TECHNO INDIA UNIVERSITY
WESTBENGAL

Data Structures & Algorithms
(PCC-CS 301)

Dr. Debashis Das
Associate Professor
Department of CSE
Techno India University, Kolkata

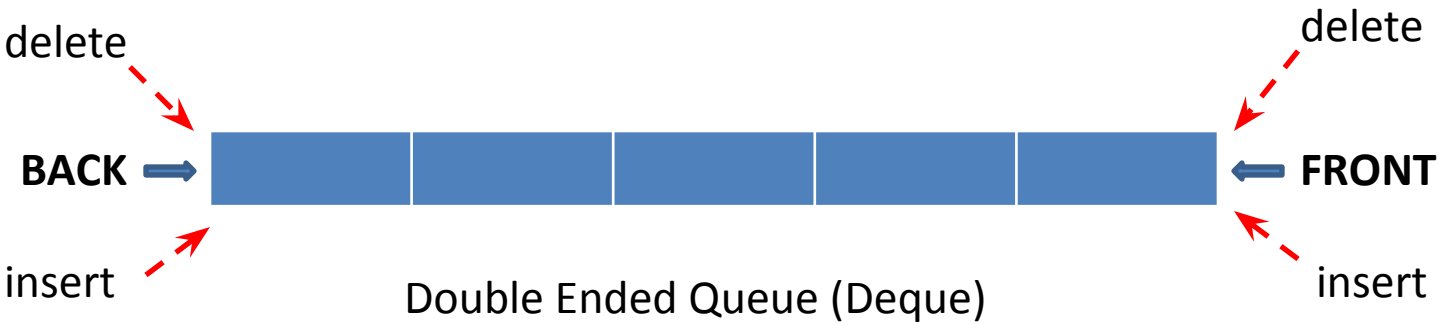
Topics Covered

1. Linear Data Structure
    a. Double Ended Queue (Deque)
    b. Stack using Deque
    c. Queue using Deque

Double Ended Queue

Deck of playing cards

A card can be inserted or taken from the top as well as from the bottom of the deck



Double Ended Queue (Deque)

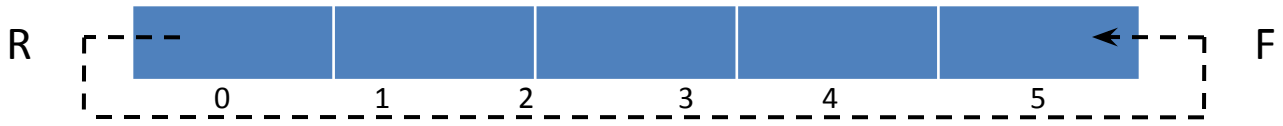
[pronounced as Deck]

Queue

- Double Ended Queue (Deque)
  - Properties
    - Deque is a generalized version of Queue data structures
    - An element can be inserted from both end of the queue
    - An element can be deleted from both end of the queue
    - It does not specifically maintain FIFO concept
    - The concept of Stack and Queue both can be represented through this single Deque data structure
      - Stack can be represented by restricting data insertion and deletion at back end only
      - Queue can be represented by restricting data insertion at back end only and deletion from front end only

Double Ended Queue

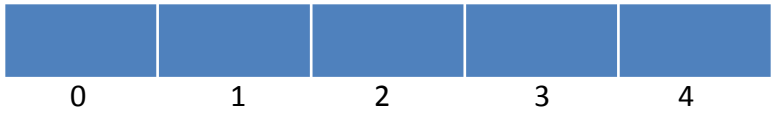
- Deque
  - Representation
    - Using array
    - Using linked-list (to be discussed later)
  - Array representation
    - Using circular array



Double Ended Queue

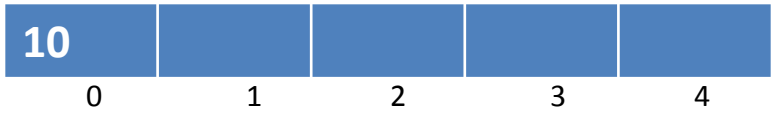
- Double Ended Queue
  - Different cases

