Implementation of Deque using circular array

Deque or Double Ended Queue is a generalized version of Queue data structure that allows insert and delete at both ends.In previous post we had discussed introduction of deque. Now in this post we see how we implement deque Using circular array.

Operations on Deque:

Mainly the following four basic operations are performed on queue:

insetFront(): Adds an item at the front of Deque.

insertRear(): Adds an item at the rear of Deque.

deleteFront(): Deletes an item from front of Deque.

deleteRear(): Deletes an item from rear of Deque.

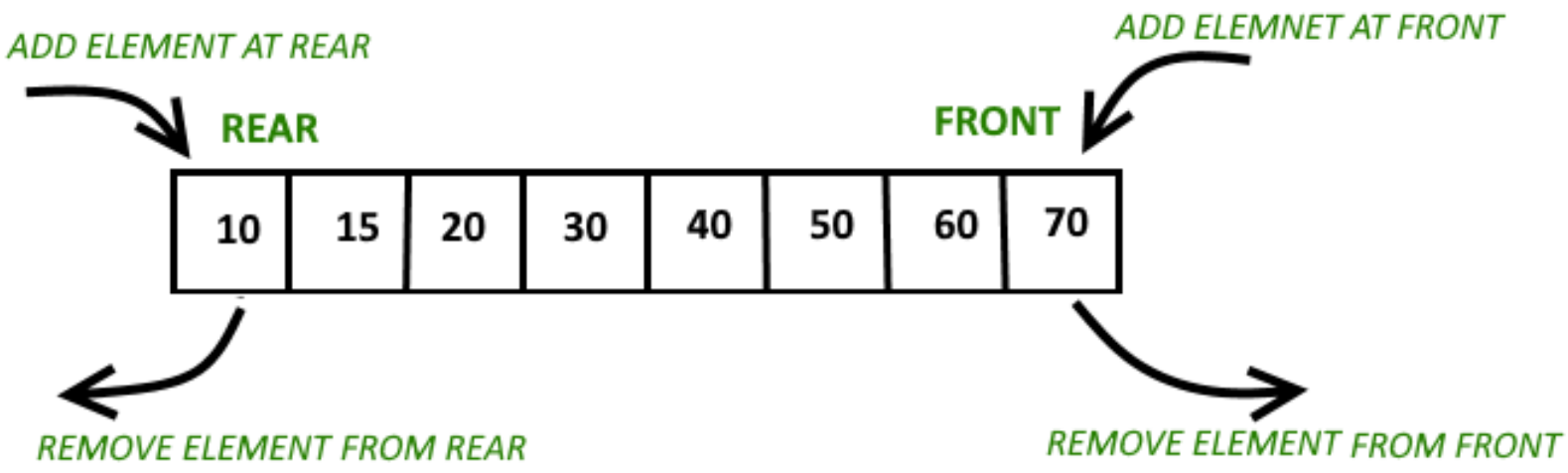
In addition to above operations, following operations are also supported

getFront(): Gets the front item from queue.

getRear(): Gets the last item from queue.

isEmpty(): Checks whether Deque is empty or not.

isFull(): Checks whether Deque is full or not.



Recommended: Please try your approach on *{IDE}* first, before moving on to the solution.

Circular array implementation deque

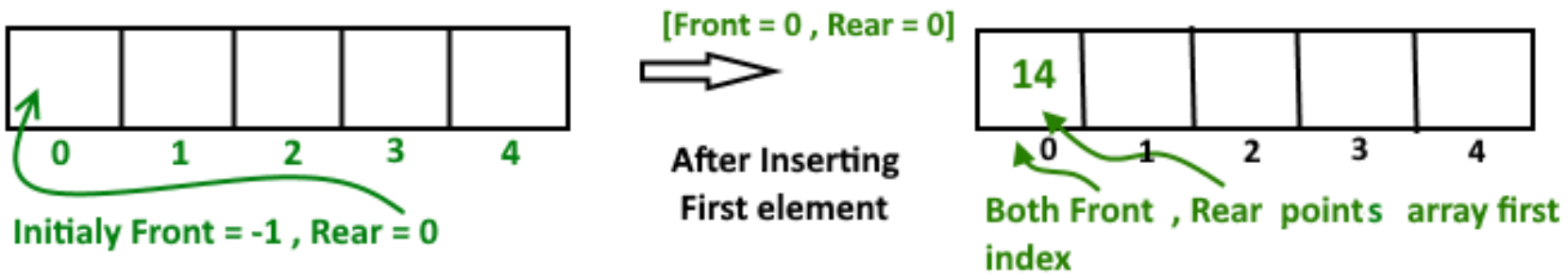
For implementing deque, we need to keep track of two indices, front and rear. We enqueue(push) an item at the rear or the front end of qedue and dequeue(pop) an item from both rear and front end.

Working

1. Create an empty array ‘arr’ of size ‘n’

initialize **front = -1** , **rear = 0**

Inserting First element in deque either front end or read end they both lead to the same result.



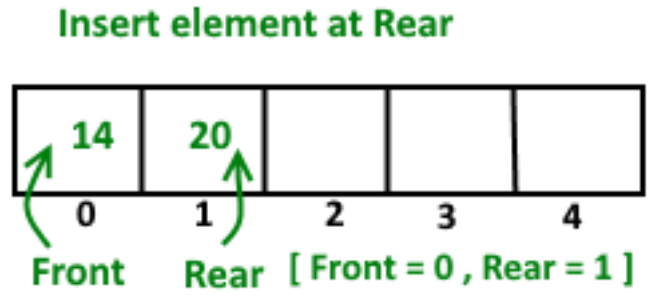
After insert **Front** Points = 0 and **Rear** points = 0

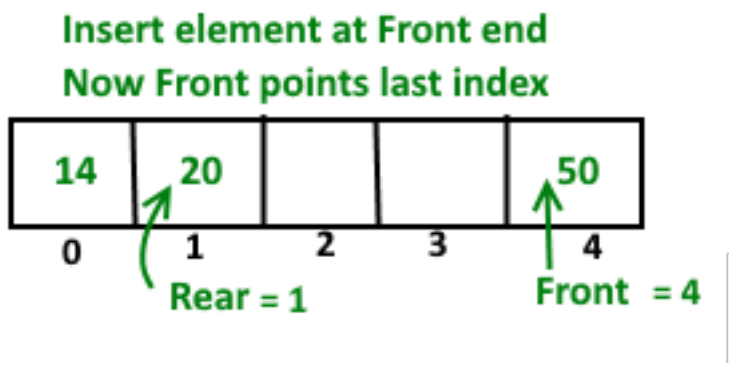
Insert Elements at Rear end

```
a). First we check deque if Full or Not
b). IF Rear == Size-1
       then reinitialize Rear = 0 ;
    Else increment Rear by '1'
    and push current key into Arr[ rear ] = key
Front remain same.
```

Insert Elements at Front end

```
a). First we check deque if Full or Not
b). IF Front == 0 || initial position, move Front
                     to points last index of array
       front = size - 1
    Else decremented front by '1' and push
         current key into Arr[ Front] = key
Rear remain same.
```



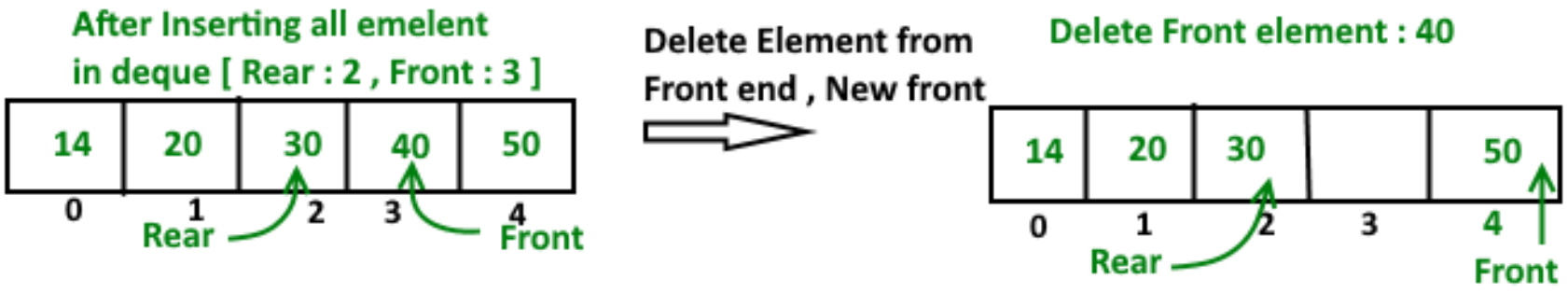


Delete Element From Rear end

```
a). first Check deque is Empty or Not
b).  If deque has only one element
        front = -1 ; rear =-1 ;
    Else IF Rear points to the first index of array
         it's means we have to move rear to points
         last index [ now first inserted element at
         front end become rear end ]
            rear = size-1 ;
    Else || decrease rear by '1'
            rear = rear-1;
```

Delete Element From Front end

```
a). first Check deque is Empty or Not
b).  If deque has only one element
            front = -1 ; rear =-1 ;
    Else IF front points to the last index of the array
         it's means we have no more elements in array so
          we move front to points first index of array
            front = 0 ;
    Else || increment Front by '1'
            front = front+1;
```



Below c++ implementation of above idea.

```
// C++ implementation of De-queue using circular
// array
#include<iostream>
using namespace std;

// Maximum size of array or Dequeue
#define MAX 100

// A structure to represent a Deque
class Deque
{
    int  arr[MAX];
    int  front;
    int  rear;
    int  size;
public :
    Deque(int size)
    {
        front = -1;
        rear = 0;
        this->size = size;
    }

    // Operations on Deque:
    void  insertfront(int key);
    void  insertrear(int key);
    void  deletefront();
    void  deleterear();
    bool  isFull();
    bool  isEmpty();
    int  getFront();
    int  getRear();
};

// Checks whether Deque is full or not.
bool Deque::isFull()
{
    return ((front == 0 && rear == size-1)||
            front == rear+1);
}

// Checks whether Deque is empty or not.
bool Deque::isEmpty ()
{
    return (front == -1);
}
```

```
// Inserts an element at front
void Deque::insertfront(int key)
{
    // check whether Deque if  full or not
    if (isFull())
    {
        cout << "Overflow\n" << endl;
        return;
    }

    // If queue is initially empty
    if (front == -1)
    {
        front = 0;
        rear = 0;
    }

    // front is at first position of queue
    else if (front == 0)
        front = size - 1 ;

    else // decrement front end by '1'
        front = front-1;

    // insert current element into Deque
    arr[front] = key ;
}

// function to inset element at rear end
// of Deque.
void Deque ::insertrear(int key)
{
    if (isFull())
    {
        cout << " Overflow\n " << endl;
        return;
    }

    // If queue is initially empty
    if (front == -1)
    {
        front = 0;
        rear = 0;
    }

    // rear is at last position of queue
    else if (rear == size-1)
        rear = 0;

    // increment rear end by '1'
    else
        rear = rear+1;

    // insert current element into Deque
    arr[rear] = key ;
}

// Deletes element at front end of Deque
void Deque ::deletefront()
{
    // check whether Deque is empty or not
    if (isEmpty())
    {
        cout << "Queue Underflow\n" << endl;
```

```
        return ;
    }

    // Deque has only one element
    if (front == rear)
    {
        front = -1;
        rear = -1;
    }
    else
        // back to initial position
        if (front == size -1)
            front = 0;

        else // increment front by '1' to remove current
            // front value from Deque
            front = front+1;
}

// Delete element at rear end of Deque
void Deque::deleterear()
{
    if (isEmpty())
    {
        cout << " Underflow\n" << endl ;
        return ;
    }

    // Deque has only one element
    if (front == rear)
    {
        front = -1;
        rear = -1;
    }
    else if (rear == 0)
        rear = size-1;
    else
        rear = rear-1;
}

// Returns front element of Deque
int Deque::getFront()
{
    // check whether Deque is empty or not
    if (isEmpty())
    {
        cout << " Underflow\n" << endl;
        return -1 ;
    }
    return arr[front];
}

// function return rear element of Deque
int Deque::getRear()
{
    // check whether Deque is empty or not
    if(isEmpty() || rear < 0)
    {
        cout << " Underflow\n" << endl;
        return -1 ;
    }
    return arr[rear];
}

// Driver program to test above function
```

```
int main()
{
    Deque dq(5);
    cout << "Insert element at rear end  : 5 \n";
    dq.insertrear(5);

    cout << "insert element at rear end : 10 \n";
    dq.insertrear(10);

    cout << "get rear element " << " "
         << dq.getRear() << endl;

    dq.deleterear();
    cout << "After delete rear element new rear"
         << " become " << dq.getRear() << endl;

    cout << "insert element at front end \n";
    dq.insertfront(15);

    cout << "get front element " << " "
         << dq.getFront() << endl;

    dq.deletefront();

    cout << "After delete front element new "
       << "front become " << dq.getFront() << endl;
    return 0;
}
```

Run on IDE

Output:

```
insert element at rear end  : 5
insert element at rear end : 10
get rear element : 10
After delete rear element new rear become : 5
insert element at front end : 15
get front element : 15
After delete front element new front become : 5
```

Time Complexity: Time complexity of all operations like insertfront(), insertlast(), deletefront(), delete-last()is O(1).

In mext post we will discuss deque implementation using Doubly linked list.

This article is contributed by **Nishant Singh**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to con-tribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Queue

Recommended Posts:

Implement Queue using Stacks
Find the first circular tour that visits all petrol pumps
Distance of nearest cell having 1 in a binary matrix
Applications of Queue Data Structure
Sliding Window Maximum (Maximum of all subarrays of size k)