DATA STRUCTURE AND ALGORITHM

LECTURER: ER. SAROJ GHIMIRE
QUALIFICATION: MSC.CSIT, BE(COMPUTER)

Lincoln University College

Objectives

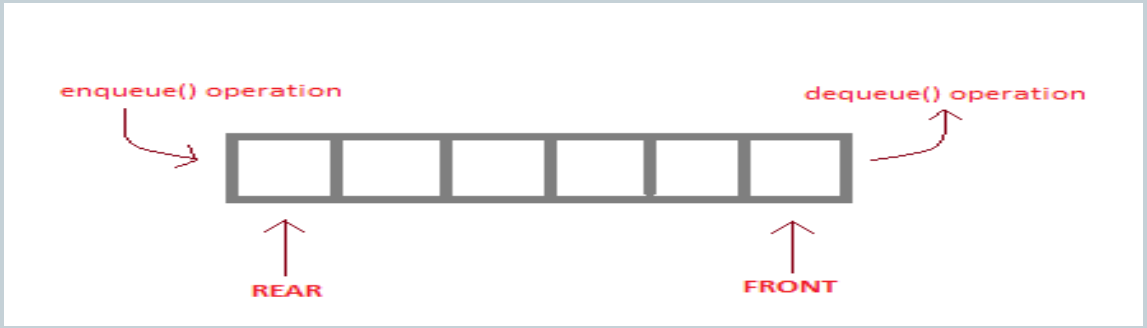
- Examine queue processing
- Define a queue abstract data type
- Demonstrate how a queue can be used to solve problems
- Examine various queue implementations

Overview

- Basic Concept of Queue, Queue as ADT, Basic Operation
- Linear Queue, Circular Queue, Priority Queue
- Application of Queue

Concept Of Queue

- Queue is also an abstract data type or a linear data structure in which the first element is inserted from one end called the REAR(also called tail), and the removal of existing element takes place from the other end called as FRONT(also called head
- This makes queue as **FIFO**(First in First Out) data structure, which means that element inserted first will be removed first
- Which is exactly how queue system works in real world. If you go to a ticket counter to buy movie tickets, and are first in the queue, then you will be the first one to get the tickets. Right? Same is the case with Queue data structure. Data inserted first, will leave the queue first
- The process to add an element into queue is called **Enqueue** and the process of removal of an element from queue is called **Dequeue**



Queue as ADT

- Queue() creates a new queue that is empty. It needs no parameters and returns an empty queue.
- Enqueue(item) adds a new item to the rear of the queue. It needs the item and returns nothing.
- Dequeue() removes the front item from the queue. It needs no parameters and returns the item. The queue is modified.
- Is Empty() tests to see whether the queue is empty. It needs no parameters and returns a Boolean value.
- size() returns the number of items in the queue. It needs no parameters and returns an integer

