Queue

1.Introduction

Queues are useful to solve various system programs. Some simple applications of queues in our everyday life as well as in computer science.

Suppose there are a number of customers in front of a counter to get service (say, to collect tickets or to withdraw/deposit money in a teller of a bank). The customers are forming a queue and they will be served in the order they arrived, that is, a customer who comes first will be served first.

2 .Definition

Like a stack, a queue is an ordered collection of homogeneous data elements; in contrast with the stack, here, insertion and deletion operations take place at two extreme ends. A queue is also a linear data structure like an array, a stack and a linked list where the ordering of elements is in a-linear fashion. The only difference between a stack and a queue is that in the case of stack insertion and deletion (PUSH and POP) operations are at one end (TOP) only, but in a queue insertion (called ENQUEUE) and deletion (called DEQUEUE) operations take place at two ends called the REAR and FRONT of the queue, respectively. Figure represents a model of a queue structure. Queue is also termed first-in first-out (FIFO)

There are several operations performed in the queue:

- Firstly the queue is initialized to zero (i.e. empty).
- Determine whether the queue is empty or not.
- Determine if the queue is full or not.
- Insertion of the new element from the rear end (Enqueue).
- Deletion of the element from the front end (Dequeue).

Stacks	Queues
Uses LIFO (Last in, First out) approach.	Uses FIFO (First in, First out) approach.
Items are added or deleted from only one end called “Top” of the stack.	Items are added from “Rear” end of the queue and are removed from the “front” of the queue.
The basic operations for the stack are “push” and “Pop”.	The basic operations for a queue are “enqueue” and “dequeue”.
We can do all operations on the stack by maintaining only one pointer to access the top of the stack.	In queues, we need to maintain two pointers, one to access the front of the queue and the second one to access the rear of the queue.
The stack is mostly used to solve recursive problems.	Queues are used to solve problems related to ordered processing.

Applications Of Queue

Let us discuss the various applications of the queue data structure below.

1.The queue data structure is used in various CPU and disk scheduling. Here we have multiple tasks requiring CPU or disk at the same time. The CPU or disk time is scheduled for each task using a queue.

2.The queue can also be used for print spooling wherein the number of print jobs is placed in a queue.

3.Handling of interrupts in real-time systems is done by using a queue data structure. The interrupts are handled in the order they arrive.

4.Breadth-first search in which the neighboring nodes of a tree are traversed before moving on to next level uses a queue for implementation.

5.Call center phone systems use queues to hold the calls until they are answered by the service representatives.

Basic Operations

1. enqueue() − add (store) an item to the queue.
2. dequeue() − remove (access) an item from the queue.

Enqueue Operation

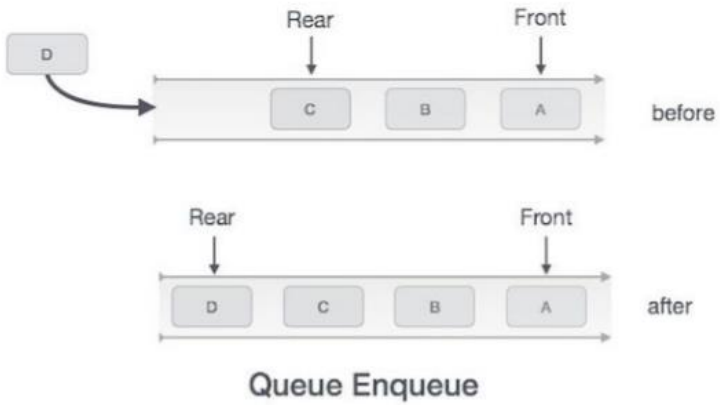
Queues maintain two data pointers, front and rear. Therefore, its operations are comparatively difficult to implement than that of stacks. The following steps should be taken to enqueue (insert) data into a queue –

Step 1 − Check if the queue is full.

Step 2 − If the queue is full, produce overflow error and exit.

Step 3 − If the queue is not full, increment rear pointer to point the next empty space. Step Step 4 − Add data element to the queue location, where the rear is pointing.

Step 5 − return success.



Queue Enqueue

Algorithm for enqueue operation

```
procedure enqueue(data)

   if queue is full
      return overflow
   endif

   rear ← rear + 1
   queue[rear] ← data
   return true

end procedure
```

Dequeue Operation

Accessing data from the queue is a process of two tasks − access the data front is pointing and remove the data after access. The following steps are taken to perform dequeue operation

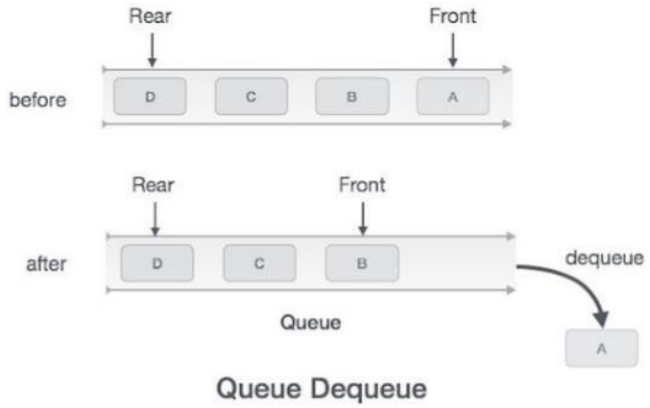
Step 1 − Check if the queue is empty.

Step 2 − If the queue is empty, produce underflow error and exit.

Step 3 − If the queue is not empty, access the data where front is pointing.

Step 4 − Increment front pointer to point to the next available data element

Step 5 − Return success



Queue Dequeue

Algorithm for dequeue operation

```
procedure dequeue

   if queue is empty
      return underflow
   end if

   data = queue[front]
   front ← front + 1
   return true

end procedure
```