F=R=null



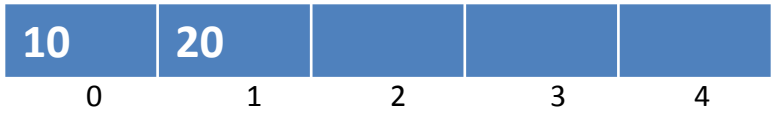
Empty Queue

F=R=0



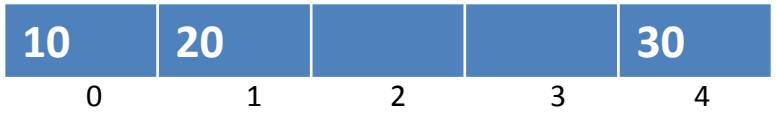
Insert 1st element

F=0 , R=1



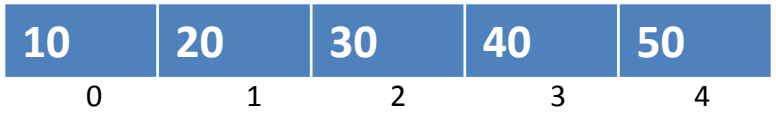
Insert at back

F=4 , R=1



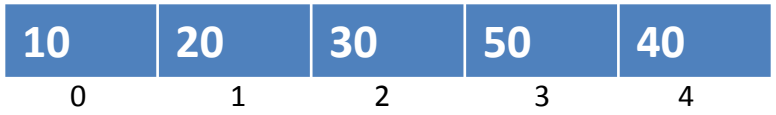
Insert at front

F=0 , R=4



Queue Overflow

F=3 , R=2



Queue Overflow

Double Ended Queue

- Double Ended Queue
  - Operations
    - ENQUEUE (data insertion into queue)
      - Insert at Front
      - Insert at Back
    - DEQUEUE (data deletion from queue)
      - Delete from Front
      - Delete from Back

    Primary operation

    - Display (showing element of queue)
    - QueueSize (returns the total element)
    - IsFullQueue (checks if Queue is overflow)
    - IsEmptyQueue (checks if Queue is underflow)

    Auxiliary operation

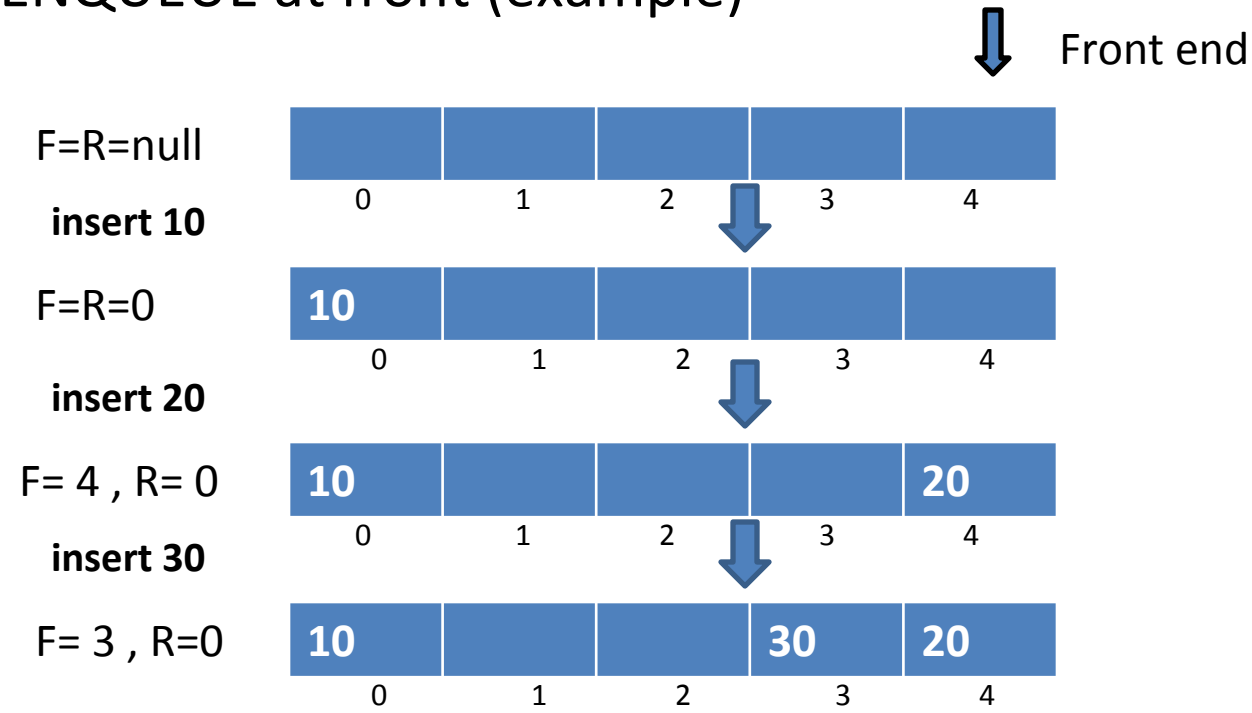
Double Ended Queue

- Operation
  - ENQUEUE
    - Insert element at front

```
void ENQUEUE(data)
{
 if  IsFullQueue = TRUE
    print  Q is full
 else
    if   IsEmptyQueue = TRUE
        F := 0 and R:= 0
    else
       if F = 0
         F:= Max_Size
       else
         F:=F-1
    Q(F) := data
}
```

Double Ended Queue

- Operation
  - ENQUEUE at front (example)



