Queues

Introduction

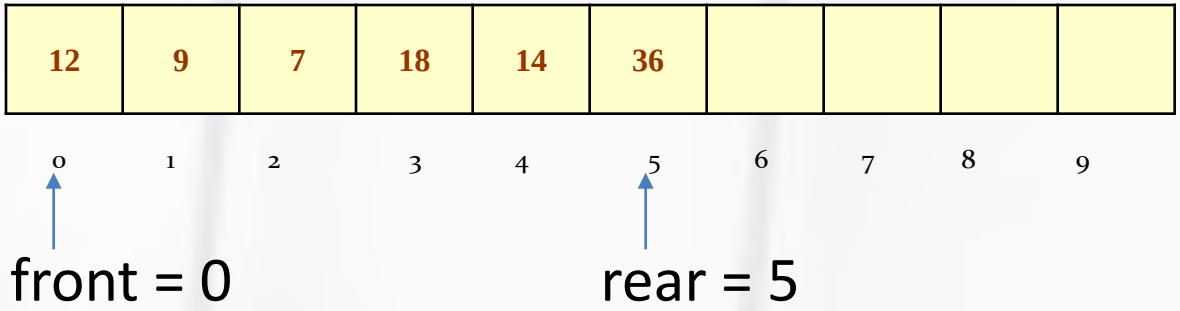
- Queue is an important data structure which stores its elements in an ordered manner.
- We can explain the concept of queues using the following analogy:

People moving on an escalator. The people who got on the escalator first will be the first one to step out of it.

- A queue is a FIFO (First-In, First-Out) data structure in which the element that is inserted first is the first one to be taken out.
- The elements in a queue are added at one end called the *rear* and removed from the other one end called the *front*.

Array Representation of Queues

- Queues can be easily represented using linear arrays.
- Every queue has front and rear variables that point to the position from where deletions and insertions can be done, respectively.
- Consider the queue shown in figure

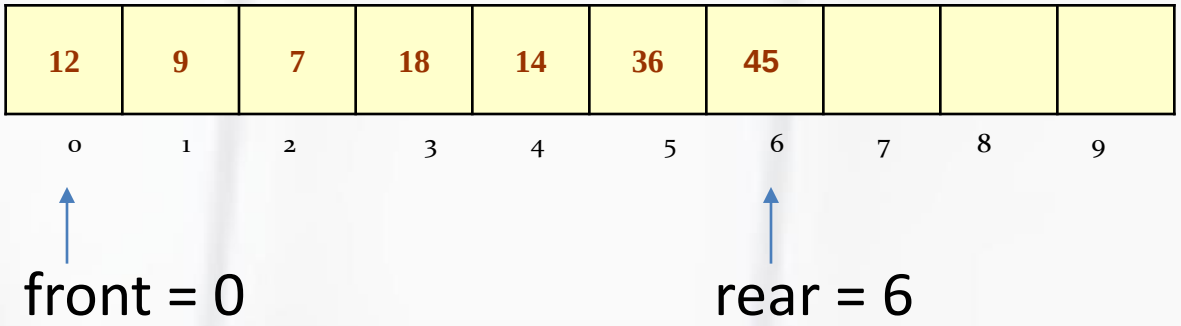


- Here, front = 0 and rear = 5.

0 1 2 3 4 5 6 7 8 9

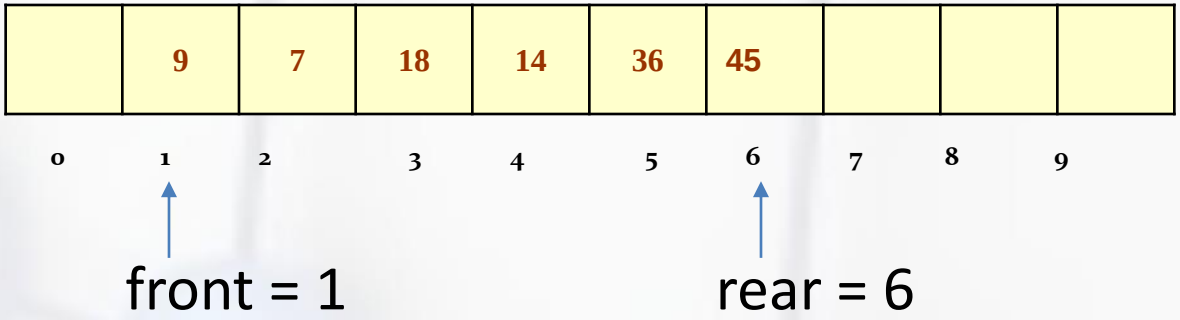
Array Representation of Queues

- If we want to add one more value in the list say with value 45, then rear would be incremented by 1 and the value would be stored at the position pointed by rear.



