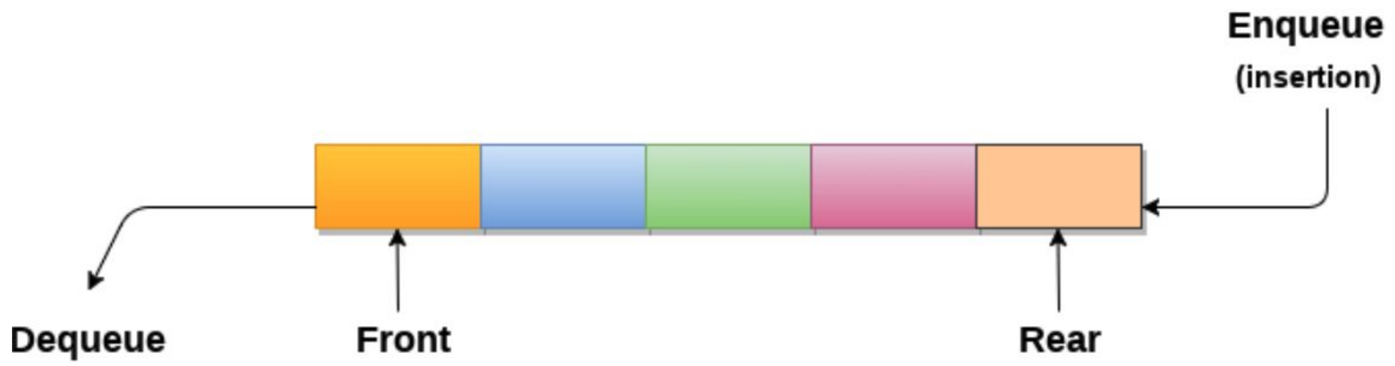
Introduction to Queue

Relevel
by Unacademy

Introduction to Queues

Queues as a data structure works very similar to how a queue works in real life as well.

Introduction to Queues



Operations in Queues

Queue has a data structure has two fundamental and primary operations:

- **Enqueue:** Inserting or pushing an element in the queue
- **Dequeue:** Removing an element from the queue

Introduction to Queues

Implementations of Queue

Arrays

LinkedList

Stack

Introduction to Queues

Queues using Arrays

Since we are using Arrays to implement Queues, we will be using an Array, and two pointers, front and rear.

Front pointer basically points at the front of the queue.
Rear pointer points toward the end of the queue.

Introduction to Queues

```
class Queue {

    constructor(){
      this.data = [];
      this.rear = 0;
      this.front = 0;
    }
}
```

Introduction to Queues

Enqueue:

An element can only be added to the queue from the end, so the rear pointer is shifted in this scenario.

Rear pointer is by default pointing at the position where an element can be added. So, simply add the element at the rear index and shift the rear pointer by one step.

```
enqueue(ele) {
        this.data[this.rear] = ele;
        this.rear = this.rear + 1;
}
```

Introduction to Queues

Deque:

An element can only be removed from the queue from the start, so the front pointer is shifted in this scenario.

Front pointer is by default pointing at the element that needs to be removed. So, simply remove the element at the front index and shift the front pointer by one step.

```
   dequeue() {
          if(this.isEmpty() === false) {
                This.data[this.front] = 0;
                this.front = this.front + 1;
                return this.data[this.front];
          }
        }
```

Queues using LinkedList

Similar to Arrays, Queues can be implemented using LinkedList as well.

Since we have already done LinkedList in the previous class, it should be pretty simple to understand.
Keep in mind the FIFO functionality.

Enqueue Operation:
Enqueue can be considered similar to addAtHead() operation of Linkedlist.

Time Complexity of Enqueue Operation: O(1)

Queues using LinkedList

Dequeue Operation:

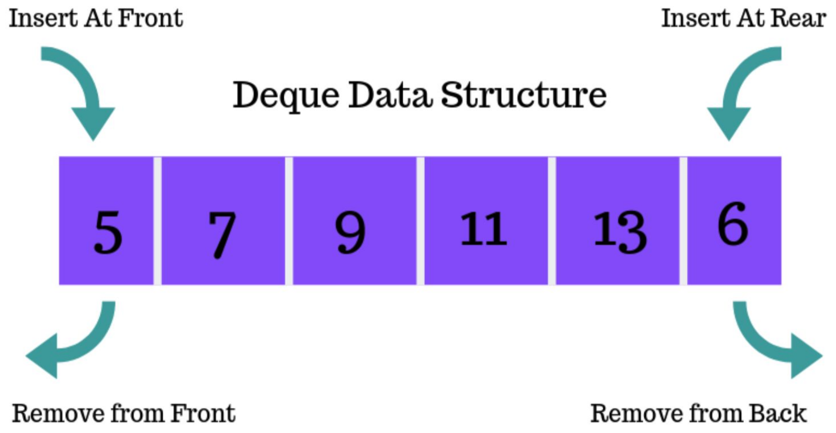
Enqueue can be considered similar to removeAtLast() operation of Linkedlist.

Time Complexity of Dequeue Operation: O(n)

Encourage students to try the implementation of Queues using Linkedlist.

Deque

In a Deque or Double-ended queue, data can be added and removed from both the ends of the queue.
It performs both the combined operations of stack and queue together and can be used as any of them. New items can be added at either the front or the rear. Likewise, existing items can be removed from either end.



Deque

Operations in Deque

Queue has a data structure has two fundamental and primary operations:

- addFront: Inserts an element at the front.
- addBack: Inserts an element at the back.
- removeFront: Removes an element from the front.
- removeBack: Removes an element from the back

Deque

Implementations of Deque

Arrays

Doubly LinkedList

Deque

Deque using Array

Since we are using Arrays to implement Deque, so similar to Queues, we will be using an Array, and two pointers, front and rear.

Front pointer basically points at the front of the queue.
Rear pointer points toward the end of the queue

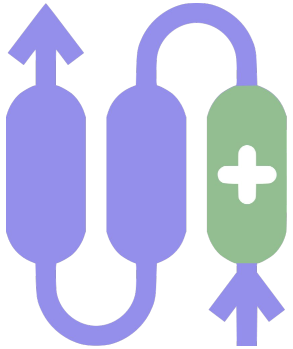
```
class Deque {
   constructor() {
   this.items = {};
   this.rear = 0;
   this.front = 0;
  }  }
```

Deque

addBack:

Dequeue addBack is similar to Queue's Enqueue method.

```
addBack(element) {
    this.items[this.rear] = element;
    this.rear++;
}
```

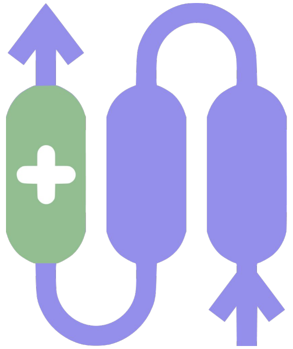


Deque

addFront:

When adding an element to the front of Deque, there are three scenarios,

1. If the Deque is Empty then same as addBack method.
2. When an element is removed from the front of the Deque , front > zero,
   - Then decrement the count
   - Assign the element to that front index
3. If the front is equal to zero then, we need to shift the element by one position right and free the first position and assign the element at the first index.



Deque

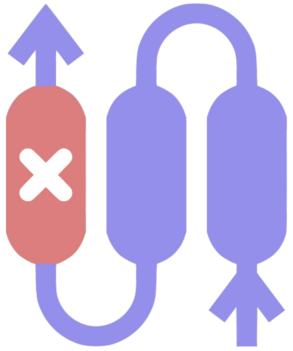
```
    addFront(element) {
        if (this.isEmpty()) {                 //1
            this.addBack(element);
        } else if (this.front  > 0) {     //2
            this.items[this.front] = element;
            this.front --;
            this.items[this.front] = element;
        } else {                                    //3
       for (let index = this.rear; index > 0; index--) {
                this.items[index] =  this.items[index -1];
            }             this.rear++;
            this.items[0] = element;
        }     return true;
   }
```

Deque

removeFront:

Dequeue removeFront is similar to Queue's Deque method.

```
removeFront() {
    if (this.isEmpty()) {
        return undefined;
    }
    let result = this.items[this.front];
    this.items[this.front] = 0;
    this.front++;
    return result;
}
```

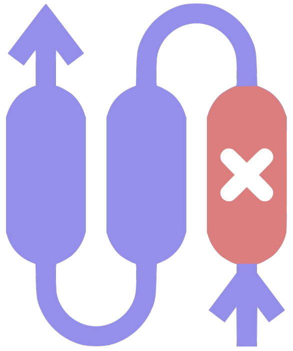


Deque

removeBack:

RemoveBack method of the Deque is similar to the pop method of stack.

```
removeBack() {
    if (this.isEmpty()) {
        return undefined;
    }
    let result = this.items[this.rear - 1];
    this.items[this.rear - 1] = 0;
    this.rear--;
    return result;
}
```



Thank you!