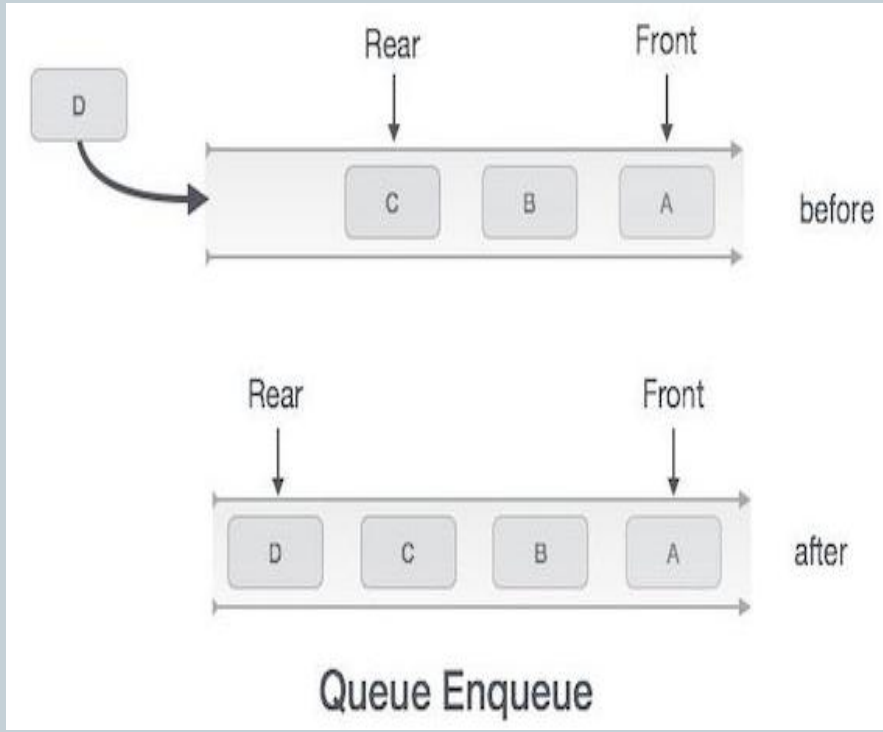
Queue operation

Enqueue Operation

- Queues maintain two data pointers, front and rear. Therefore, its operations are comparatively difficult to implement than that of stacks.

Algorithm:

1. Check if the queue is full.
2. If the queue is full, produce overflow error and exit.
3. If the queue is not full, increment rear pointer to point the next empty space.
4. Add data element to the queue location, where the rear is pointing.
5. Return success.



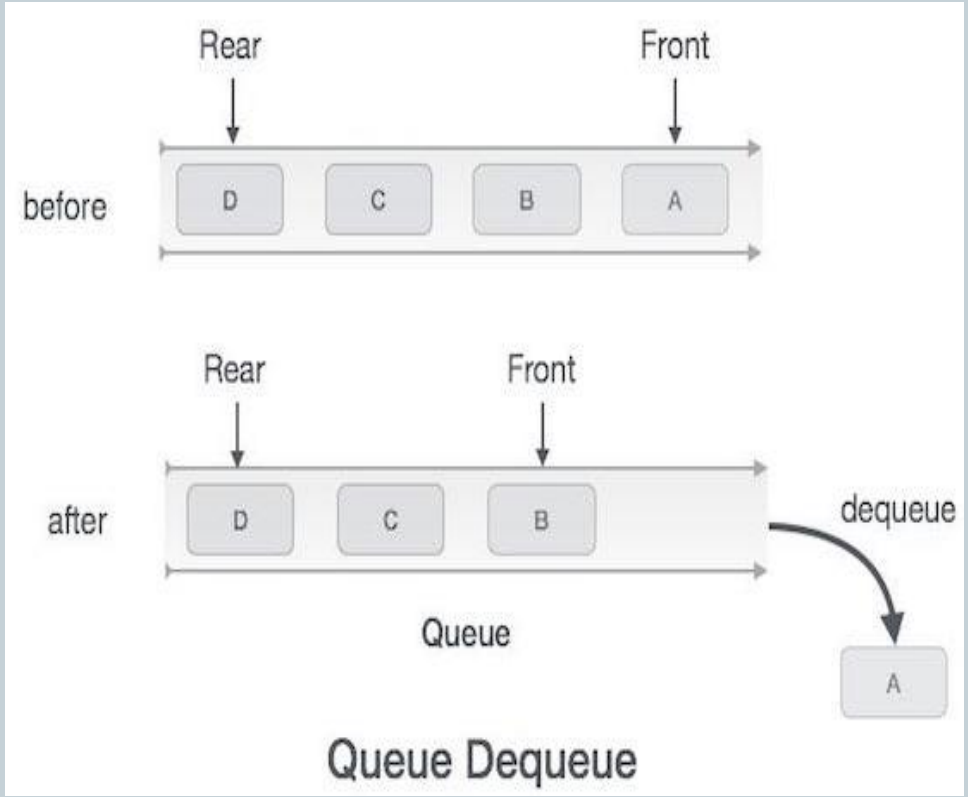
Queue Enqueue

Queue operation

- Dequeue Operation
  - Accessing data from the queue is a process of two tasks – access the data where front is pointing and remove the data after access.

