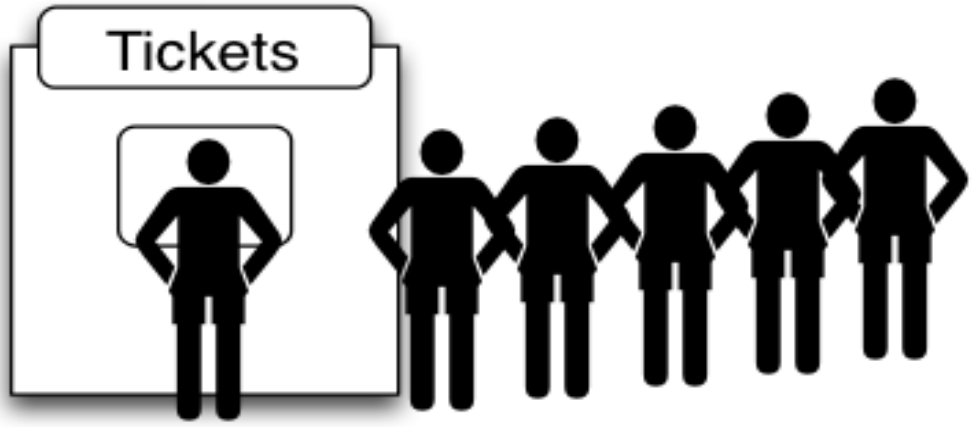
An Introduction to Queue

By Anshika Subodh Singh

Remember the hot summer afternoon when you were standing amongst the huge crowd outside the entrance of your favourite water park, waiting to get in. Being a shy kid in a situation where hundreds of people are waiting eagerly to push their way into cool relaxing water rides, your chances to get an entry pass soon would've been thin. But you're glad that there's a "queue" formed and service will be provided only according to "first come, first serve" basis. So little wait will surely yield results.



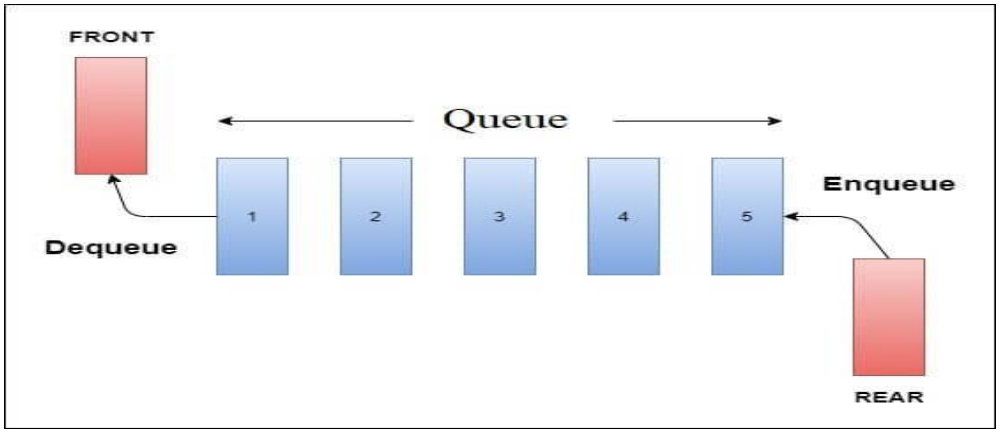
You must have come across situations like this where problems were solved just by forming a simple queue. Take, for example, single-lane one-way road, queues at the ticket windows or more technical applications like printer printing, CPU task scheduling and many more to be mentioned later in the article.
So, the question is, what is a queue and more importantly could it be of some use to you? And How?

What is Queue?

The programming world is all about forming logic to solve real-world problems since our computers only understand the language of 0's and 1's. Data is the main component of the process. In order to organize, maintain and store data, we provide structure to it which is known by the term Data Structure. Some famous examples you'd come across are linked list, array, etc. These are the specific ways of implementing Abstract Data Structures.
In order to understand the relation between the two, let's take an example:

> Suppose you are designing a mobile robot that is going to participate in a maze runner competition. During the competition, your bot has to find the shortest path from the start point to the endpoint. At this moment, you just know how to identify the start and the endpoint and where they exist on the map but the specific "shortest route" has not been defined yet or found out till the time, bot explores the map.

Abstract Data Structure is the abstraction of a data structure. It only has an interface and doesn't provide any detail of how something will be implemented. Data Structure, on the other hand, is the specific method of implementing the task.



Queue is an abstract linear data structure that models real-world queue by having two primary operations enqueue and dequeue. Since it is representative of a real-world queue, you would have guessed that elements could be added(enqueued) or removed (dequeued) in the "first come, first serve" manner or "**First-in, First-out**" (**FIFO**) order.

Real-world examples could be the conveyor belt model, Car queue, queue at the ticket window, etc.

Characteristics of a Queue

- Element could only be added from one end, called the tail.
- Element could only be removed from another end, called the head or front.
- Element existing in between cannot be accessed. Nor could new elements could be inserted.
- Deque is a double-ended queue in which insertion and deletion could be done from both ends.
- Queue can handle different data types.

Implementation:

Operations that can be performed:

Sr. no.	Operation	Explanation	Time Complexity,O(n)
1.	enqueue()	Adds an element to the queue	O(1) , constant
2.	dequeue()	Removes an element from the queue	O(1), constant
3.	peek()	Gets an element from the front without removing it	O(1), constant
4.	isFull()	Checks if the queue is full. Returns true if full and false otherwise.	O(1), constant
5.	isEmpty()	Checks if the queue is empty.	O(1), constant

		Returns true if empty and false otherwise.	
6.	size()	Checks the size of the queue	O(1), constant

Example to show the execution of operations:

Here, we are going to execute a sequence of operations on an empty queue called ‘q’, to demonstrate how the queue functions. The sequence is in order from 1 to 9.

Operation Sequence	**Queue Operation**	**Queue Contents**	**Return Value**
1.	q.isEmpty()	[ ]	True
2.	q.enqueue(4)	[4]	
3.	q.enqueue(‘dog’)	[‘dog’, 4]	
4.	q.size()	[True, ‘dog’, 4]	3
5.	q.isEmpty()	[True, ‘dog’, 4]	False
6.	q.enqueue(8,4)	[8, 4, True, ‘dog’, 4]	
7.	q.dequeue()	[8, 4, True, ‘dog’]	4
8.	q.dequeue()	[8, 4, True]	‘dog’
9.	q.size()	[8, 4, True]	2

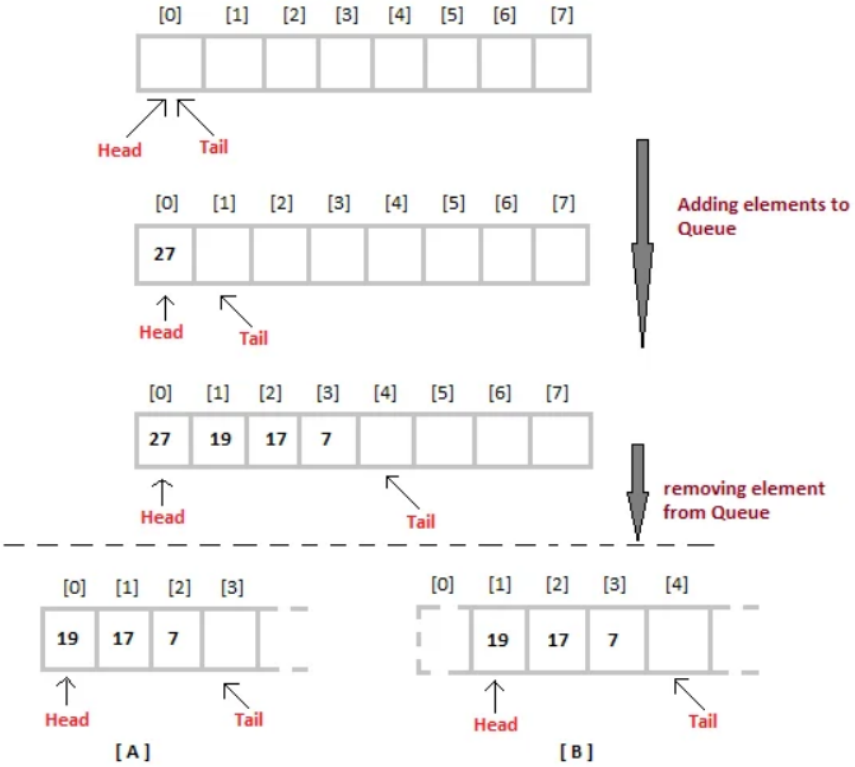
Data Structures that can be used to implement Queue:

Following data structures can be used to implement queue:

1. Array
2. Stack
3. Linked List

The easiest method to implement queue is through an array.

Queue Implementation using an array:



1. To implement queue using an array, we first need to define an array of some size.
2. In the beginning, two pointers namely head and tail will point at the index of the array. As new elements are added, the head pointer remains fixed while the tail pointer moves forward.
3. New elements are inserted where the tail pointer points.
4. Algorithm to execute enqueue:
    a. Check if the queue is full
    b. If it is full then print overflow and exit.
    c. If it isn't full then add the element and increment the tail pointer.
5. Algorithm to execute dequeue:
    a. Check if the queue is empty.
    b. If it is empty then print underflow error and exit.
    c. If it isn't empty then remove the element and decrement the tail pointer.
6. Algorithm to remove head element:
    a. Method1: Remove the element at which the head pointer points and shift all the elements leftwards.
    b. Method2: Remove the element at which the head pointer points. Make the head pointer point towards the element present at the next location.
7. To execute peek(), just access the element whose position is pointed out by the head pointer.

Code Snippet:

```
class Queue(object):
    def __init__(self,qSize):
        self.qSize=qSize
        self.headIndex=0
        self.tailIndex=0
        self.data=[]

    def isEmpty(self):
        return self.size==0

    def isFull(self):
        return self.size==self.qSize

    def peek(self):
        return self.data[self.headIndex]

    def enqueue(self,x):
        if not self.isFull():
            self.data.append(x)
            self.tailIndex+=1
        else:
            print("Queue already full")

    def dequeue(self):
        if not self.isEmpty():
            self.data.pop()
            self.tailIndex-=1
        else:
            print("Queue is empty")


    def size(self):
        return self.tailIndex

    def __str__(self):
        return "Queue: "+str(self.data)

q=Queue(6)
print(q)
a1=q.enqueue(1)
print(q)
```

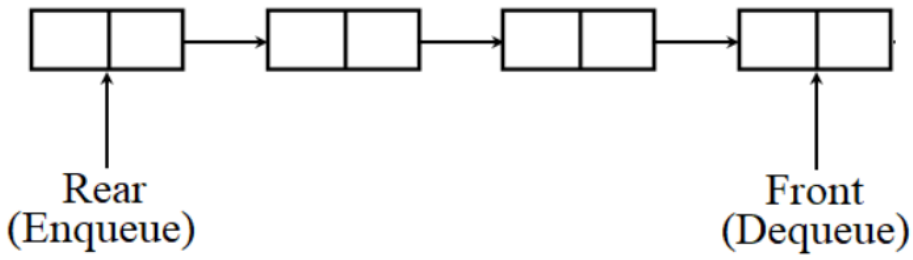
```
a2=q.enqueue(67)
print(q)
a4=q.dequeue()
print(q)
a5=q.size()
print(q)
a6=q.isFull()
a7=q.isEmpty()
```

OUTPUT:

```
Queue: []
Queue: [1]
Queue: [1, 67]
Queue: [1]
Queue: [1]
```

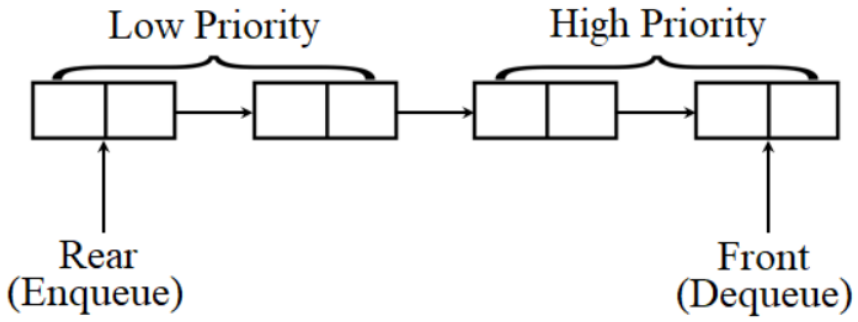
Types of Queue:

1. Simple Queue



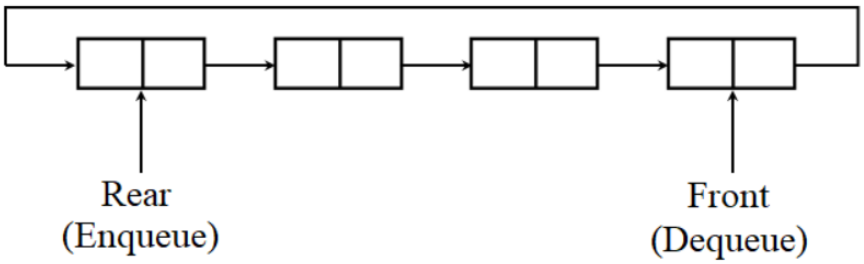
2. Priority Queue:

   Priority queue has each element some weightage or priority of service. An element with high priority will be dequeued first by default.



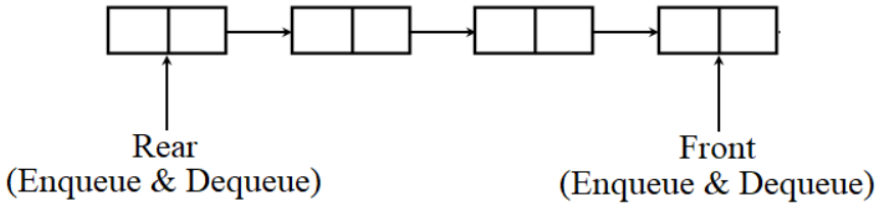
3. Circular Queue:

   In this queue, tail of the queue points at the head. Hence, a new element could be added at the head if the node position is free and when tail is full. It is also known as ring-buffer.



4. Double-ended Queue or Deque:

   Here enqueue and dequeue operations can take place at both ends.



Advantages of Using Queue:

1. Fast
2. Optimised
3. Flexible
4. Handle multiple data types

Disadvantages of using Queue:

1. Time taken to search queue is Linear, i.e, O(n) is the time complexity of searching a queue.
2. Removal and insertion of elements at mid positions are difficult.

Applications of Queue:

1. Breadth-First Seach (BFS) algorithms like A* make use of queue.
2. Minimum Spanning Tree (MST) algorithms also use queue.
3. Handling Interrupts in real-time systems.
4. Serving requests on a single shared resource like printer, CPU task scheduling, etc.

Conclusion

Stacks and queues are usually the building blocks of a complex problem. In this article, we were introduced to the concept of queue along with real-world applications. Being an abstract data structure, it could be implemented using various data structures, in any language of your choice and be custom designed for the application you're working on. We even added some code snippets implementing the same in python. Go forth and try writing code for the same in your favourite language.