Array Representation of Queues

- Now, front = 0 and rear = 6. Every time a new element has to be added, we will repeat the same procedure.
- Now, if we want to delete an element from the queue, then the value of front will be incremented. Deletions are done from only this end of the queue.



- Now, front = 1 and rear = 6.

Array Representation of Queues

- Before inserting an element in the queue we must check for **overflow** conditions.
- An **overflow** occurs when we try to insert an element into a queue that is already full, i.e. when **rear = MAX – 1**, where MAX specifies the maximum number of elements that the queue can hold.
- Similarly, before deleting an element from the queue, we must check for **underflow** condition.
- An **underflow** occurs when we try to delete an element from a queue that is already empty. If front = -1 and rear = -1, this means there is no element in the queue.

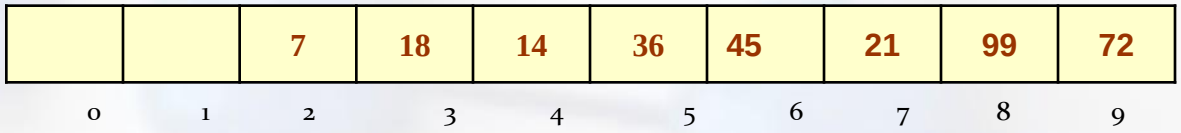
Algorithm for Insertion Operation

```
Algorithm to insert an element in a queue
Step 1: IF REAR=MAX-1, then;
            Write OVERFLOW
            Goto Step 4
        [END OF IF]
Step 2: IF FRONT == -1 and REAR = -1, then
            SET FRONT = REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = NUM
Step 4: Exit
```

Algorithm for Deletion Operation

```
Algorithm to delete an element from a queue
Step 1: IF FRONT = -1 OR FRONT > REAR, then
                  Write UNDERFLOW
                  Goto Step 2
        ELSE
                  SET VAL = QUEUE[FRONT]
                  SET FRONT = FRONT + 1

        [END OF IF]
Step 2: Exit
```

Circular Queues

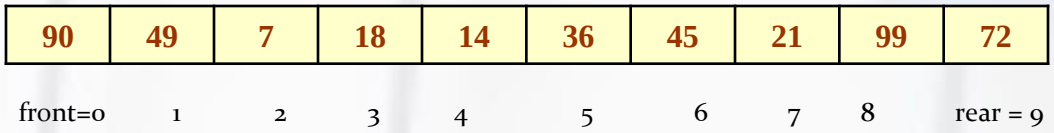


- We will explain the concept of circular queues using an example.
- In this queue, front = 2 and rear = 9.
- Now, if you want to insert a new element, it cannot be done because the space is available only at the left of the queue.
- If rear = MAX – 1, then OVERFLOW condition exists.
- This is the major drawback of a linear queue. Even if space is available, no insertions can be done once rear is equal to MAX – 1.
- This leads to wastage of space. In order to overcome this problem, we use circular queues.
- In a circular queue, the first index comes right after the last index.
- A circular queue is full, only when front=0 and rear = Max – 1.

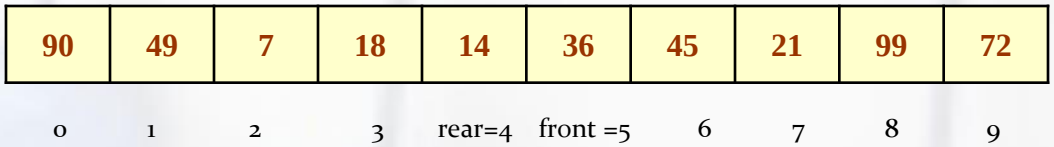
Inserting an Element in a Circular Queue

- For insertion we check for three conditions which are as follows:

- If front=0 and rear= MAX – 1, then the circular queue is full.

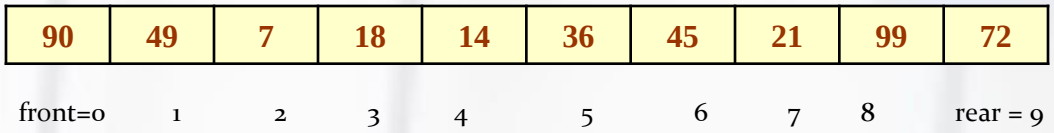


- If Rear+1 = Front, then also the circular queue is full.