    Algorithm :

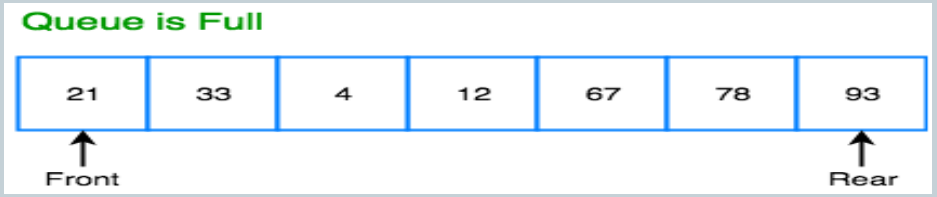
    1. Check if the queue is empty.
    2. If the queue is empty, produce underflow error and exit.
    3. If the queue is not empty, access the data where front is pointing.
    4. Increment front pointer to point to the next available data element.
    5. Return success



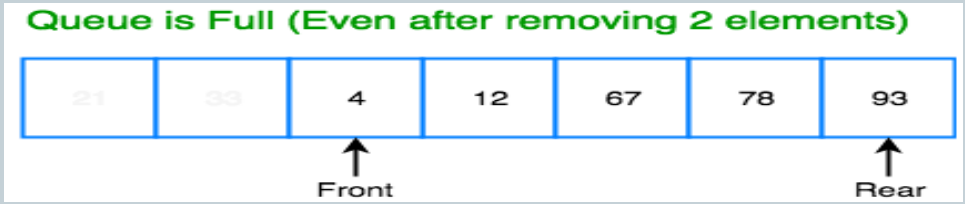
Queue Dequeue

Circular Queue

- In a Linear queue, once the queue is completely full, it's not possible to insert more elements. Even if we dequeue the queue to remove some of the elements, until the queue is reset, no new elements can be inserted. You must be wondering why?



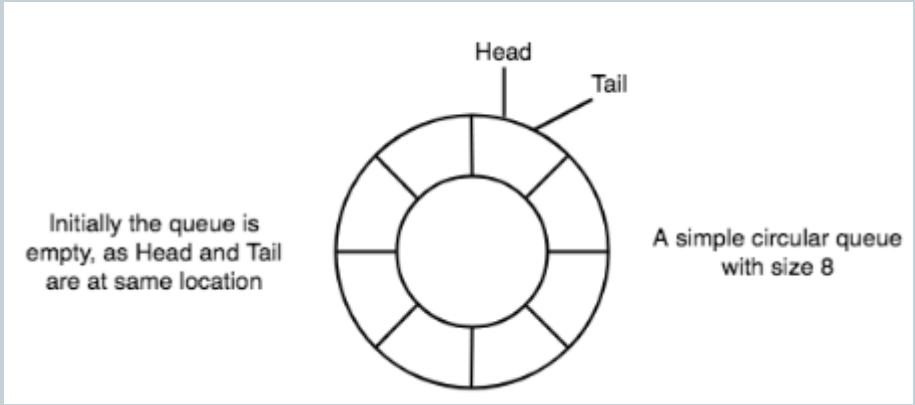
- When we **dequeue** any element to remove it from the queue, we are actually moving the **front** of the queue forward, thereby reducing the overall size of the queue. And we cannot insert new elements, because the **rear** pointer is still at the end of the queue



- The only way is to reset the linear queue, for a fresh start

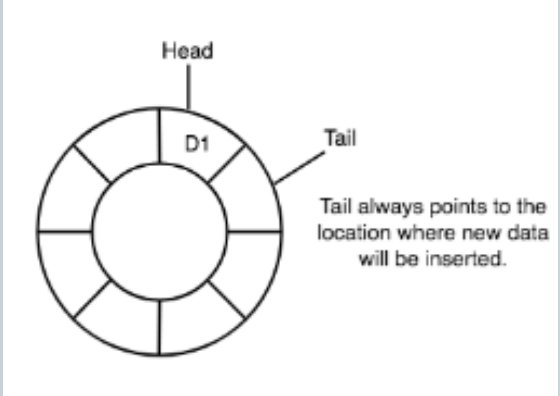
Circular Queue ...

