Types of Queues

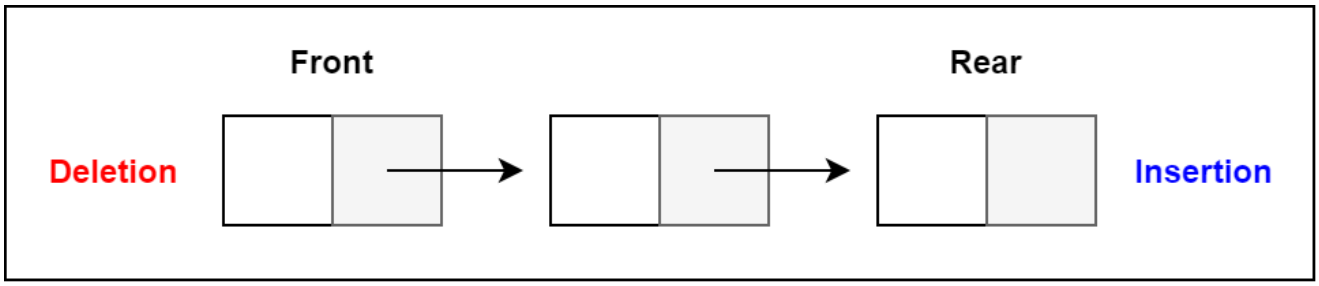
Alright, students, today we'll learn about a fascinating data structure known as the queue. Imagine a ticket queue outside a cinema hall where people stand in line, and the first person to enter the queue is the first one to get the ticket. That's precisely how a queue works!

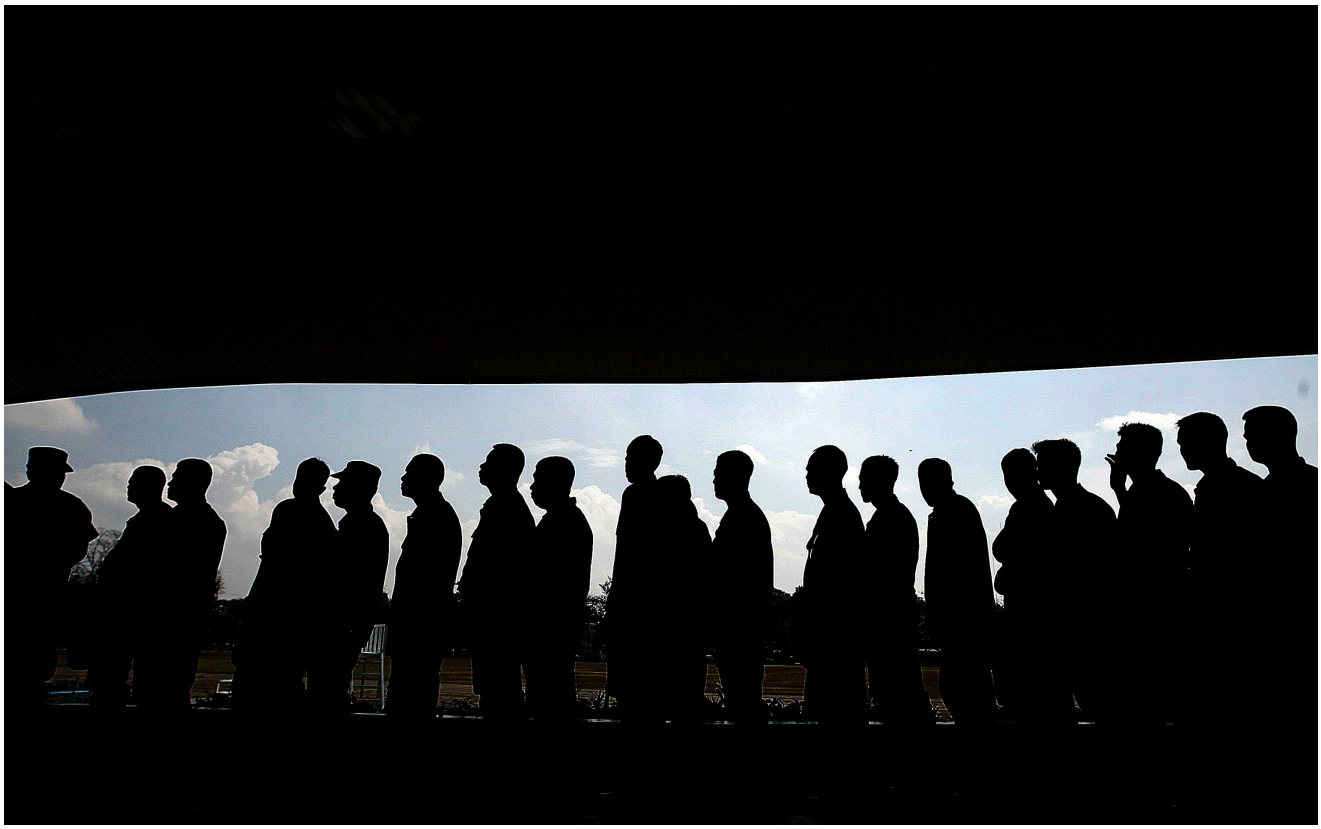
Now, let's talk about the four different types of queues:

1. **Simple Queue**: It's the basic type of queue we just discussed. Elements are added at the rear and removed from the front, following the First In First Out (FIFO) principle.
2. **Circular Queue**: This type of queue is like a variation of the simple queue. When the rear reaches the end of the queue, it wraps around to the beginning, allowing us to reuse empty spaces efficiently.
3. **Priority Queue**: Unlike a simple queue, a priority queue assigns a priority to each element. The element with the highest priority gets dequeued first, regardless of its position in the queue.
4. **Double Ended Queue**: Also known as Deque, this type of queue allows us to insert and remove elements from both ends. It offers more flexibility in managing elements compared to other types of queues.

These different types of queues have their unique features and use cases. Understanding them will help you become proficient in choosing the right queue for specific programming problems. So, let's dive in and explore the world of queues!

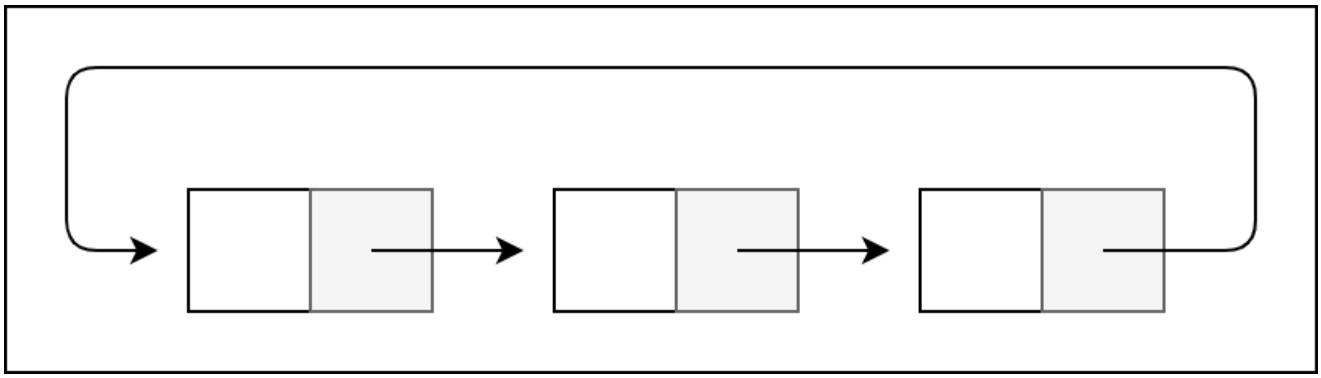
Simple Queue





In a simple queue, we have a straightforward way of doing things. When we add an element, we do it at the rear of the queue. And when we remove an element, it comes out from the front. It's like standing in a line to get something, and the first person who entered the line gets served first. This is known as the FIFO rule, which stands for First In First Out. So, the element that got in first will be the first one to get out. It's a simple and organized way of managing our data!

