Circular Queue

By Er. Kushal Ghimire

Concept

Linear Queue:

Rear = Rear + 1

Front = Front + 1

Circular Queue:

Rear = (Rear + 1) % MAX

Front = (Front + 1) % MAX

Fig: Linear Queue

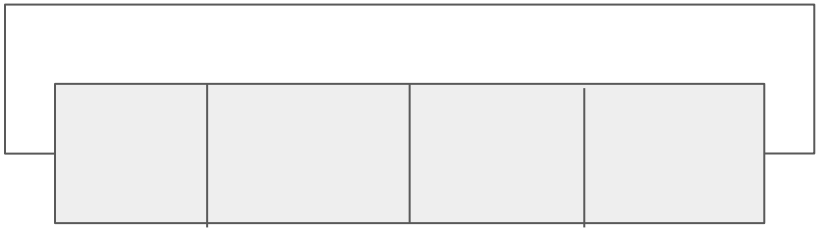
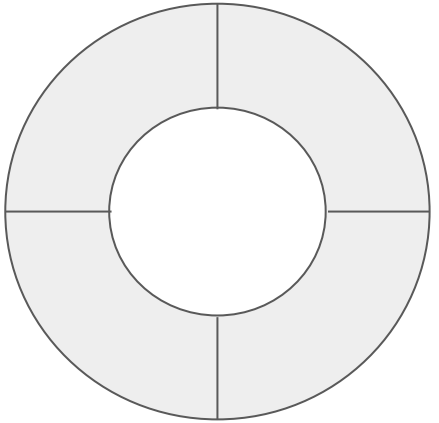


Fig: Circular Queue



Problems of Linear Queue

Memory Waste

- Enqueue operation on linear queue is performed through rear. Once the element is enqueued, rear pointer is incremented by 1 to point to the next location. **The rear pointer can point to each index of our queue only once as the rear pointer can only be incremented.**
- Dequeue operation on linear queue is performed through front. Once the element is dequeued, front pointer is incremented by 1 to point to the next location. **The front pointer can point to each index of our queue only once as the front pointer can only be incremented.**
- The empty space that are available after the dequeue operation are not accessible through the rear pointer. Since, the enqueue operation can only be done through rear end, thus, inaccessible memory spaces after dequeue operations, are wasted.

Memory Wastage in Linear Queue: W = Wasted Memory

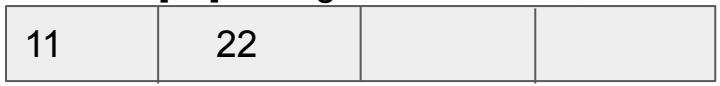
1) Initially, our linear queue is empty. Say, r = -1, f = -1

r=-1, f=-1

2) Enqueue an element [11] through rear. i.e r = 0, f = 0

r=0, f=0

3) Enqueue an element [22] through rear. i.e r = 1, f = 0



f=0 r=1

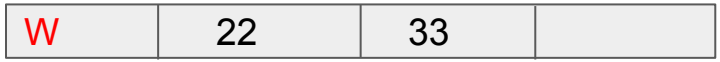
4) Enqueue an element [33] through rear. i.e r = 2, f = 0

f=0 r=2

1) After last insertion: f= 0, r = 2

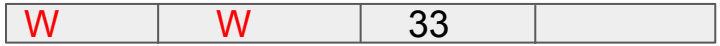
f=0 r=2

2) Dequeue an element through front.
i.e r = 2, f = 1



f=1 r=2

3) Dequeue an element through front.
i.e r = 2, f = 2



f=2, r=2

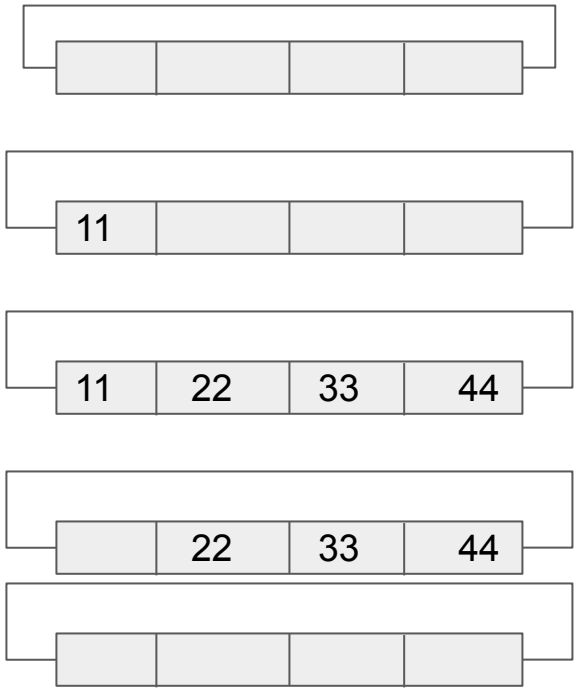
Why do we need to leave one space empty in circular queue ?

To distinguish the full and empty case of a circular queue, one empty space is sacrificed in a circular queue. Let us demonstrate the problem with an example. Let SIZE of queue is 4.

- Initially, queue is empty i.e. r= -1, f = 0
- After one enqueue(11) operation, r = 0, f = 0
- Let us enqueue **till the queue is full.**
  - enqueue(22) => r = 1, f = 0
  - enqueue(33) => r = 2, f = 0
  - enqueue(44) => r = 3, f = 0
- After one dequeue() operation, r = 3, f = 1
- Let us dequeue **till the queue is empty.**
  - Dequeue() => r = 3, f = 2
  - Dequeue() => r = 3, f = 3
  - Dequeue() => r = 3, f = 0
- We can clearly see that the rear and front pointer are on the exact same position while the queue was empty as well as while the queue is full. Hence, to overcome this problem, we can simply leave an empty space(sacrifice one space) to distinguish between the full and empty case.

Why do we need to leave one space empty in circular queue ?

To distinguish the full and empty case of a circular queue, one empty space is sacrificed in a circular queue.



Initial Condition: r = -1, f = 0

One insertion
Enqueue(11): r = 0, f = 0

Insert till queue is full
Enqueue(22): r = 1, f = 0
Enqueue(33): r = 2, f = 0
Enqueue(44): **r = 3, f = 0**

One deletion
Dequeue(): r = 3, f = 1

Delete till queue is empty
Dequeue(): r = 3, f = 2
Dequeue(): r = 3, f = 3
Dequeue(): **r = 3, f = 0**

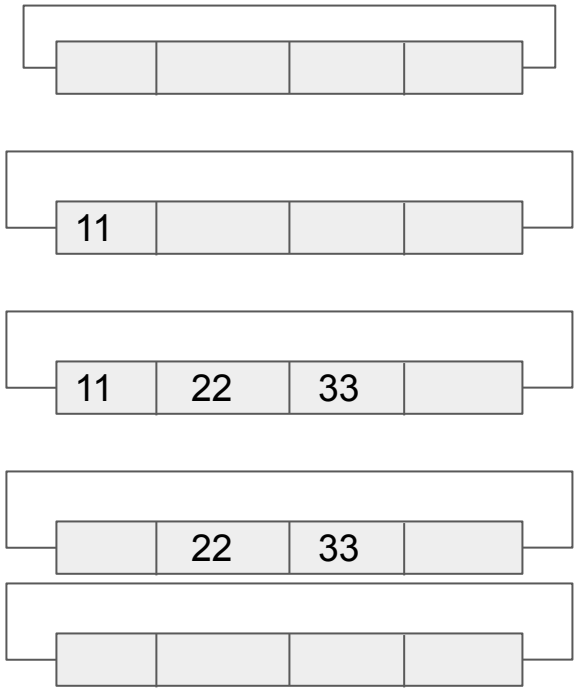
Hence we can clearly see that the full and empty case are same.
i.e.

Initial Condition : **r = -1, f = 0**
Queue Full : **(r+1)%MAX == f**
Queue Empty : **(r+1)%MAX == f**

To overcome this situation, we leave one extra space.

Why do we need to leave one space empty in circular queue ?

To distinguish the full and empty case of a circular queue, one empty space is sacrificed in a circular queue.



Initial Condition: r = -1, f = 0

One insertion
Enqueue(11): r = 0, f = 0

Insert till queue is full
Enqueue(22): r = 1, f = 0
Enqueue(33): r = 2, f = 0

One deletion
Dequeue(): r = 2, f = 1

Delete till queue is empty
Dequeue(): r = 2, f = 2
Dequeue(): r = 2, f = 3

Hence we can clearly see that the full and empty case differs when we leave one space.
Thus,

Initial Condition : **r = -1, f = 0**
Queue Full : **(r+2)%MAX == f**
Queue Empty : **(r+1)%MAX == f**