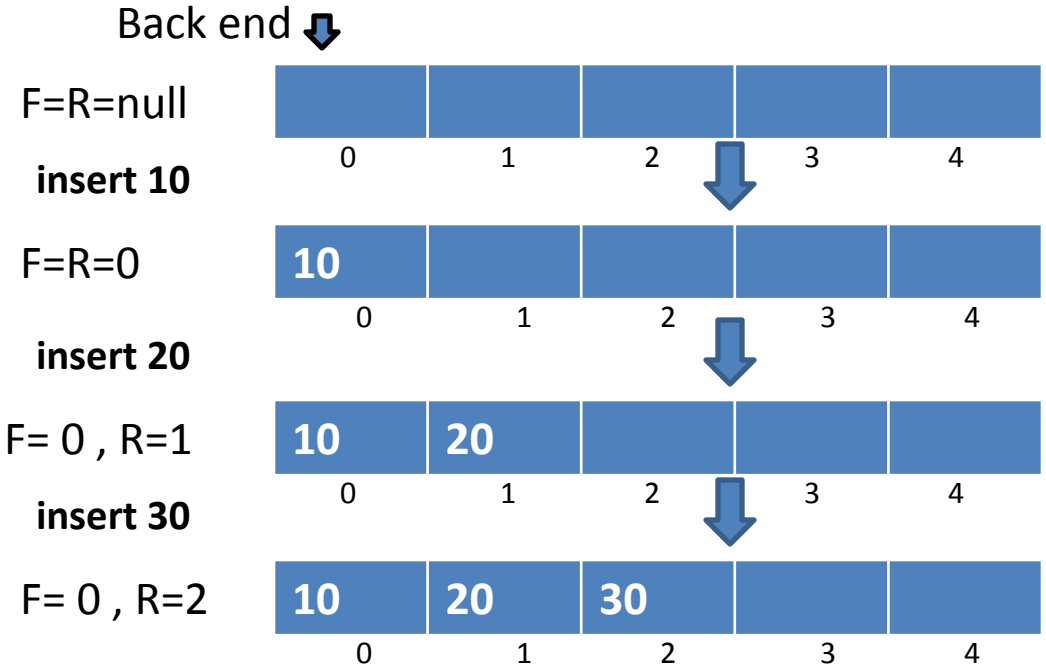
Double Ended Queue

- Operation
  - ENQUEUE
    - Insert element at back

```
void ENQUEUE(data)
{
 if  IsFullQueue = TRUE
    print  Q is full
 else
    if   IsEmptyQueue = TRUE
        F := 0 and R:= 0
    else
       R:= R+1
    Q(R) := data
}
```

Double Ended Queue

- Operation
  - ENQUEUE at back (example)



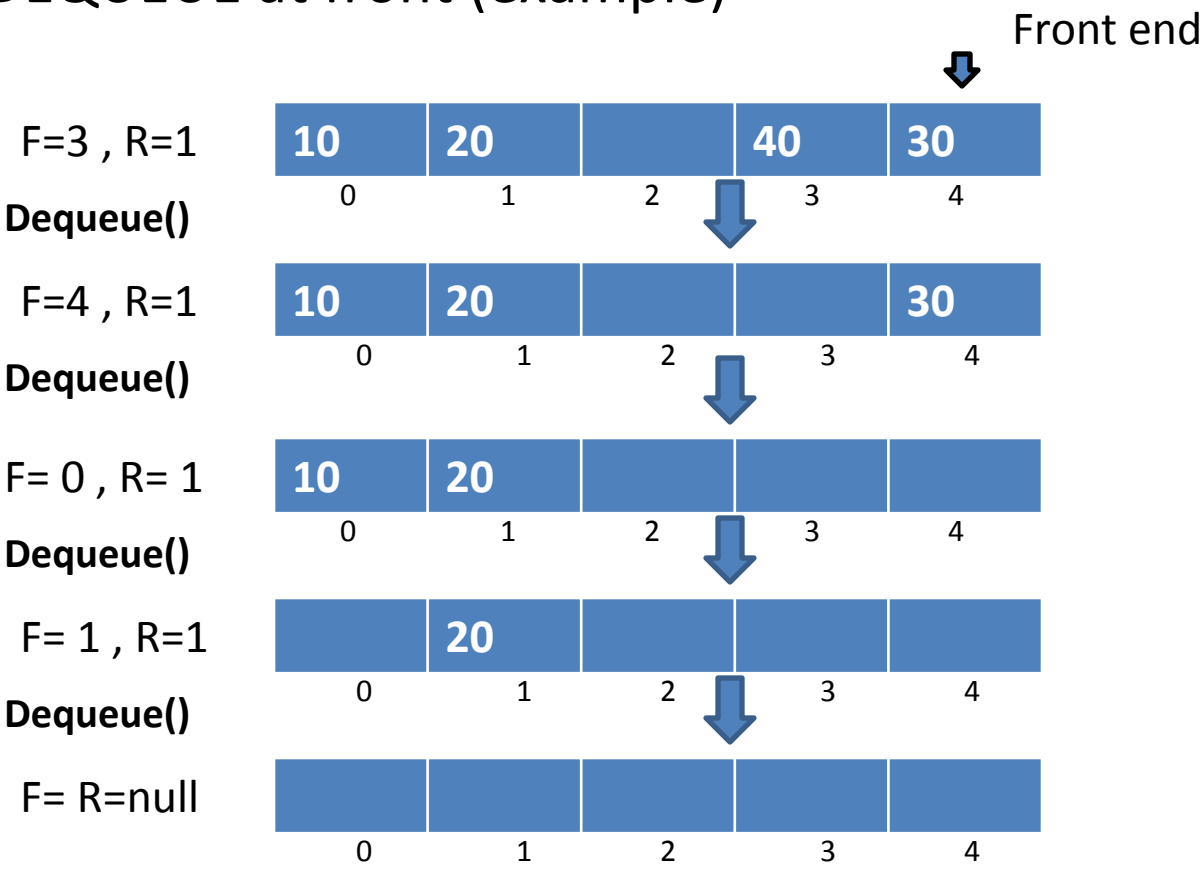
Double Ended Queue

- Operation
  - DEQUEUE
    - Delete element from front

```
int DEQUEUE()
{
 if  IsEmptyQueue = TRUE
   return NULL
 else
    data := Q(F)
    if   F = R
      F := null and R:= null
    else
       if  F = Max_Size
          F := 0
       else
          F := F+1
   return data
}
```

Double Ended Queue

- Operation
  - DEQUEUE at front (example)



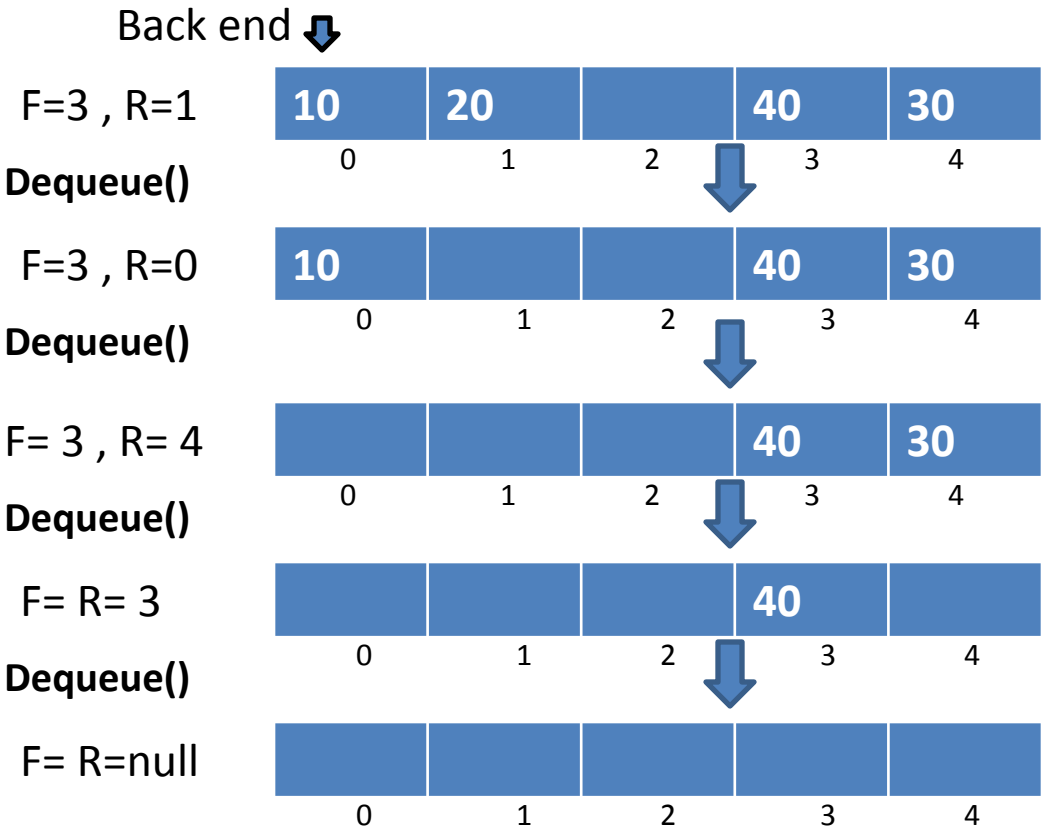
Double Ended Queue

- Operation
  - DEQUEUE
    - Delete element from back

```
int DEQUEUE()
{
 if  IsEmptyQueue = TRUE
   return NULL
 else
    data := Q(R)
    if   F = R
      F := null and R:= null
    else
        if  R = 0
           R := Max_Size
        else
           R := R - 1
   return data
}
```

Double Ended Queue

- Operation
  - DEQUEUE at back (example)



Double Ended Queue

- Operation
  - Display
    - The queue can be displayed through an auxiliary pointer without shifting FRONT or REAR

```
void Display()
{
  if  IsEmptyQueue = TRUE
    print  Q is empty
  else
     for i= F to R
        print  Q(i)
}
```

Double Ended Queue

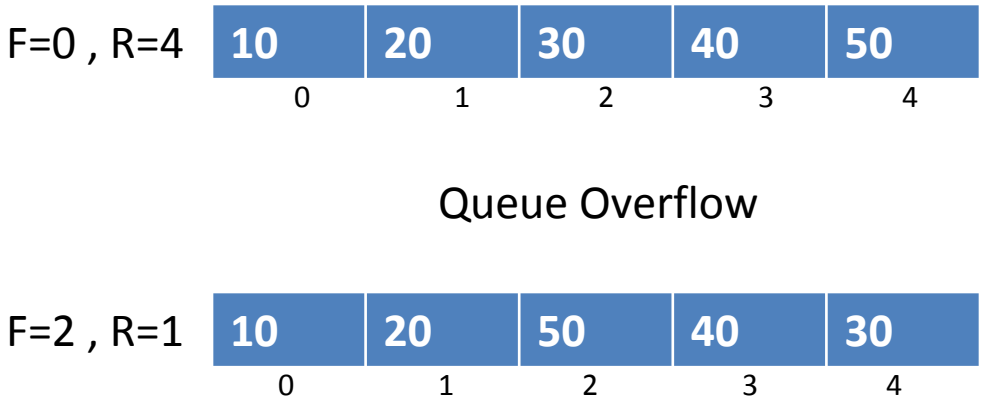
- Operation
  - QueueSize
    - This function returns the counting of elements present in the current queue

```
int QueueSize()
{
  if  F = null and R = NULL
     return 0
  else
     for i = F to R
        count := count +1
     return count
}
```

Double Ended Queue

- Operation
  - IsFullQueue
    - This function checks whether the Queue is full or not
    - We cannot insert data into Queue if it is full

```
Boolean IsFullQueue()
{
 if  (F=0 and R = Max_Size) or
      (F = R+1)
   return TRUE
 else
   return FALSE
}
```



Double Ended Queue

- Operation
  - IsEmptyQueue
    - This function checks whether the Queue is empty or not
    - We cannot delete or display the Queue if it is empty

```
Boolean IsEmptyQueue()
{
  if  F = null and R = null
    return TRUE
  else
    return FALSE
}
```

Double Ended Queue

- Operation: complexity

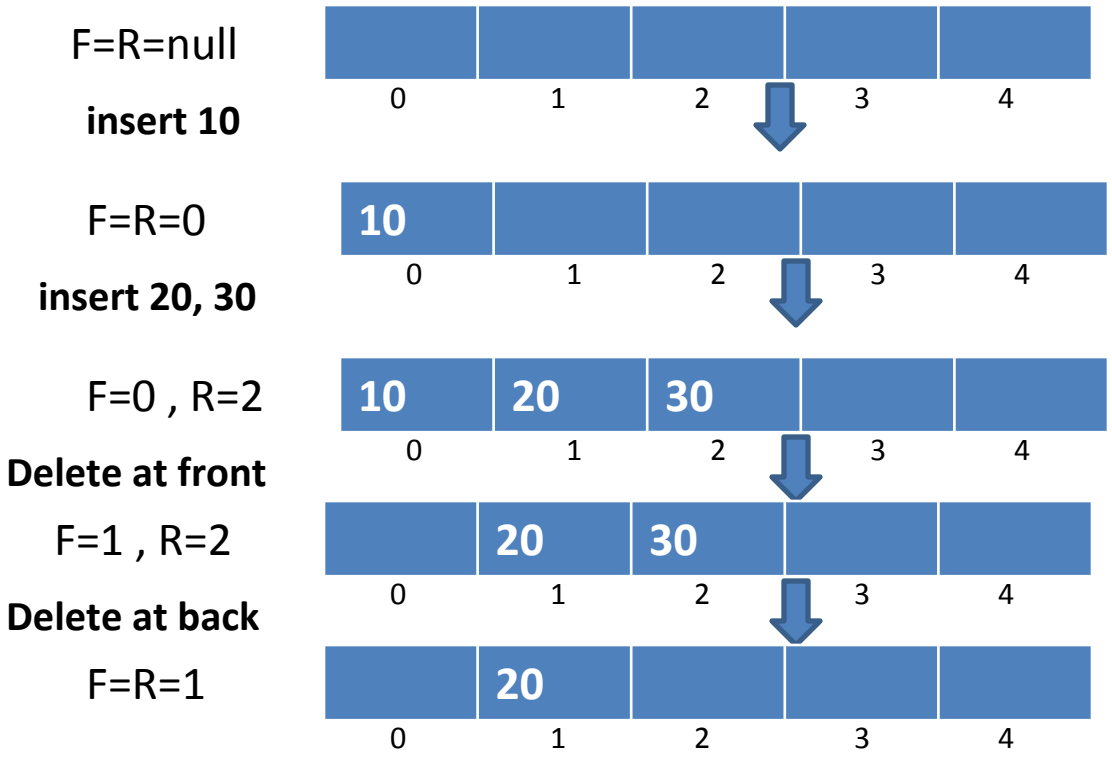
Operation	Time Complexity
Enqueue()	O(1)
DeQueue()	O(1)
Display()	O(n)
QueueSize()	O(n)
IsFullQueue()	O(1)
IsEmptyQueue()	O(1)

Double Ended Queue

- Variants
  - Input Restricted Deque
    - Data can be inserted at one end but deletion can be made from both the ends
      - Insert data at front or
      - Insert data at back
  - Output Restricted Deque
    - Data can be inserted at both ends but deletion ca be made from one end only
      - Delete data from front or
      - Delete data from back