- **Circular Queue** is also a linear data structure, which follows the principle of **FIFO**(First In First Out), but instead of ending the queue at the last position, it again starts from the first position after the last, hence making the queue behave like a circular data structure. It is also called **'Ring Buffer'**.
- Basic features of Circular Queue
  1. In case of a circular queue, head pointer will always point to the front of the queue, and tail pointer will always point to the end of the queue.
  2. Initially, the head and the tail pointers will be pointing to the same location, this would mean that the queue is empty

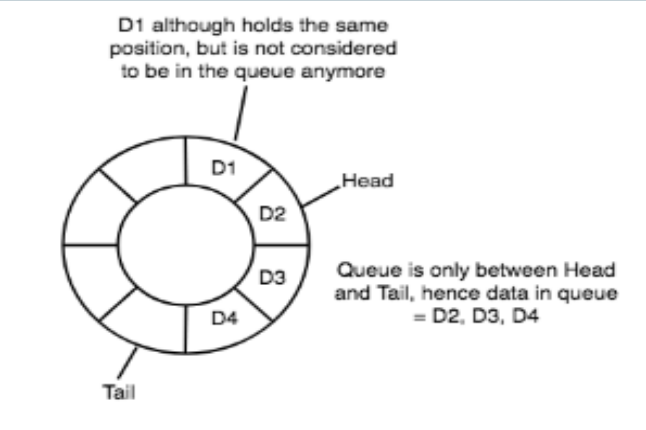


Circular Queue ...

3. New data is always added to the location pointed by the tail pointer, and once the data is added, tail pointer is incremented to point to the next available location

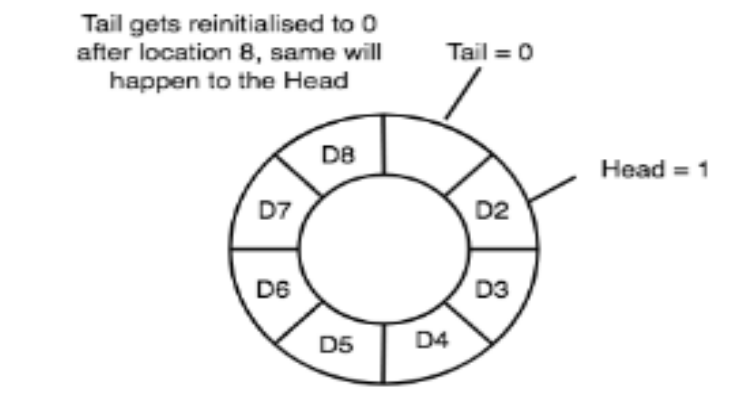


4. In a circular queue, data is not actually removed from the queue. Only the head pointer is incremented by one position when **dequeue** is executed. As the queue data is only the data between head and tail, hence the data left outside is not a part of the queue anymore, hence removed.

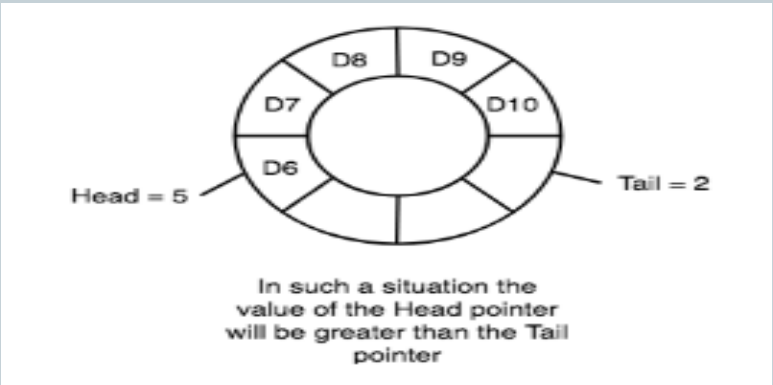


Circular Queue

5. The head and the tail pointer will get reinitialized to **0** every time they reach the end of the queue



6. Also, the head and the tail pointers can cross each other. In other words, head pointer can be greater than the tail. Sounds odd? This will happen when we dequeue the queue a couple of times and the tail pointer gets re initialized upon reaching the end of the queue.

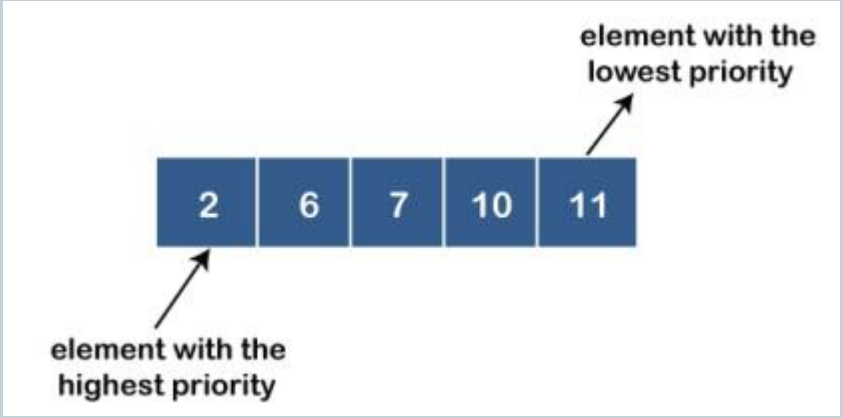


Priority Queue

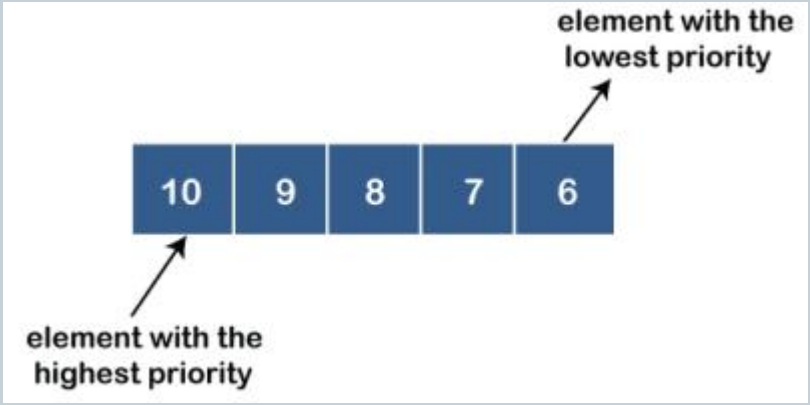
- A priority queue is an abstract data type that behaves similarly to the normal queue except that each element has some priority, i.e., the element with the highest priority would come first in a priority queue. The priority of the elements in a priority queue will determine the order in which elements are removed from the priority queue.
- The priority queue supports only comparable elements, which means that the elements are either arranged in an ascending or descending order.
- For example, suppose we have some values like 1, 3, 4, 8, 14, 22 inserted in a priority queue with an ordering imposed on the values is from least to the greatest. Therefore, the 1 number would be having the highest priority while 22 will be having the lowest priority

Priority Queue

- Ascending order priority queue
  - In ascending order priority queue, a lower priority number is given as a higher priority in a priority.
  - For example, we take the numbers from 1 to 5 arranged in an ascending order like 1,2,3,4,5; therefore, the smallest number, i.e., 1 is given as the highest priority in a priority queue.



- Descending order priority queue
  - In descending order priority queue, a higher priority number is given as a higher priority in a priority.
  - For example, we take the numbers from 1 to 5 arranged in descending order like 5, 4, 3, 2, 1; therefore, the largest number, i.e., 5 is given as the highest priority in a priority



Applications of Queues

- It is used to schedule the jobs to be processed by the CPU.
- When multiple users send print jobs to a printer, each printing job is kept in the printing queue. Then the printer prints those jobs according to first in first out (FIFO) basis.
- Breadth first search uses a queue data structure to find an element from a graph.
- Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
- In real life scenario, Call Center phone systems uses Queues to hold people calling them in an order, until a service representative is free.
- Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive i.e First come first served
- real life, Call Center phone systems will use Queues, to hold people calling them in an order, until a service representative is free.
- Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive, First come first served

Further Readings

- Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice Hall, 1988.
- