Date 11/02/23

{Data Structures}

{Queues:: A queue is an ordered list In which Insertions are done at one End (rear) and deletions at other End (front). The first Element Inserted is the first one to be delete. FIFO/LILO.

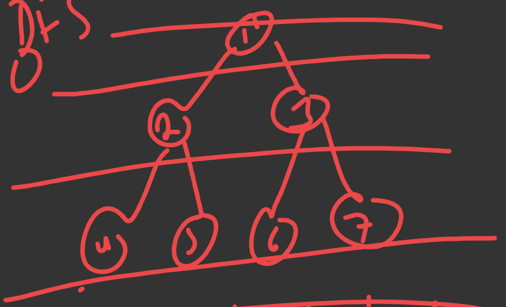
There are two basic operations on queue.

1) Enqueue:: An Element is Inserted in a queue.

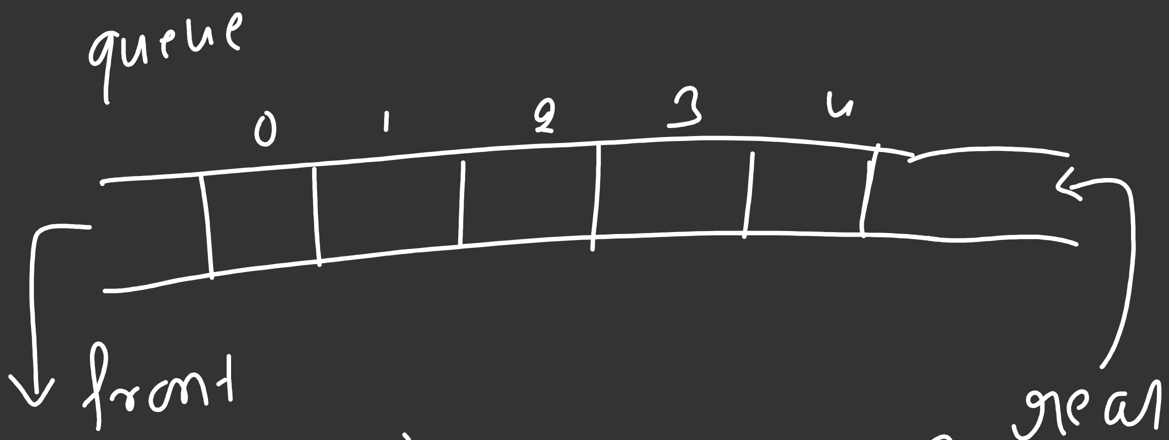
2) Dequeue:: An Element is removed from the queue.

Note:: If queue is full → overflow (when you want to Insert an Element)
If queue is Empty → underflow (—— " —— delete an Element)

BFS:



1	2	3	4	5	6	7



(deletion End)

Dequeue

(Insertion End)

(Enqueue)

Other operations::

- QueueSize( )
- IsEmptyQueue( )
- front( )

Applications of queue

- Job Scheduling in OS
- Multiprogramming.
- Waiting times of Customer at Call Center.

Implementation of queue::

- Array
- Linked lists.

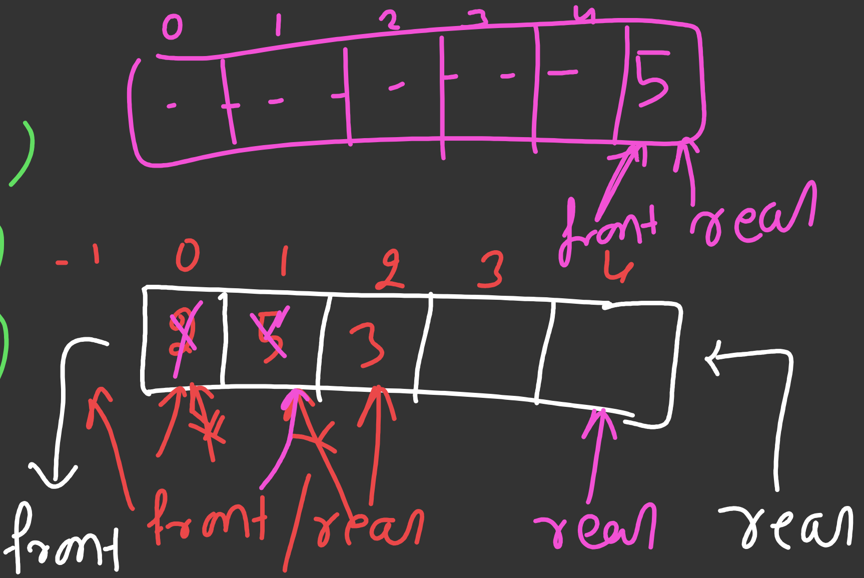
n=5

Queue[n]

Enqueue O(1)

Dequeue O(1)

find element O(1)



Initially

front = rear = - 1

Enqueue (2)

front++
rear++

Enqueue(5)

rear++

underflow

If (front == rear == -1)

Overflow

If (rear == n-1)

front++
rear++

Dequeue

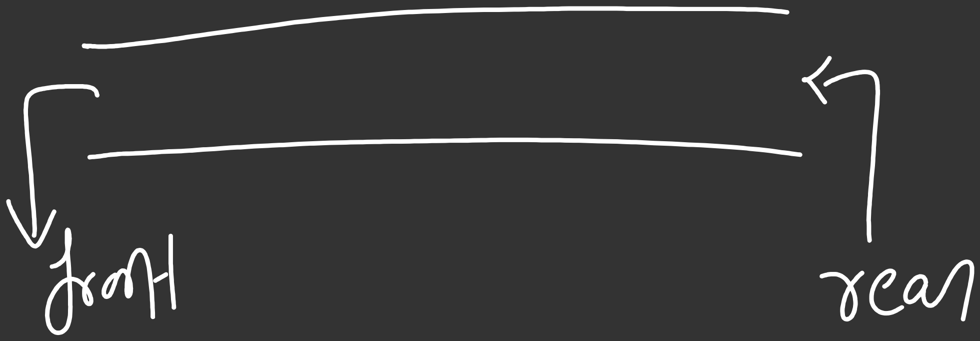
front++

Enque(5)

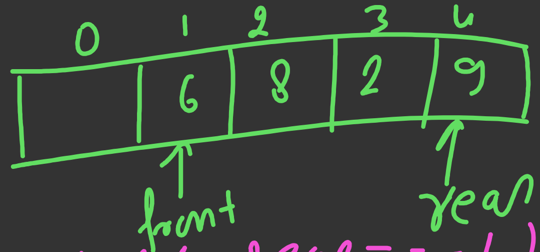
rear++

Drawback of ~~array~~ queue implementation ~~of~~ using array.

```
front = rear = -1 ;
Enqueue(x)
{
    If(rear == n-1)
    {  "overflow"
    }
  Else If ( front==-1 && rear==-1)
  {
        front++ ; rear++ ;
          Queue[rear]=x;
  }
  Else{
        rear++ ;
  }                         Copy
}
```



```
main( )
{      n=5        Queue[n]
    Enqueue(8)
    Enqueue(6)
    Enqueue(5)
    Traverse( )
    Dequeue( )
}
```

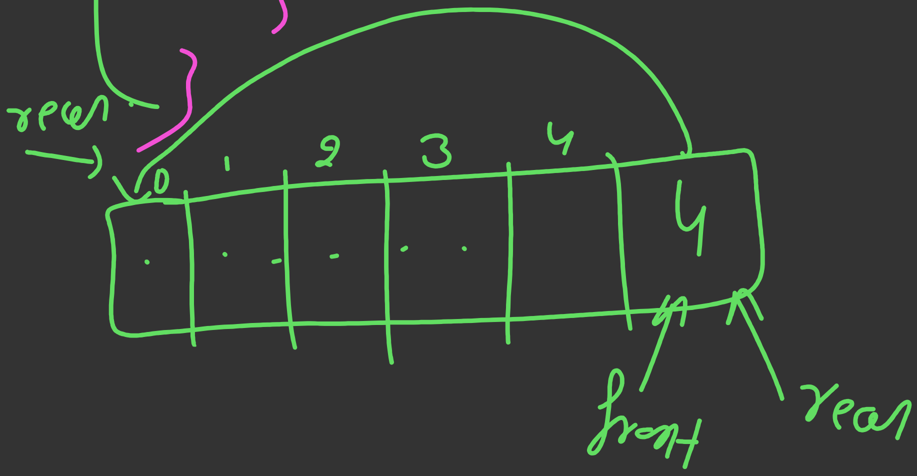


```
Dequeue( )
{
    if (front==-1 && rear==-1)
    {
        pf("underflow")

    }
    Else if (front == rear)
    {
        front=rear=-1;

    }
    Else
    {
        front++;
    }
}
```

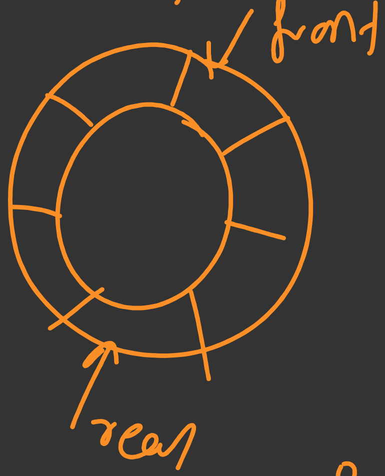
```
Traverse( )
{
    int i;
θ(n)
    for(i=front; i<=rear; i++)
    {
        Queue[i];
    }
}
```

