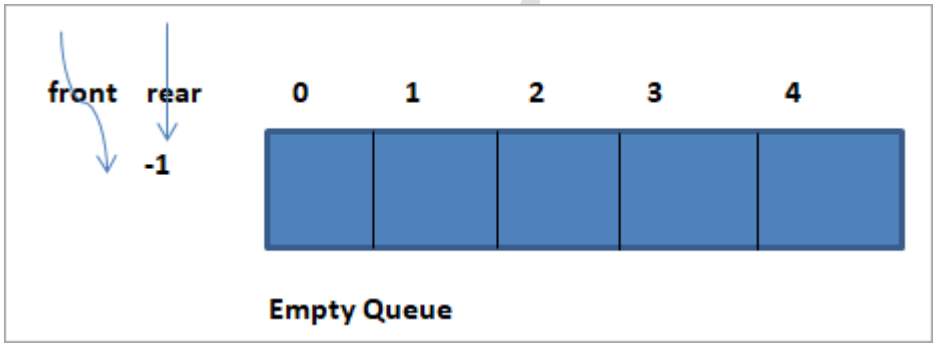
4) **delet()** **front = 1.** and **rear =1.**



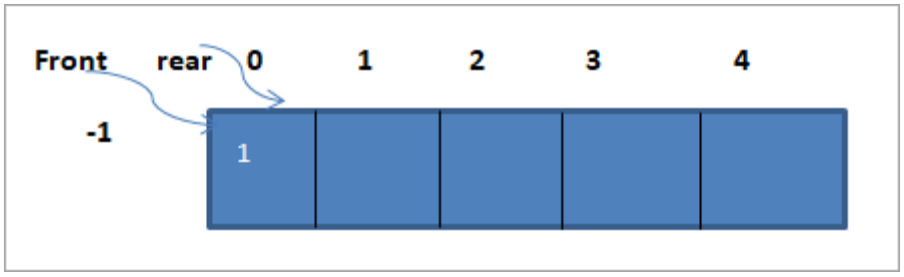
5) Like this now insert 30, 40, and 50,50,70,80 respectability then **front =1** and **rear = 7.**

6) Now in case of linear queue, we can not access 0 block for insertion but in circular queue next item will be inserted of 0 block then **front =0** and **rear = 0.**

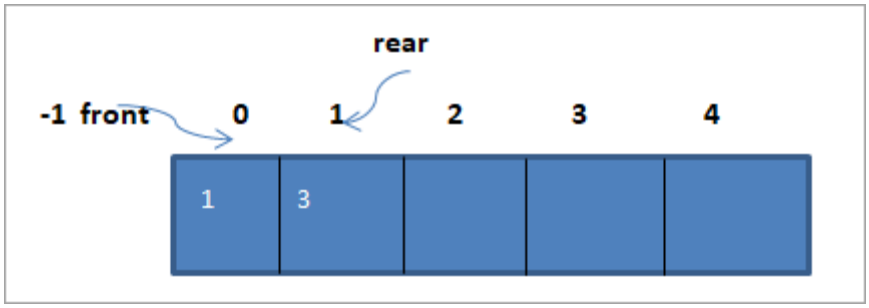
Example2: Consider the following circular queue of 5 elements as shown below:



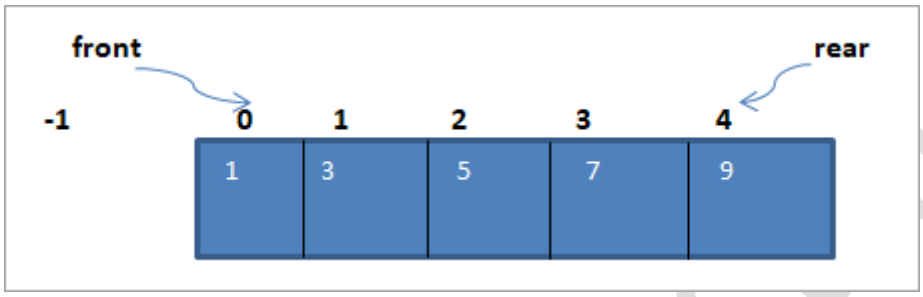
Next, we insert item 1 in the queue.