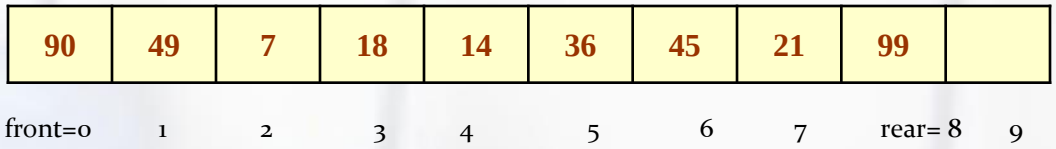


Inserting an Element in a Circular Queue

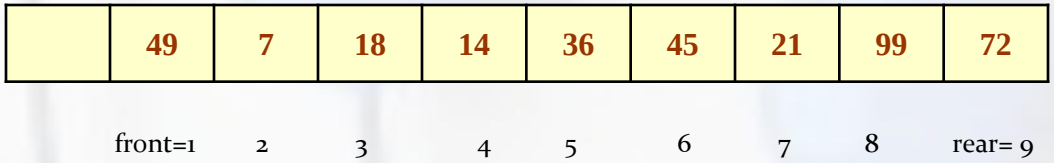
- For insertion we check for three conditions which are as follows:
- If front=0 and rear= MAX – 1, then the circular queue is full.



- If rear != MAX – 1, then the rear will be incremented and value will be inserted



- If front!=0 and rear=MAX -1, then it means that the queue is not full. So, set rear = 0 and insert the new element.

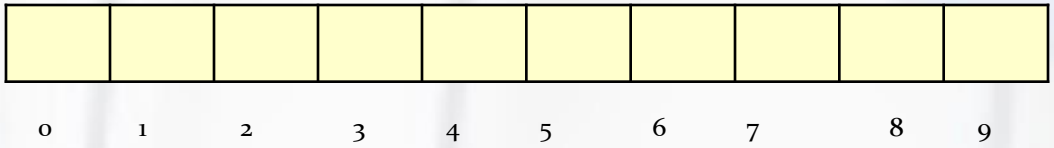


Algorithm to Insert an Element in a Circular Queue

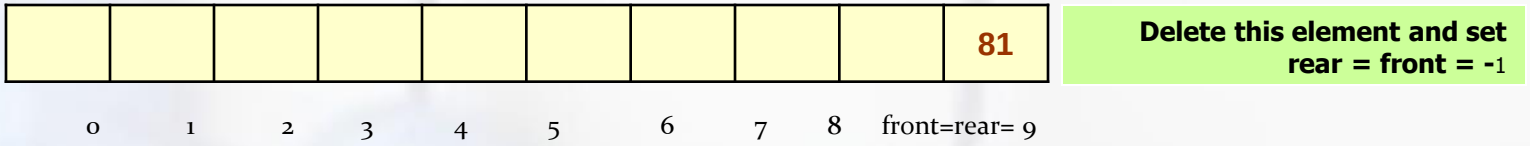
```
Step 1:IF (FRONT = 0 and Rear = MAX - 1) or (Rear+1 = Front,
then
                Write "OVERFLOW"
                Goto Step 4
        [END OF IF]
Step 2: IF FRONT = -1 and REAR = -1, then;
                SET FRONT = REAR = 0
        ELSE IF REAR = MAX - 1
                SET REAR = 0
        ELSE
                SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = VAL
Step 4: Exit
```

Deleting an Element from a Circular Queue

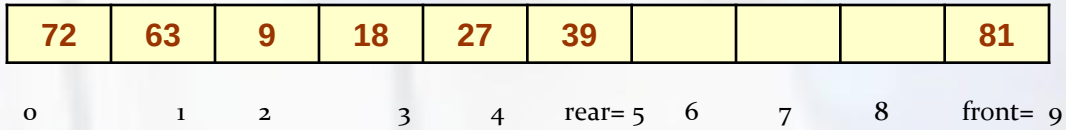
- To delete an element again we will check for three conditions:
- If front = -1, then it means there are no elements in the queue. So an underflow condition will be reported.



- If the queue is not empty and after returning the value on front, if front = rear, then it means now the queue has become empty and so front and rear are set to -1.



- If the queue is not empty and after returning the value on front, if front = MAX -1, then front is set to 0.

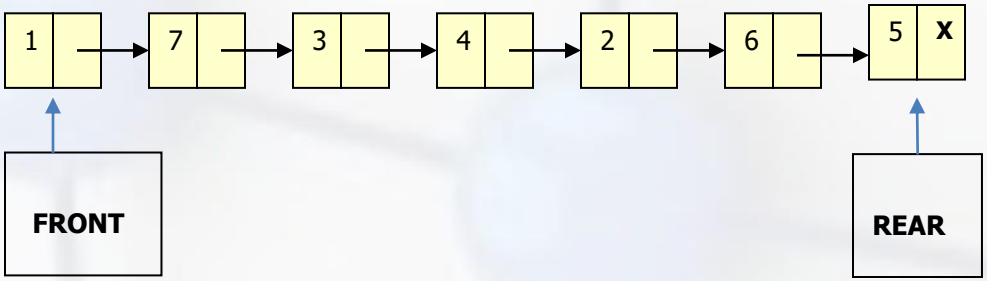


Algorithm to Delete an Element from a Circular Queue

```
Step 1: IF FRONT = -1, then
                Write "Underflow"
                Goto Step 4
        [END OF IF]
Step 2: SET VAL = QUEUE[FRONT]
Step 3: IF FRONT = REAR
                SET FRONT = REAR = -1
        ELSE
                IF FRONT = MAX -1
                        SET FRONT = 0
                ELSE
                        SET FRONT = FRONT + 1
                [END OF IF]
        [END OF IF]
Step 4: EXIT
```

Linked Representation of Queues

- In a linked queue, every element has two parts: one that stores data and the other that stores the address of the next element.
- The START pointer of the linked list is used as FRONT.
- We will also use another pointer called REAR which will store the address of the last element in the queue.
- All insertions will be done at the rear end and all the deletions will be done at the front end.
- If FRONT = REAR = NULL, then it indicates that the queue is empty.



Inserting an Element in a Linked Queue

```
Algorithm to insert an element in a linked queue

Step 1: Allocate memory for the new node and name it as
PTR
Step 2: SET PTR->DATA = VAL
Step 3: IF FRONT = NULL, then
                SET FRONT = REAR = PTR
                SET FRONT->NEXT = REAR->NEXT = NULL
        ELSE
                SET REAR->NEXT = PTR
                SET REAR = PTR
                SET REAR->NEXT = NULL
        [END OF IF]
Step 4: END
```

Deleting an Element from a Linked Queue

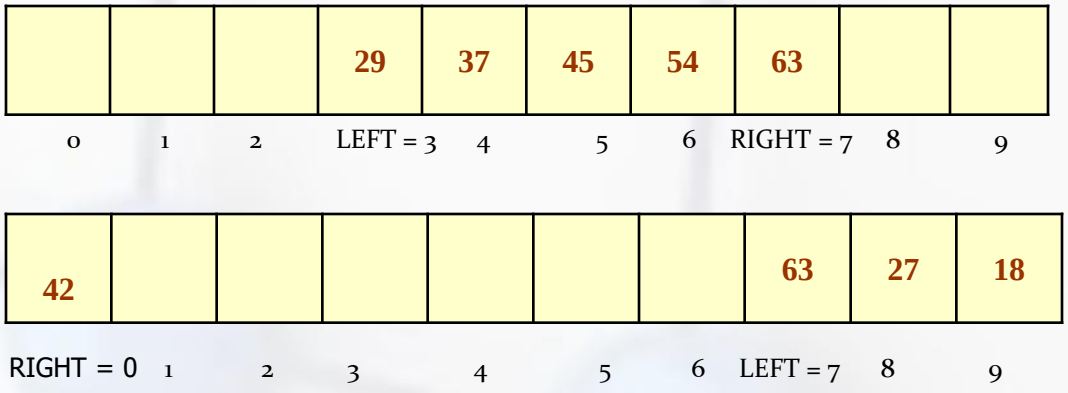
```
Algorithm to delete an element from a linked queue

Step 1: IF FRONT = NULL, then
                Write "Underflow"
                Go to Step 5
        [END OF IF]
Step 2: SET PTR = FRONT
Step 3: FRONT = FRONT->NEXT
Step 4: FREE PTR
Step 5: END
```

Deques

- **Deque or Double Ended Queue** is a **generalized version of Queue** data structure that allows insert and delete at both ends.
- A deque can be implemented either using a circular array or a circular doubly linked list.
- Linked list implementation of deque also known as a **head-tail linked** list because elements can be added to or removed from the front (head) or back (tail).

Deques

- In a deque, two pointers are maintained, LEFT and RIGHT which point to either end of the deque.
- The elements in a deque stretch from LEFT end to the RIGHT and since it is circular, Dequeue[MAX-1] is followed by Dequeue[0].



Deques

- **Operations on Deque:**
  Mainly the following four basic operations are performed on queue:
- ***insertFront()***: Adds an item at the front of Deque.
  insertRear(): Adds an item at the rear of Deque.
  deleteFront(): Deletes an item from front of Deque.
  deleteRear(): Deletes an item from rear of Deque.
- A deque is a generalized data structure that can be used as regular ***queue*** or ***stack***.
- As a Queue: Use only ***deleteFront()*** *and* ***insertRear()*** *operations.*
- As a stack: Use only ***insertRear()*** *and* ***deleteRear()*** *operations.*

Deques

- There are two variants of a double-ended queue:

➢ ***Input restricted deque:*** In this dequeue insertions can be done only at one of the ends while deletions can be done from both the ends.

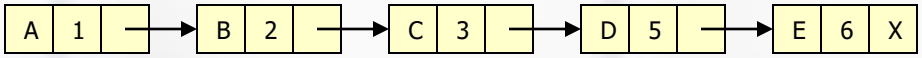
➢ ***Output restricted deque:*** In this dequeue deletions can be done only at one of the ends while insertions can be done on both the ends.

Priority Queues

- A **priority queue** is a queue in which each element is assigned a priority.
- The priority of elements is used to determine the order in which these elements will be processed.
- The general rule of processing elements of a priority queue can be given as:
  - An element with higher priority is processed before an element with lower priority.
  - Two elements with same priority are processed on a **first come first served** (FCFS) basis.
- Priority queues are widely used in **operating systems** to execute the highest priority process first.
- In computer's memory, priority queues can be represented using arrays or linked lists.

Linked Representation of Priority Queues

- When a priority queue is implemented using a linked list, then every node of the list contains three parts: (i) the information or data part, (ii) the priority number of the element, (iii) and address of the next element.
- If we are using a sorted linked list, then element having higher priority will appear before the element with lower priority.

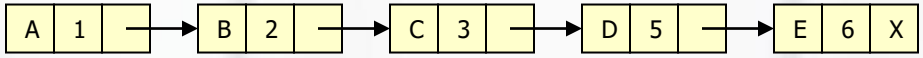


- **Note:** Assumes lower the priority field value, higher the priority.

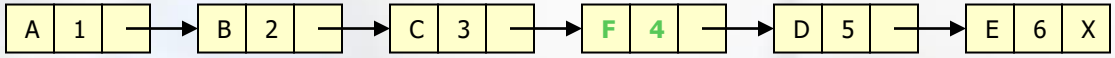
Linked Representation of Priority Queues

Insertion into a priority queue:

- Traverse the list until we find a node that has a priority lower than that of the new element.
- The new node is inserted before the node with the lower priority.



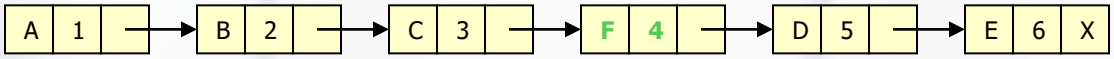
Priority queue after insertion of a new node F with priority 4.



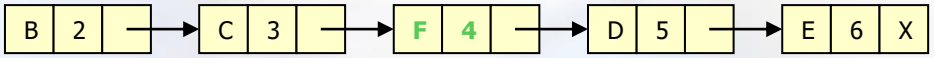
Linked Representation of Priority Queues

Deletion a priority queue:

- From a priority queue, element with highest priority is removed first.
- In this case, elements are stored in descending order of priority. Hence the first element is removed.



Priority queue after deletion of a highest priority element.



Array Representation of Priority Queues

- When arrays are used to implement a priority queue, then a separate queue for each priority number is maintained.
- Each of these queues will be implemented as circular queues. Every individual queue will have its own FRONT and REAR pointers.
- We can use a two-dimensional array for this purpose where each queue will be allocated same amount of space.
- Given the front and rear values of each queue, a two- dimensional matrix can be formed.

Array Representation of Priority Queues

FRONT	REAR
3	3
1	3
4	5
4	1

	1	2	3	4	5
1			A		
2	B	C	D		
3				E	F
4	I			G	H

Figure 8.29 Priority queue matrix

FRONT	REAR
3	3
1	3
4	1
4	1

	1	2	3	4	5
1			A		
2	B	C	D		
3	R			E	F
4	I			G	H

Figure 8.30 Priority queue matrix after insertion of a new element

Applications of Queues

- Queues are widely used as waiting lists for a single resource like printer, disk, CPU etc. hared by many user agents.
- Queues are used to transfer data asynchronously between different applications/processes running in a system(e.g., IPC mechanism pipes, sockets use queues).
- Queues are used as buffers on devices MP3 players and portable CD players, iPod playlist.
- Queues are used in Playlist for music player apps to add songs to the end, play from the front of the list.