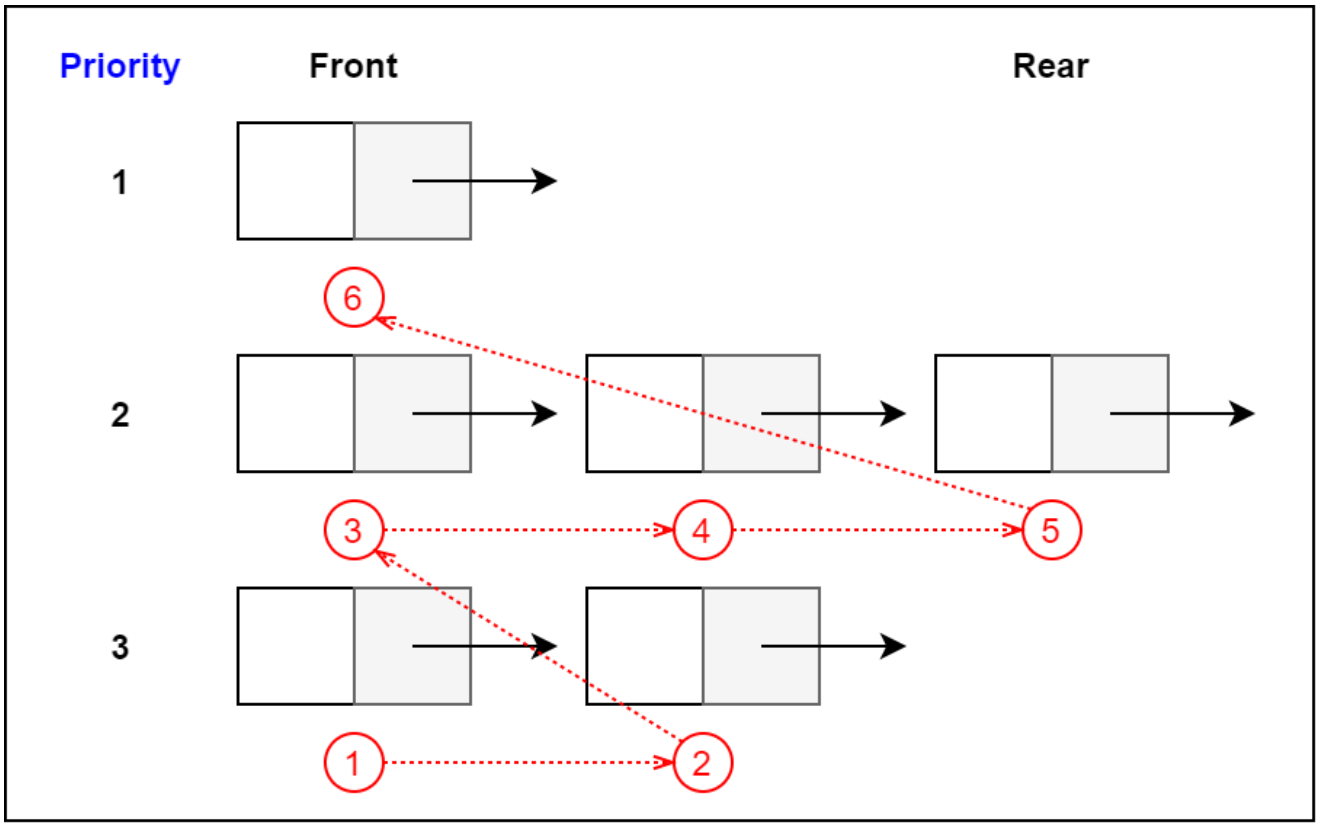
Circular Queue



Now, let's explore the circular queue! In a circular queue, we make a special kind of connection between the last element and the first element, creating a circular link. This means that after the last element, the next element is the first element, and the cycle continues.

One of the main advantages of a circular queue over a simple queue is that it offers better memory utilization. You see, in a simple queue, when the last position is full and the first position is empty, we cannot insert any more elements at the beginning. But in a circular queue, since it forms a circle, we can insert an element in the first position even if the last position is full. This gives us more flexibility in managing our data and makes it more efficient in terms of memory usage. It's a clever way to make the most out of our queue!

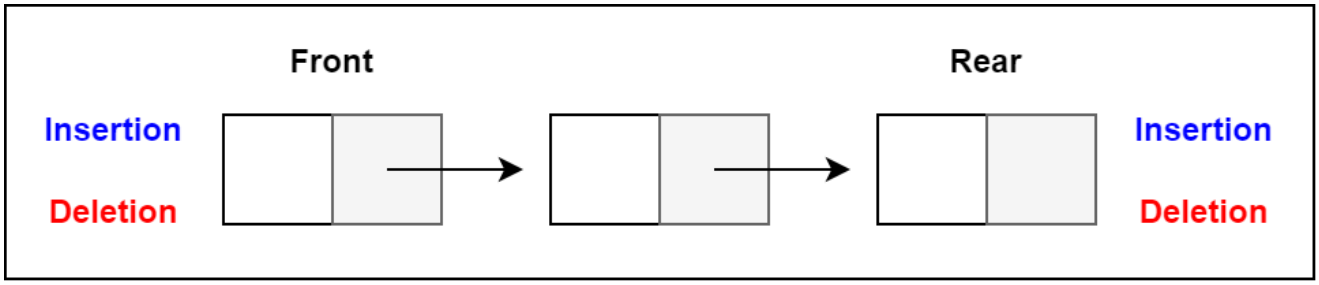
Priority Queue



Now, let's discuss the intriguing concept of a priority queue! In a priority queue, each element is not just an ordinary item, but it also comes with a priority assigned to it. This priority determines how the elements are served in the queue. The element with the highest priority gets served first, and if there are multiple elements with the same priority, they are served based on the order in which they arrived.

So, when we insert elements into the priority queue, we consider the order of their arrival. However, when it comes to removing elements, we pay attention to their priorities. The element with the highest priority will be the first to be removed from the queue. This makes the priority queue a very powerful and versatile data structure, as it allows us to handle items based on their significance or importance. It's like giving VIP treatment to the most important elements in our queue!

Deque (Double Ended Queue)



Now, let's explore the fascinating world of the double-ended queue, often known as a deque. Unlike the other queues we've discussed so far, the deque is unique because it allows us to

perform both insertion and removal of elements not only from the front but also from the rear!

Picture it like this: imagine you have a queue with two ends, one at the front and the other at the rear. This special feature means that the deque does not strictly follow the traditional FIFO (First In First Out) rule like the simple queue does.

Because of its flexibility, the deque is an incredibly versatile data structure. It allows us to efficiently add and remove elements from both ends, making it suitable for various scenarios and problem-solving situations. Whether we need to add something important at the front or remove something from the rear, the deque has got us covered!