Roll No. 23355

Rajdhani

DATE / /

Title : Assignment 3 : Circular Queue Linear data structure

Aim : To implement circular queue using Array as a linear data structure

Problem statement :
Implement circular queue using array as linear list.
Perform following operations on it :
a) Insertion (Enqueue)
b) Deletion (Dequeue)
c) Display

Objective :
- To understand the simple queue as a linear data structure with its limitation
- Understand & implement circular queue with array & perform various operations on it.
- understand & apply the queue full & queue empty condition
- know possible applications of queue.

Outcome :
- Able to overcome the simple queue limitations by implementing circular queue.
- Implement different operations like insert & delete on the circular queue
- Display contents of queue after every operation
- Able to implement real time application using queue.

Rajdhani
DATE / /

Theory

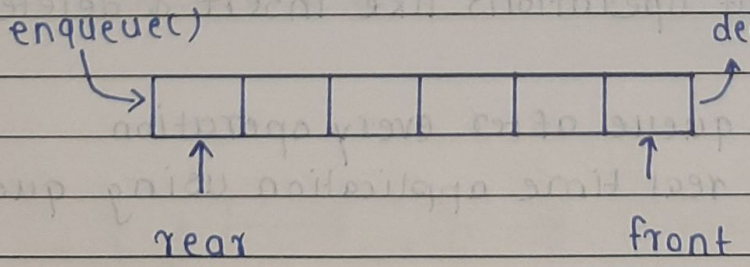
- Concept of queue as linear data structure:
- Queue is a linear data structure which follows a particular order in which operations are performed. The order is First In First Out (FIFO).
- It is a special kind of list, where items are inserted at one end & deleted from other end.

- Simple queue ADT (1D Array):

```
Struct Queue
{
    data[max];
    int front, rear;
};
```

Operations on queue:
Enqueue()
dequeue()
display()

- Graphical Representation of queue



- **Realization of ADT using array**

```
struct queue
{
    data[Max];
    int front, rear;
};
```

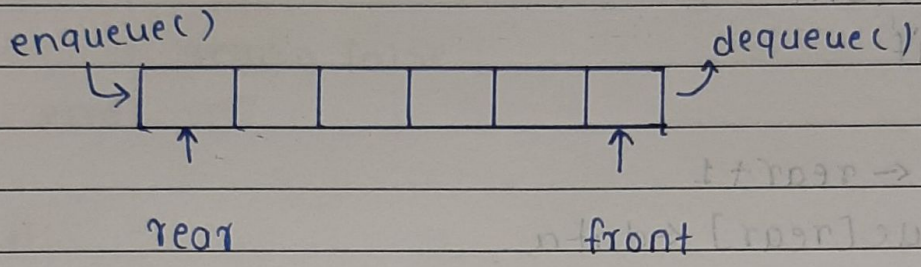


fig. Graphical representation of queue using Array

- **Realization of ADT using linked organization**

```
struct qNode
{
    data;
    qNode *next;
    qNode *front;
    qNode *rear;
};
```

```
Struct Queue
{
    qNode *front
    qNode *rear
};
```

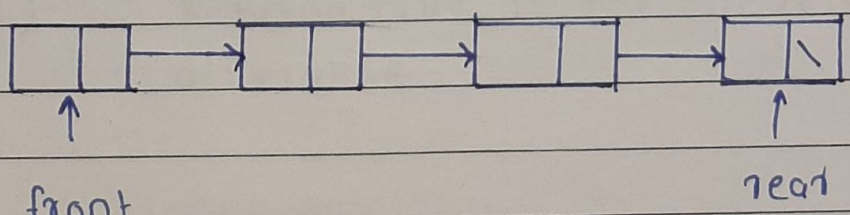


fig. Queue using linked organization

Rajdhani
DATE / /

- Array realization in detail with psuedocode of all operations:

1) Enqueue :

```
enqueue (data)
  if queue is full
     overflow condition
     return
  end if
  ~~rear~~
  rear ← rear + 1
  queue [rear] ← data
end
```

2) Dequeue :

```
dequeue ( )
  if queue is empty
     underflow condition
     return
  end if

  data ← queue [front]
  front ← front + 1
  return
end
```

Rajdhani
DATE / /

3) isEmpty :

```
isEmpty ()
  if front = -1 or front > rear
      Empty queue
      return true
  end if
  else
      return false
end
```

4) isfull :

```
isfull ()
  if rear = max_size
      Queue is full
      return true
  else
      return false
end
```

Working :

1) Enqueue () : Inserts an element at the rear end of queue
2) dequeue () : Deletes the front element.
3) isEmpty () : Returns true if the queue is empty, false otherwise
4) isfull () : Returns true if the queue is full, false otherwise.

Rajdhani
DATE / /

- Limitations of simple queue :
- In simple queue, if rear is present at maxsize i.e. at last index, though there may empty slots at beginning or middle of queue, queue will be reported as ~~queue~~ full.

- The solution is circular queue.

Circular Queue :
- Circular queue is a linear data structure in which the operations are performed based on First In First out (FIFO) principle & last position is connected back to the first position to make a circle.

Advantage of circular queue over simple queue :
- In circular queue, we utilize memory efficiently. Because in queue, when we ~~add~~ delete any element, only front is incremented by 1 but that position is not used later. So when we perform more add & delete operations, memory wastage increase
- But in circular queue, memory is utilized, if we delete any element that position is used later, because it is circular.

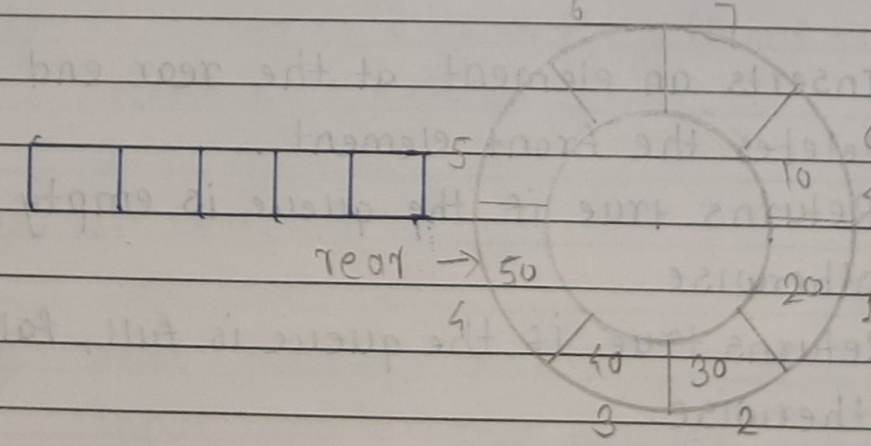


fig. circular queue

Rajdhani
DATE / /

- **circular queue possible implementation:**
- To give possible movement inside array, whenever we go past the last element, it should come back to the beginning of the array.

- Expression used for circular movement:

$$i = (i+1) \% max$$

where max is the size of queue.

- **Applications of Queue**

1) Job Scheduling
2) Round robin technique
3) keyboard buffer

Algorithms:

~~1) Insert:~~
~~1) If front = -1~~

1) Insert:
1) If (rear+1) % max + 1 = (front+1) % max
   Overflow condition
   Go to Step 4
2) If front = -1
   set front = 0
3) rear = (rear+1) % max
   Input data at rear position
4) Exit

Rajdhani
DATE / /

2) Delete :
1) If front = -1
underflow condition
Go to step 4
2) Set temp = queue [front]
3) If front = rear
set front = rear = -1
4) else
Front = (front + 1) % n
4) Exit

3) isEmpty :

If front = -1
queue is empty
return true
else
return false

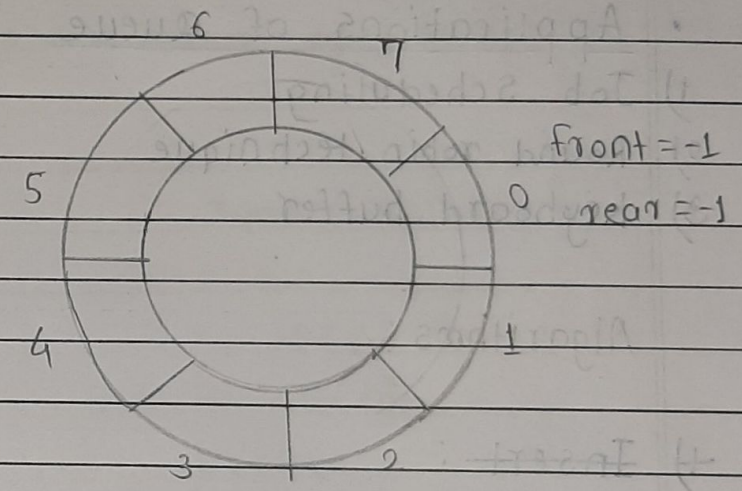


fig. empty queue condition

4) isFull :

if (front + 1) % max = (rear + 1) % max + 1
queue is full
return true
else
return false

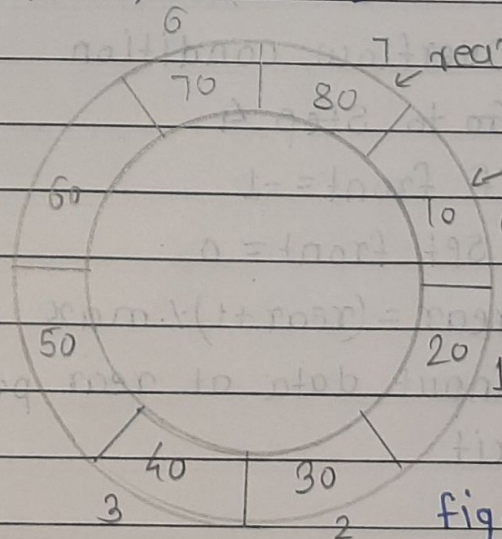


fig. queue full condition

Validation:

1) Array size should be in range 1 to maximum. It should not be negative, zero nor more than maximum size.
2) Patient id should not repeated.

Test cases:

Conclusion:

Analysis of insertion & deletion of operations in circular queue:

Operation	Time Complexity
1) enqueue()	O(1)
2) Dequeue()	O(1)