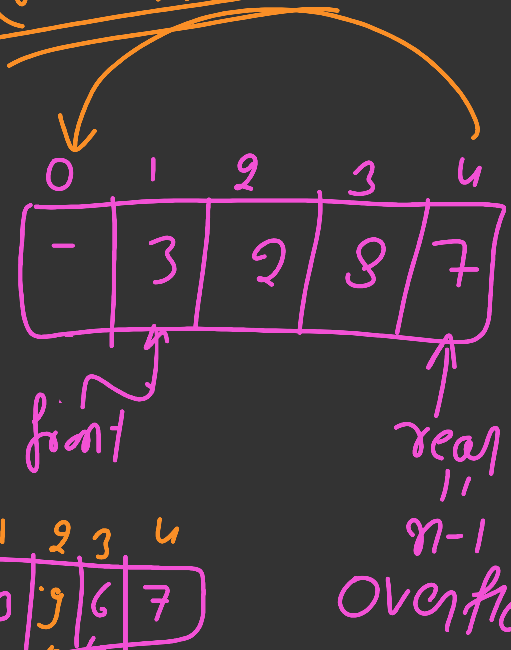


$rear = (rear+1) \% n$

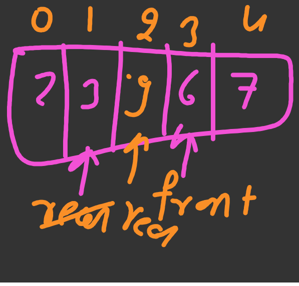
~~rear~~ =

$front == (rear+1) \% n$ ← Array is full





0	1	2	3	4
-	3	2	8	7

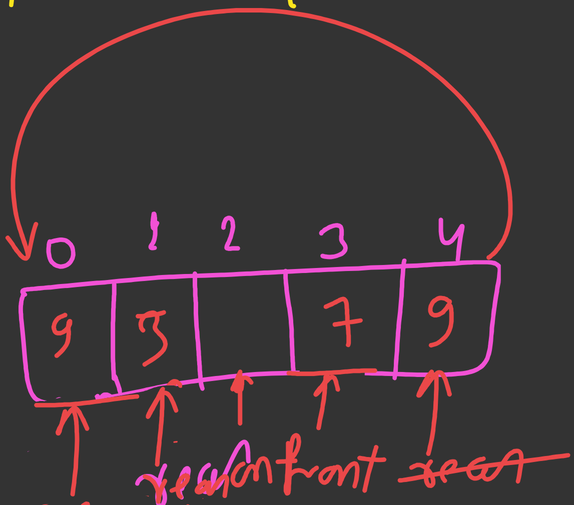


0	1	2	3	4
7	3	9	6	7

```
main ( )
{      Queue[5]
 Enqueue(5)
 Enqueue(3)
 Enqueue(2)
 Enqueue(8)
 Enqueue(7)
 Traverse ( )
 Dequeue( )
}
Enqueue(9)
```

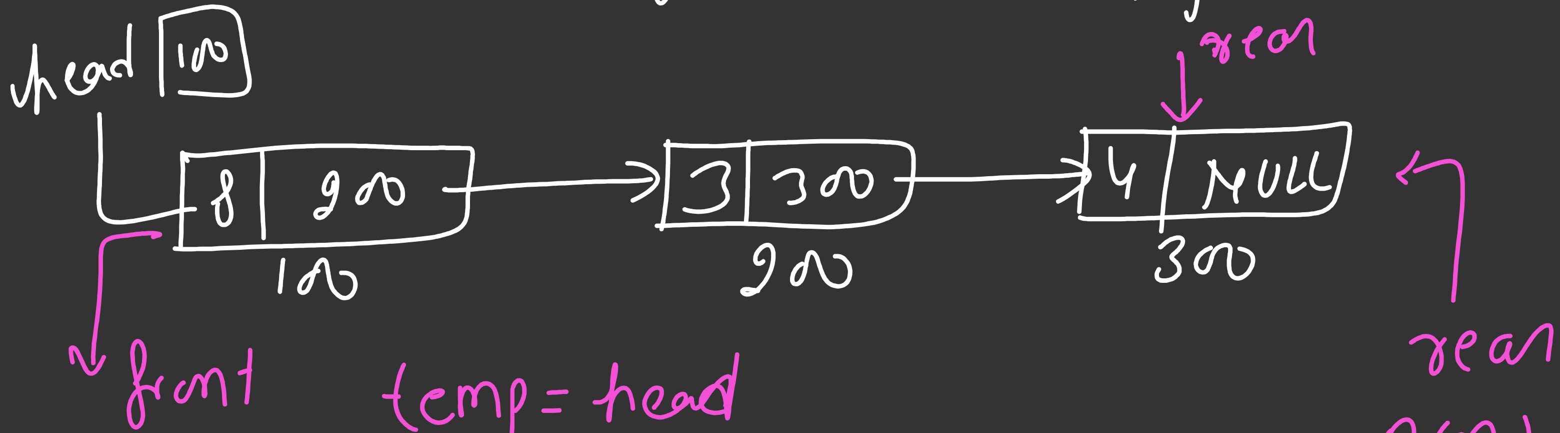
Assignment = Implement queue using Circular linked list.

```
Enqueue(int x)
{
    if(front==-1 && rear==-1)
    {
        front++; rear++;
    }
    else if( front==((rear+1)%n))
    {
        "overflow";
    }
    else
    {
          0        4          5
        rear=(rear+1)%n
        Queue[rear]=x;
    }
}
```



```
i= front
                  +1
while(i!=rear)
{
    pf(Queue[i];
}   i=(i+1)%n
```

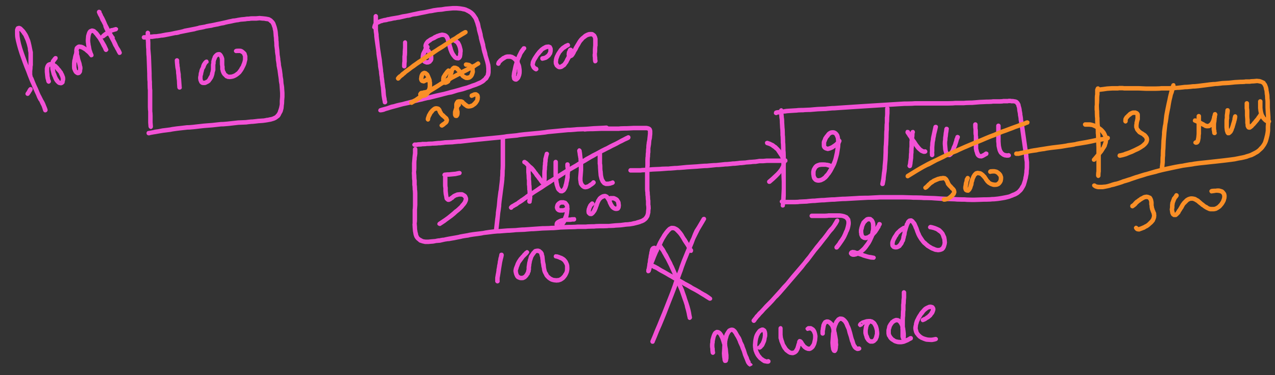
§ Implementation of queue using linked list:: (Dynamic Memory Allocation)



Dequeue O(1)

```
temp = head
~~te~~ head = head→next
free(temp)
```

O(n)



```
Struct node
{   int data;
    Struct node *next;

}
Struct node *front=NULL;
Struct node *rear=NULL;
Enqueue(int x)
{
    Struct node *newnode;
      newnode→data=x.
      newnode→next=NULL
  If(front==NULL && rear==NULL)
      front=rear=newnode.
```

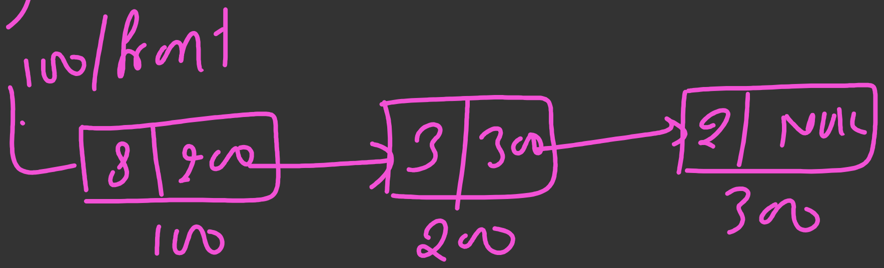
O(1)

```
main( )
{
   Enqueue(5)
   Enqueue(2)
   Enqueue(3)

}
```

rear→next=newnode

```
Traverse( )
{
      struct node *temp;
      if( front ==NUL && rear == NULL)
                  "Empty queue"
      else
      {
              100      100
              temp = front;
              while(temp~~->next~~ != NULL)
              {
                     pf(temp->data;
      }     }        temp=temp->next;
```



```
Dequeue( )
{
    if (front==NULL && rear==NULL)
            "underflow"

    Else
     {
         struct node *temp;
             temp=front;
             front=front→next;
               free(temp);
     }
}
```