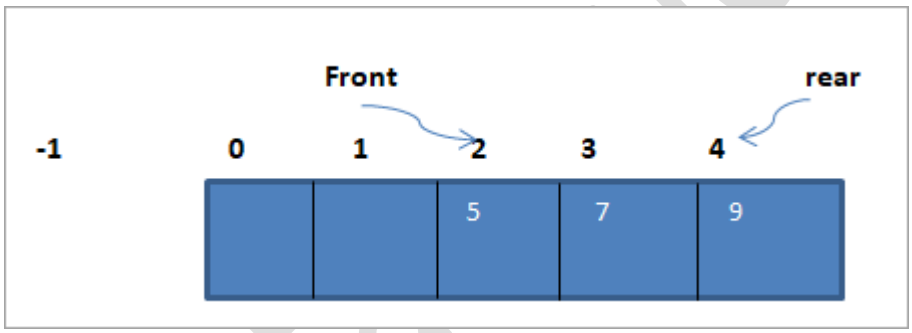
Next, we insert an item with value 3.



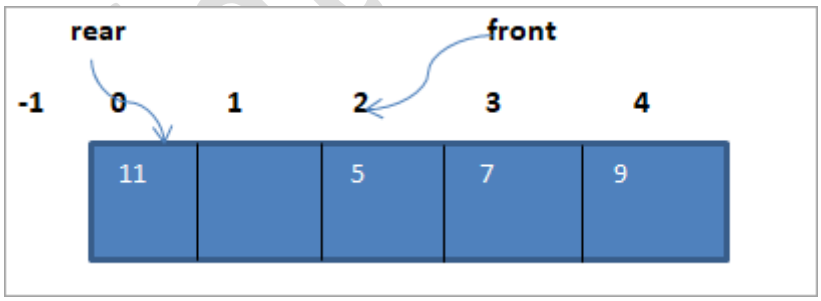
When we insert the elements to make a queue full, the representation will be as shown below.



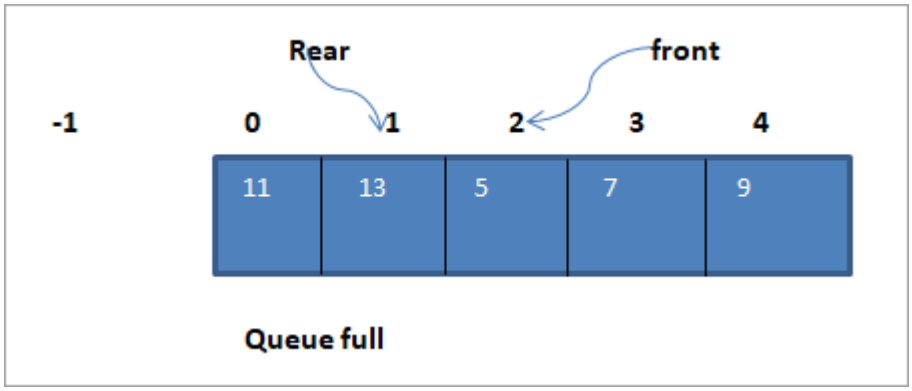
Now we delete the two elements i.e. item 1 and item 3 from the queue as shown below.



Next, we insert or enqueue element 11 in the circular queue as represented below.



Again let us insert element 13 in the circular queue. The queue will look as shown below.



APPLICATION OF CIRCULAR QUEUE

Below we have some common real-world examples where circular queues are used:

1. Computer controlled **Traffic Signal System** uses circular queue.
2. CPU scheduling and Memory management.

data structure