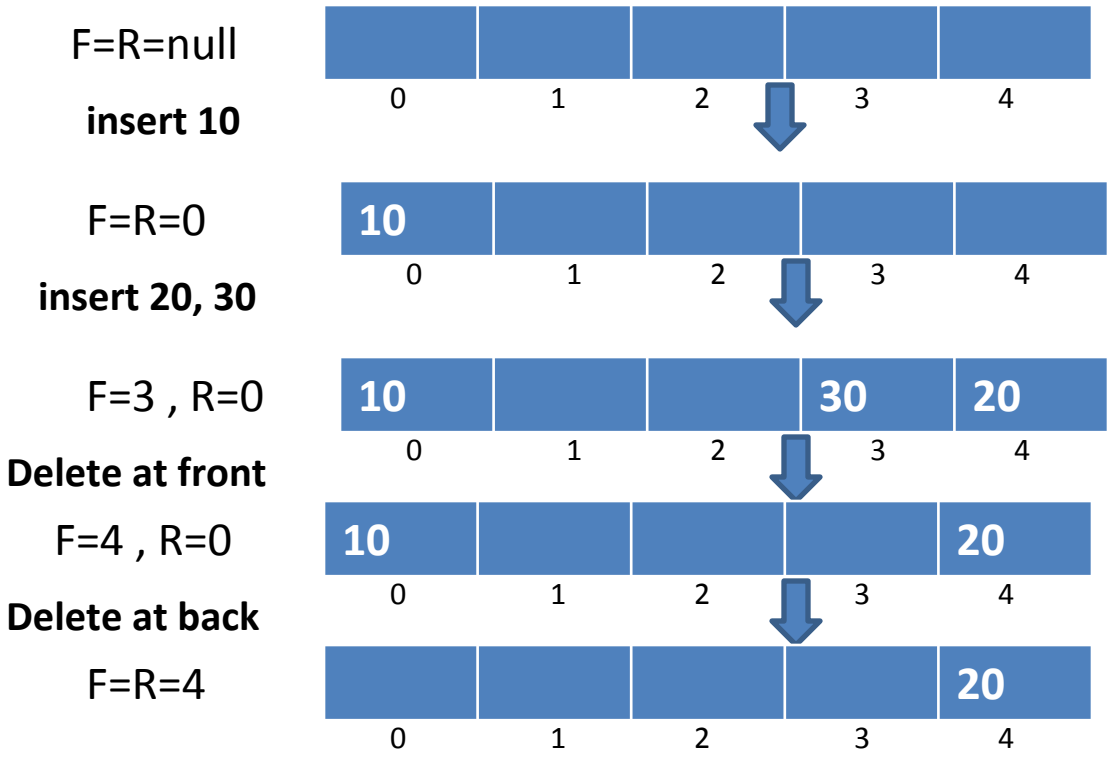
Double Ended Queue

- Input restricted Deque
  - Input restricted at **front** end



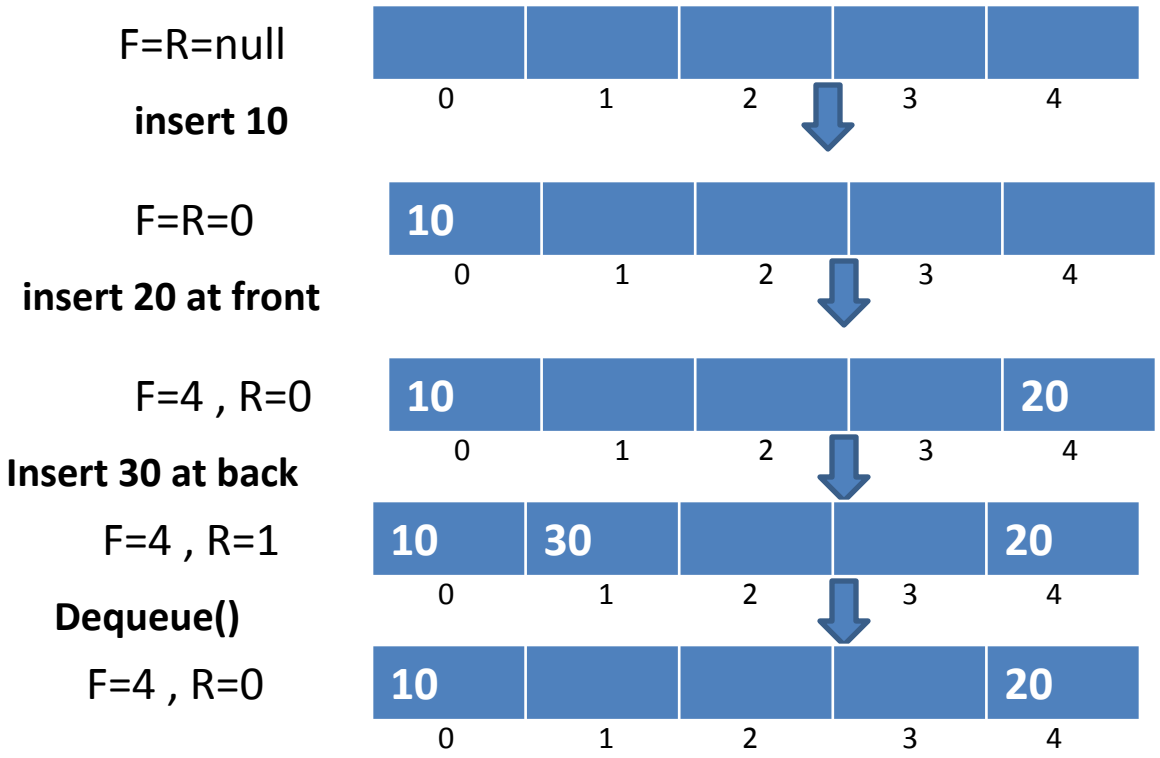
Double Ended Queue

- Input restricted Deque
  - Input restricted at **back** end



