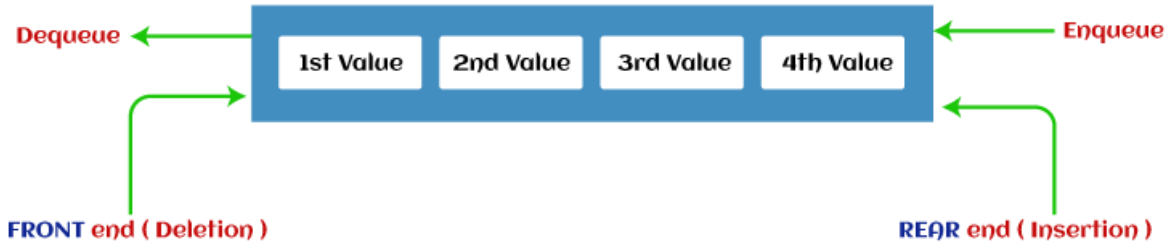
Queue

- Queue is the data structure that is similar to the queue in the real world.
- A queue is a data structure in which whatever comes first will go out first, and it follows the FIFO (First-In-First-Out) policy.
- Queue can also be defined as the list or collection in which the insertion is done from one end known as the **rear end** or the **tail** of the queue,
- whereas the deletion is done from another end known as the **front end** or the **head** of the queue.
- The real-world example of a queue is the ticket queue outside a cinema hall, where the person who enters first in the queue gets the ticket first, and the last person enters in the queue gets the ticket at last. Similar approach is followed in the queue in data structure.



Advantages of Queue:

- A large amount of data can be managed efficiently with ease.
- Operations such as insertion and deletion can be performed with ease as it follows the first in first out rule.
- Queues are useful when a particular service is used by multiple consumers.
- Queues are fast in speed for data inter-process communication.
- Queues can be used in the implementation of other data structures.

Disadvantages of Queue:

- The operations such as insertion and deletion of elements from the middle are time consuming.
- Limited Space.
- In a classical queue, a new element can only be inserted when the existing elements are deleted from the queue.
- Searching an element takes O(N) time.
- Maximum size of a queue must be defined prior.

Operations performed on queue

The fundamental operations that can be performed on queue are listed as follows -

- **Enqueue:** The Enqueue operation is used to insert the element at the rear end of the queue. It returns void.
- **Dequeue:** It performs the deletion from the front-end of the queue. It also returns the element which has been removed from the front-end. It returns an integer value.
- **Peek:** This is the third operation that returns the element, which is pointed by the front pointer in the queue but does not delete it.

- **Queue overflow (isfull):** It shows the overflow condition when the queue is completely full.
- **Queue underflow (isempty):** It shows the underflow condition when the Queue is empty, i.e., no elements are in the Queue.

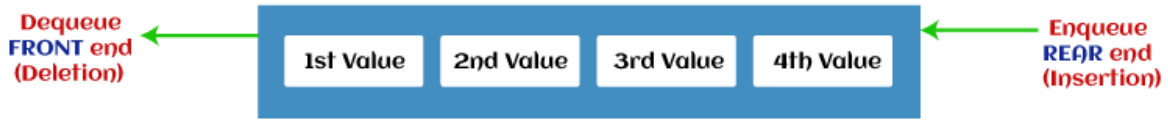
Types of Queue

There are four different types of queue that are listed as follows -

- Simple Queue or Linear Queue
- Circular Queue
- Priority Queue
- Double Ended Queue (or Deque)

Simple Queue or Linear Queue

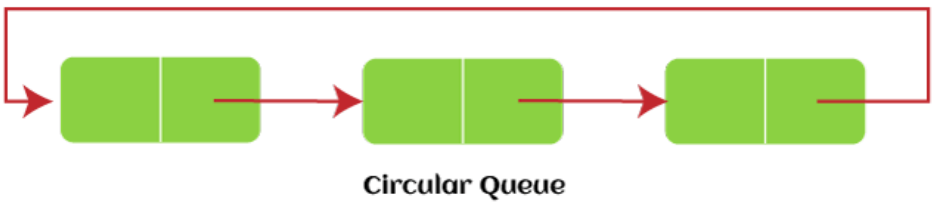
In Linear Queue, an insertion takes place from one end while the deletion occurs from another end. The end at which the insertion takes place is known as the rear end, and the end at which the deletion takes place is known as front end. It strictly follows the FIFO rule.



The major drawback of using a linear Queue is that insertion is done only from the rear end. If the first three elements are deleted from the Queue, we cannot insert more elements even though the space is available in a Linear Queue. In this case, the linear Queue shows the overflow condition as the rear is pointing to the last element of the Queue.

