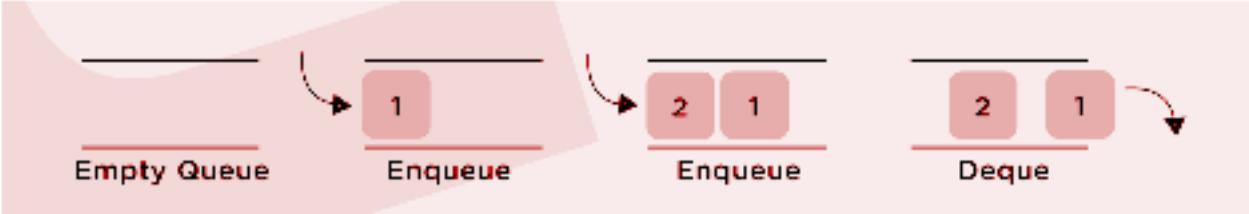
Queues

Introduction

- Like stack, the queue is also an abstract data type.
- As the name suggests, in queue elements are inserted at one end while deletion takes place at the other end.
- Queues are open at both ends, unlike stacks that are open at only one end(the top).

Let us consider a queue at a movie ticket counter:

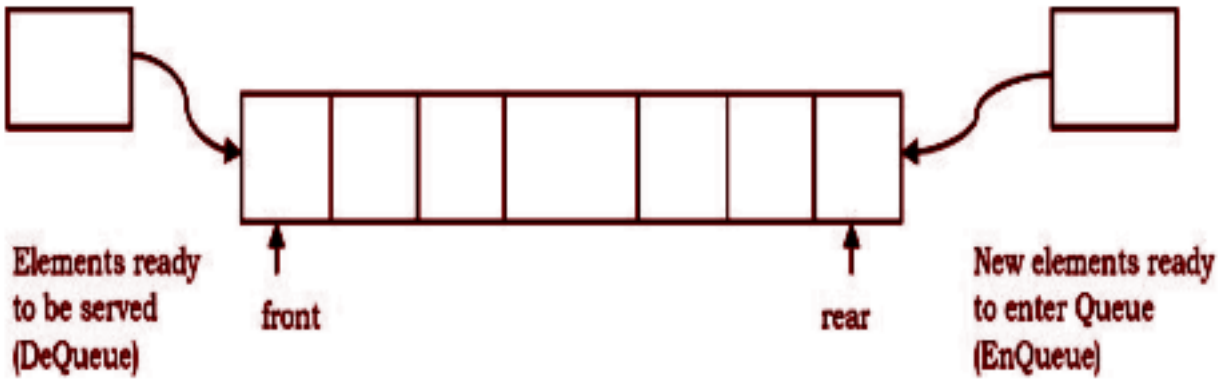
- Here, the person who comes first in the queue is served first with the ticket while the new seekers of tickets are added back in the line.
- This order is known as **First In First Out (FIFO)**.
- In programming terminology, the operation to add an item to the queue is called "enqueue", whereas removing an item from the queue is known as "dequeue".



Working of A Queue

Queue operations work as follows:

1. Two pointers called **FRONT** and **REAR** are used to keep track of the first and last elements in the queue.
2. When initializing the queue, we set the value of FRONT and REAR to -1.
3. On **enqueuing** an element, we increase the value of the REAR index and place the new element in the position pointed to by REAR.
4. On **dequeuing** an element, we return the value pointed to by FRONT and increase the FRONT index.
5. Before enqueuing, we check if the queue is already full.
6. Before dequeuing, we check if the queue is already empty.
7. When enqueuing the first element, we set the value of FRONT to 0.
8. When dequeuing the last element, we reset the values of FRONT and REAR to -1.



Implementation of A Queue Using Array

A Queue contains majorly these five functions that we will be implementing:

Main Queue Operations

• `enqueue(int data)`: Inserts an element at the end of the queue

• `int dequeue()`: Removes and returns the element at the front of the queue

Auxiliary Queue Operations

• `int front()`: Returns the element at the front without removing it

• `int size()`: Returns the number of elements stored in the queue

• `int IsEmpty()`: Indicates whether no elements are stored in the queue or not

Now, let's implement these functions in Python. Follow up the code along with the comments below:

```
class Queue():
    def __init__(self):
        self._queue = []
        self.size = 0 #Current size of the queue
        self.maxSize = 10 #Maximum size of the queue

    def enqueue(self, item): #Add an element to the queue
        if self.size < self.maxSize:
             self._queue.append(item) #Add element to last

    def dequeue(self): #Remove an element from the queue
        first = self._queue[0] #Remove the FIRST element
        del self._queue[0]
        return first
```

Queues using LL

Given below is an implementation of Queue using Linked List. This is similar to the way we wrote the LL Implementation for a Stack:

```
class Node: #Nodes of the Linked List
    def __init__(self, data):
        self.data = data
        self.next = None

class Queue: #Queue implementation using Linked List
    def __init__(self):
        self.front = self.rear = None

    def isEmpty(self):
        return self.front == None

    def enqueue(self, item):
        temp = Node(item) #Adding a new node
        if self.rear == None:
            self.front = self.rear = temp
            return
        self.rear.next = temp
        self.rear = temp

    def dequeue(self):
        if self.isEmpty():#If there is no element to dequeue
            return
        temp = self.front
        self.front = temp.next

        if(self.front == None):
            self.rear = None
```

Queue using Python List

`List` is Python's built-in data structure that can be used as a queue. Instead of `enqueue()` and `dequeue()`, `append()` and `pop()` function is used.

```
#Inbuilt implementation of Queue using List
queue = []

# Adding elements to the queue
queue.append('1')#Using the .append() function
queue.append('2')
queue.append('3')

print("Initial Queue:")
print(queue)#Queue after appending the elements

# Removing elements from the queue
print("\nElements dequeued from queue:")
print(queue.pop(0))#Removing first element from queue
print(queue.pop(0))
print(queue.pop(0))

print("\nQueue after removing elements:")
print(queue)
```

Output:

```
Initial queue:
['1', '2', '3']
Elements dequeued from queue:
1
2
3
Queue after removing elements:
[]
```

In-built Queue in Python

- **Queue** is built-in module of Python which is used to implement a queue.
- `queue.Queue(maxsize)` initializes a variable to a maximum size of maxsize.
- This Queue follows the **FIFO** rule.

There are various functions available in this module:

- **maxsize** – Returns the maximum number of items allowed in the queue.
- **empty()** – Returns **True** if the queue is empty, otherwise it returns **False**.
- **get()** – Remove and return an item from the queue.
- **put(item)** – Put an item into the queue.
- **qsize()** – Return the number of items currently present in the queue.

Note: A max size of zero '0' means an infinite queue.

```
from queue import Queue

q = Queue(maxsize = 3) # Initializing a queue

print(q.maxsize())# Maximum size of the Queue

q.put('14') # Adding elements to the queue
q.put('28')
q.put('36')

print("\nisFull: ", q.full()) # Check if the queue is full

print("\nElement dequeued from the queue: ")
print(q.get()) # Removing an element from queue

print("\nisEmpty: ", q.empty()) # Check if the queue is empty
```

We get the following output:

```
3

isFull:  True

Element dequeued from the queue:
14 # A queue follows FIFO

isEmpty:  False
```

Practice problems:

- https://www.spoj.com/problems/ADAQUEUE/
- https://www.hackerearth.com/practice/data-structures/queues/basics-of-queues/practice-problems/algorithm/number-recovery-0b988eb2/
- https://www.codechef.com/problems/SAVJEW
- https://www.hackerrank.com/challenges/down-to-zero-ii/problem