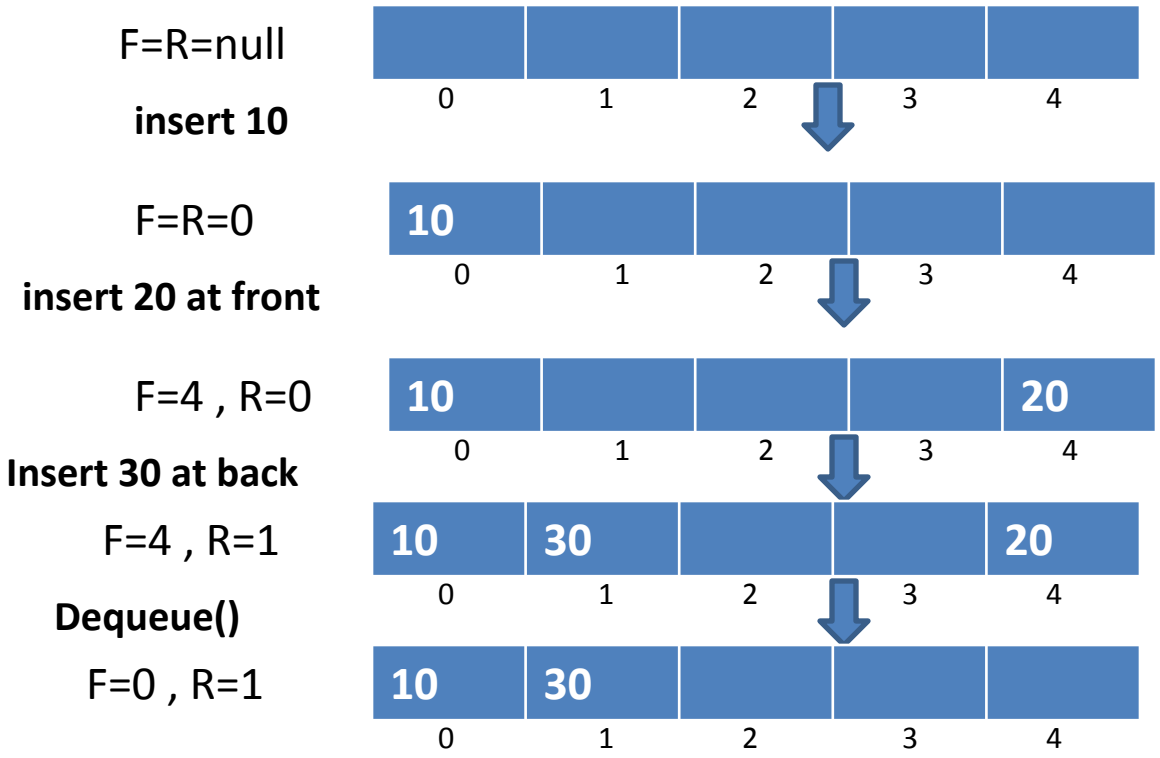
Double Ended Queue

- Output restricted Deque
  - Output restricted at **front** end



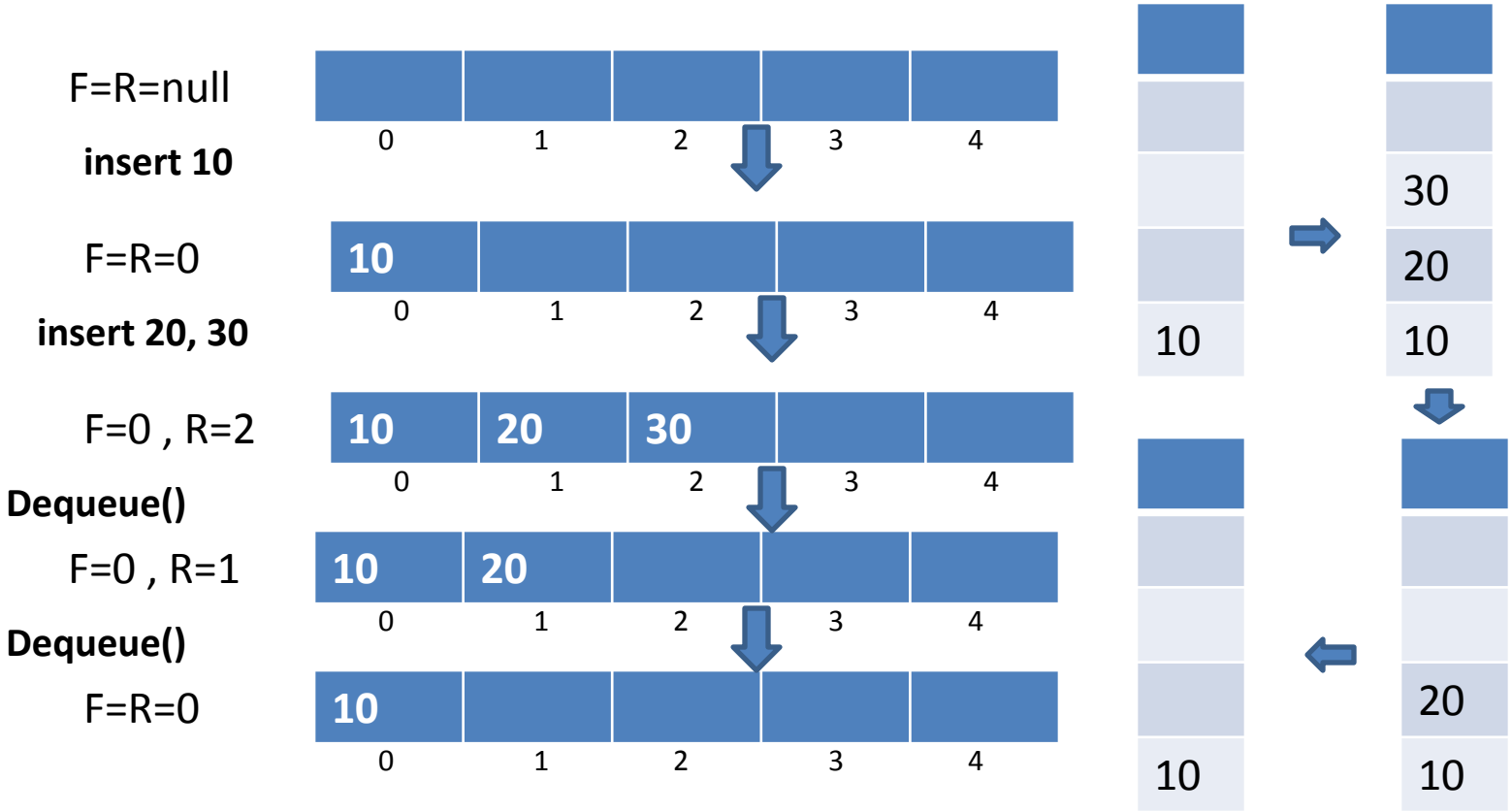
Double Ended Queue

- Output restricted Deque
  - Output restricted at **back** end



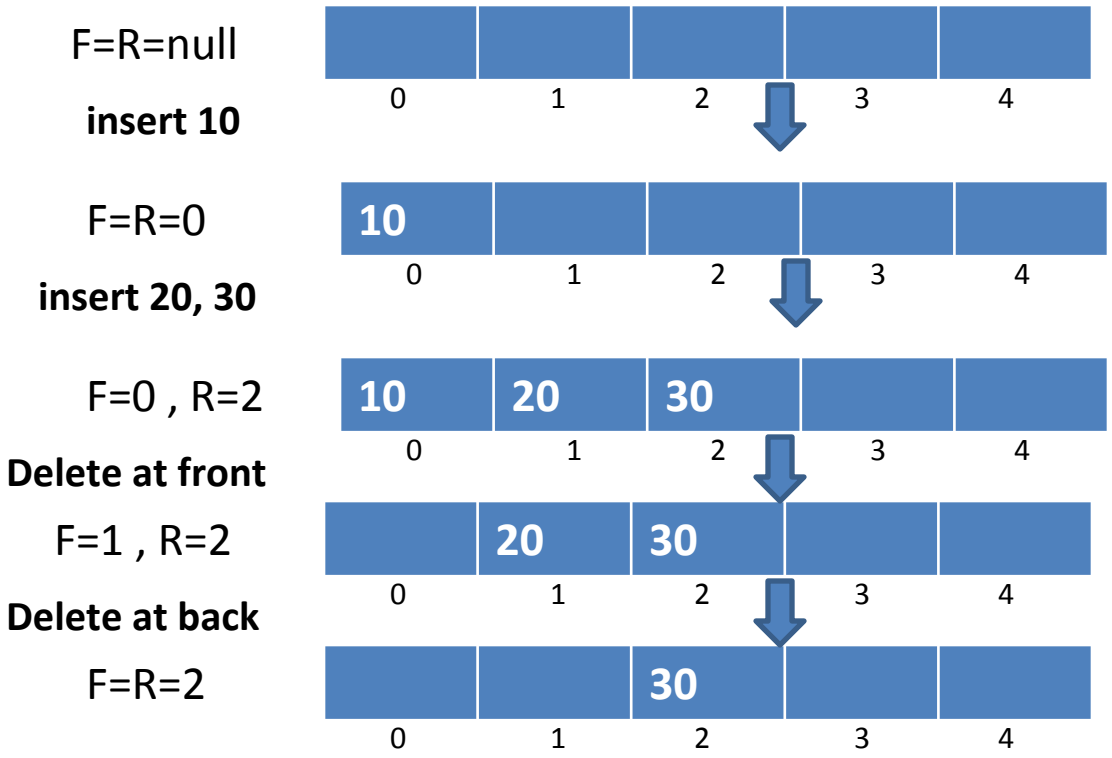
Double Ended Queue

- Deque as Stack
  - Insertion at **back** and deletion **from** back



Double Ended Queue

- Deque as Queue (simple queue)
  - Insertion at **back** and deletion from **front**



Queries?