Circular Queue

In Circular Queue, all the nodes are represented as circular. It is similar to the linear Queue except that the last element of the queue is connected to the first element. It is also known as Ring Buffer, as all the ends are connected to another end. The representation of circular queue is shown in the below image -



The drawback that occurs in a linear queue is overcome by using the circular queue. If the empty space is available in a circular queue, the new element can be added in an empty space by simply incrementing the value of rear. The main advantage of using the circular queue is better memory utilization.

Applications

Ring Buffers are common data structures frequently used when the input and output to a data stream occur at different rates.

- Buffering Data Streams
- Computer Controlled Trafficking signal systems
- Memory Management
- CPU scheduling

Advantages

Circular Queues offer a quick and clean way to store FIFO data with a maximum size.

- Doesn't use dynamic memory → No memory leaks
- Conserves memory as we only store up to our capacity (opposed to a queue which could continue to grow if input outpaces output.)
- Simple Implementation → easy to trust and test
- Never has to reorganize / copy data around
- All operations occur in constant time O(1)

Disadvantages

Circular Queues can only store the pre-determined maximum number of elements.

Array Implementation of operations on circular queue

There are two primary operations on a circular Queue:

1: **Enqueue(item) :** add item to the Queue.

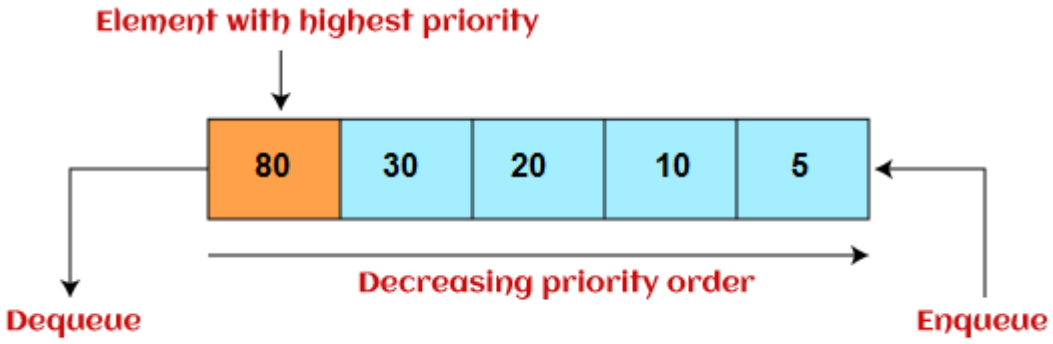
```
if Queue.isfull()
   print "Queue is Full"
else
   increase tail by 1
   Queue[tail] = item
   size++
```

2: **Dequeue()**: return the item at the front (head) of the line and remove it.

```
if Queue.isEmpty()
   print "Queue is Empty"
else
   tmp = Queue[head]
   increase head by 1
   size--
   return tmp
```

Priority Queue

It is a special type of queue in which the elements are arranged based on the priority. It is a special type of queue data structure in which every element has a priority associated with it. Suppose some elements occur with the same priority, they will be arranged according to the FIFO principle. The representation of priority queue is shown in the below image -



Insertion in priority queue takes place based on the arrival, while deletion in the priority queue occurs based on the priority. Priority queue is mainly used to implement the CPU scheduling algorithms.

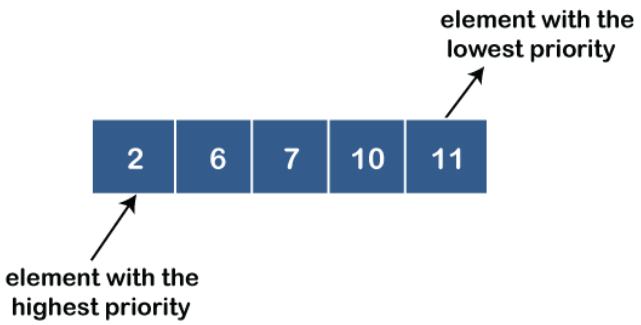
Characteristics of a Priority queue

A priority queue is an extension of a queue that contains the following characteristics:

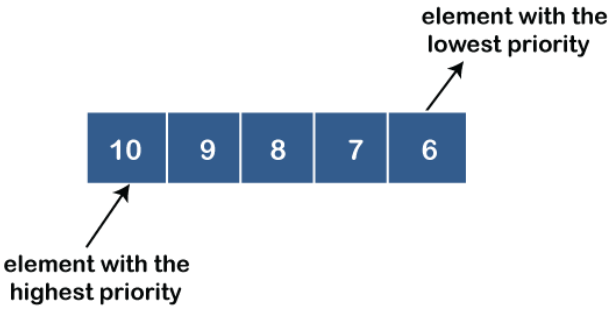
- Every element in a priority queue has some priority associated with it.
- An element with the higher priority will be deleted before the deletion of the lesser priority.
- If two elements in a priority queue have the same priority, they will be arranged using the FIFO principle.

