Rainbow
PAGE:
DATE: / /

Assignment No. 8

Name: Bhavana Bafna
Subject: DSA
Class: SE 9
Roll no.: 23107
Batch: E 9

Title: 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 limitations.
- Understand & Implement Circular Queue with Array and 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 applications using Queue.

Rainbow

PAGE:
DATE: / /

Theory:

1) Concept of Queue as a linear data structure:

Queue : It is an ordered collection of items from which items may be deleted at one end (called the front of queue) and into which items may be inserted at the other end (the rear of the queue)

It is special kind of list where items are inserted at one end and deleted from other end (fifo).

2) Simple Queue ADT (1D Array):

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

methods: for all queue, item ∈ element,
max_queue_size ∈ positive integer.

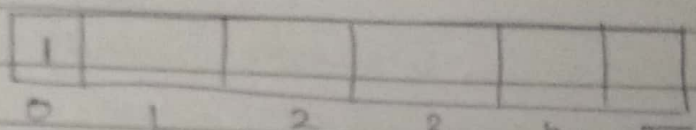
Queue createQ (max_queue_size) ::=
create an empty queue whose max size is
max_queue_size

Boolean isFullQ (queue, max_queue_size) ::=
if (no. of elements in queue == max_queue_size)
return TRUE
else return FALSE.

Queue Addq (queue, item) ::=
if (IsFullQ(queue)) queue_full
else insert item at rear of queue and return queue

Boolean IsEmptyQ(queue) ::=
    if (queue == CreateQ(max_queue_size))
        return TRUE
    else return False
Element delqueue(queue) ::=
    if (IsEmptyQ(queue)) return
    else remove and return the item at front of queue

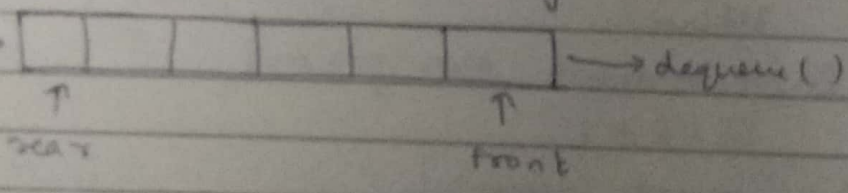
3) Graphical Representation of Queue:



Front = 0
Rear = 0

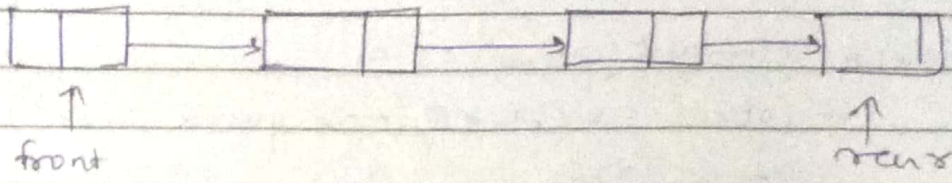
To insert: Put new element in location 0 & rear = 0, front = 0
To delete: Remove from 0th location & front = 1.

4) Realization of ADT using Array:



- initial front = rear = -1
- after 1st inserted front = rear = 0 (Increament)
- further insertion would increase rear by 1
- for deletion front increases by 1
- At last element both front & rear at same position.

5) Realization of ADT using C & Pseudocode:



- LL are linked organizations used to implement queue
  - Node * ~~next~~
  - Node * ~~front~~
  - Node * rear
- Head points of singly LL which is front (no rear required)
- Rear found using next element Null.

Operations:

1) Enqueue:
- Transverse to end node whose next is null & add node to list
- front = 0, rear = 0
- 1. [Overflow?]
  IF R > N
  then ('Overflow')
  return.
  2. [Increament rear pointer]
  R = R + 1
  3. [Insert element]
  Q [R] = item
  4. [Is front properly set?]
  If F = 0
  Then F = 1
  return.

2) Dequeue:

- initialize new node temp point to head & head → next.
- Delete temp.

PAGE:
DATE: / /

- 1. [Underflow ?]
  if F = 0
  then ("Underflow")
  return 0.
2. [Delete Element]
  item = Q[F]
3. [Is Queue Empty?]
  if F = R
  then F = R = 0
  else F = F + 1
4. [Return Element]
  return (item)

3) Create Q:

```
type def struct {
        int item [Max-size];
        int R, F;
    /* other fields */
        } ;
```

4) Empty Q:

```
int isEmpty ()
    if F = -1 or F > R
     Empty Queue
     return True
     endif
    else
        return False
    end
```

Rainbow
PAGE:
DATE: / /

5] Q Full:

```
int isFull()
    if R = max_size
        Q is Full
        return True
    else
        return False
    end.
```

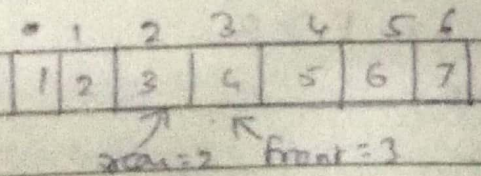
6) Limitation of simple Queue & possible solutions:

In Simple Queue:

- When new item inserted at rear, pointer to rear moves upwards & if item is deleted front also moves downwards.
- After a few insert & delete operations the rear might reach the the end of the queue & no more items can be inserted although items from from are deleted & there is space in the queue.

Solutions:

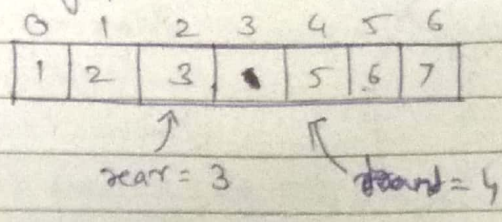
a) Use additional variable 'count':



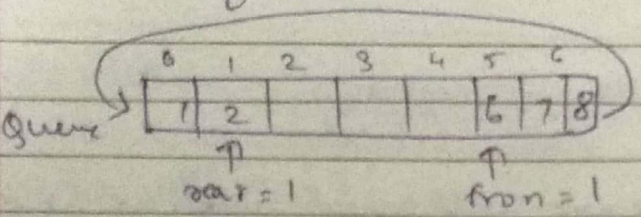
Count = 7.

IF value of count = size-max, queue = full

b) Keep a gap between element



c) Circular queue:



front = (front + 1) % length;
rear = (rear + 1) % length;

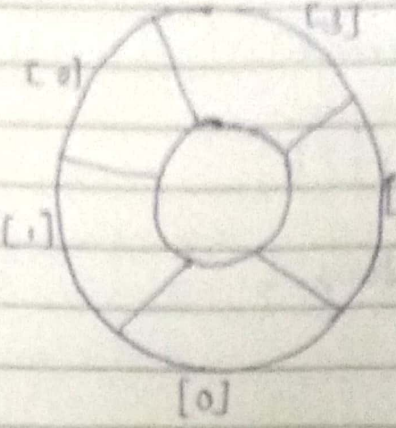
7) Circular queue & its advantage over simple queue:

Circular queue is a linear data structure in which the operations are performed base on FIFO principle & last position is connected back to first position to make a circle.

Advantage:

In circular queue we utilize memory efficiently. Because in queue, when we delete any element, only front is incremented by 1, but that position is not used later So, coben we perform more insertion & deletion operations memory storage increase. But in circular queue, memory is utilized, if we delete any element, that position is used later because it is circular.

Rainbow
PAGE:
DATE: / /



Front = 0
Rear = 0

(Empty Circular Queue)

8) Circular queue possible implementation:

- To give possible movement inside array, when we go past the last element, it should come back to beginning of the array

- Expression: $i = (i+1) \% max\_size$

front = (front + 1) % length
rear = (rear + 1) % length.

9) Application of Queue:

- Real life examples:
  - Waiting in line
  - Waiting on hold for tech support
- Applications in Computer science:
  - Typical uses of queues are in simulations & OS.
  - In OS, for controlling access to shared system resources.
  - OS often maintain queues of processes that are ready to execute or that are waiting for particular event to occur
  - Computer systems must often provide a "holding area" for messages between two process i/e "buffer" & it is implemented as queue.
  - Job scheduling.

Algorithms/Pseudocode:

1) Insert:

1. if (rear + 1) % max = Front
   overflow condition
   Jump to step 3

2. else
   rear = (rear + 1) % max
   Q[rear] = item
   if front == -1
   set front = 0

3. Exit

2) Delete:

1. if front == -1
   underflow condition

2. else
   temp = Q[Front]
   front = (front + 1) % max
   if front == (rear + 1) % max
   set front = -1
   rear = -1

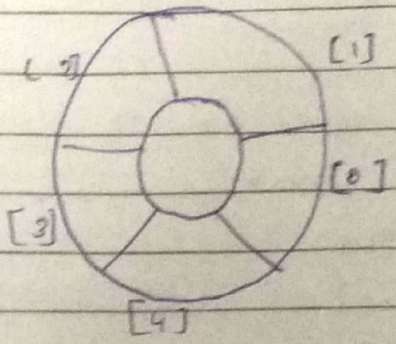
3. Exit

Rainbow

PAGE:
DATE: / /

3) Q is empty:

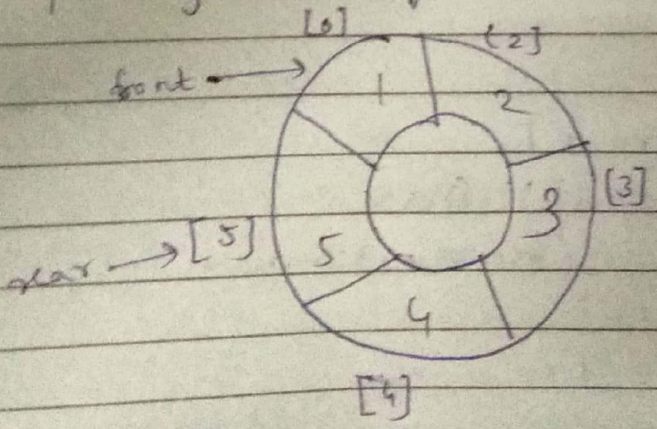
if front == -1
    print queue is empty
else
    print queue not empty.



Front = -1
Rear = -1

4) Q is Full:

if (rear + 1) % max == front
    print Queue full
else
    print Queue not full



Validations:

- Array size should be in range 1 to max, not negative, zero or more than max size.
- Name must not contain numbers.
- Age should be positive.

Conclusion:

Analaysis of insertion & deletion of operation in circular queue.

Operation	Time complexity
1) Enqueue	O(1)
2) Dequeue	O(1)