There are two types of priority queue that are discussed as follows –

- **Ascending priority queue -** In ascending priority queue, elements can be inserted in arbitrary order, but only smallest can be deleted first. Suppose an array with elements 7, 5, and 3 in the same order, so, insertion can be done with the same sequence, but the order of deleting the elements is 3, 5, 7.



- **Descending priority queue -** In descending priority queue, elements can be inserted in arbitrary order, but only the largest element can be deleted first. Suppose an array with elements 7, 3, and 5 in the same order, so, insertion can be done with the same sequence, but the order of deleting the elements is 7, 5, 3.



Application of Priority Queue

The following are the applications of a Priority Queue:

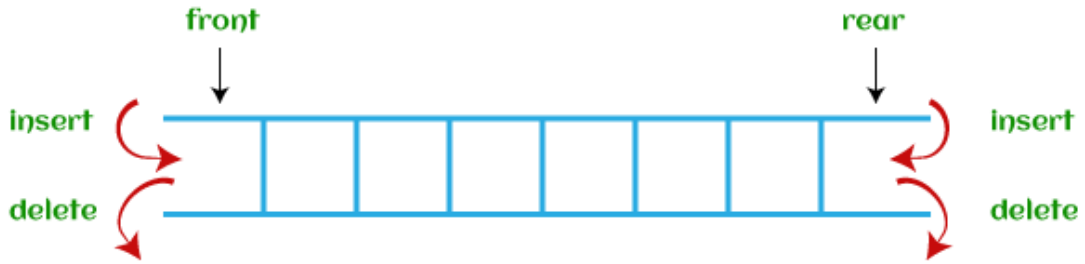
- It is used in Djikstra's Algorithm – To find the shortest path between nodes in a graph.
- It is used in Prim's Algorithm – To find the Minimum Spanning Tree in a weighted undirected graph.
- It is used in Heap Sort – To sort the Heap Data Structure
- It is used in Huffman Coding – A Data Compression Algorithm
- It is used in Operating Systems for:
    1. Priority Scheduling – Where processes must be scheduled according to their priority.
    2. Load Balancing – Where network or application traffic must be balanced across multiple servers.
    3. Interrupt Handling – When a current process is interrupted, a handler is assigned to the same to rectify the situation immediately.
- A* Search Algorithm – A graph traversal and a path search algorithm

Double Ended Queue

In Deque or Double Ended Queue, insertion and deletion can be done from both ends of the queue either from the front or rear. It means that we can insert and delete elements from both front and rear ends of the queue. Deque can be used as a palindrome checker means that if we read the string from both ends, then the string would be the same.

Deque can be used both as stack and queue as it allows the insertion and deletion operations on both ends. Deque can be considered as stack because stack follows the LIFO (Last In First Out) principle in which insertion and deletion both can be performed only from one end. And in deque, it is possible to perform both insertion and deletion from one end, and Deque does not follow the FIFO principle.

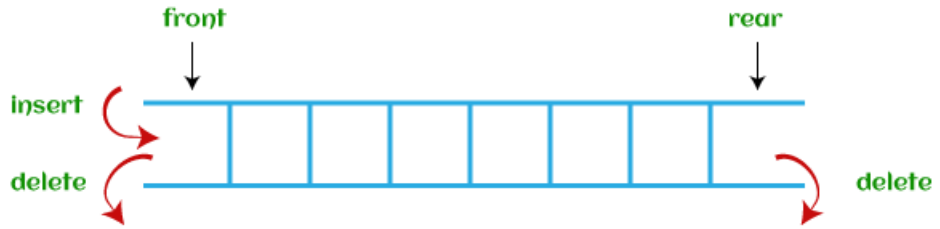
The representation of the deque is shown in the below image -



double ended queue

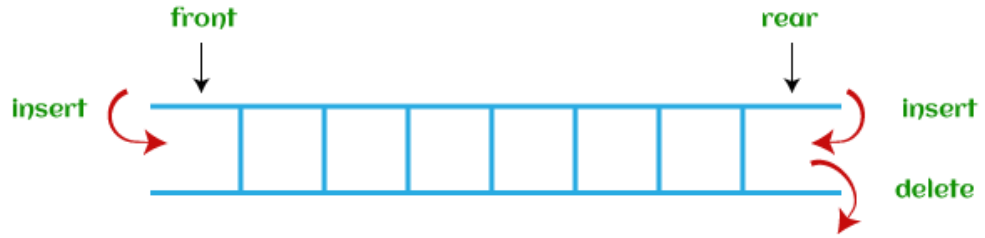
There are two types of deque that are discussed as follows -

- **Input restricted deque -** As the name implies, in input restricted queue, insertion operation can be performed at only one end, while deletion can be performed from both ends.



input restricted double ended queue

- **Output restricted deque -** As the name implies, in output restricted queue, deletion operation can be performed at only one end, while insertion can be performed from both ends.



Output restricted double ended queue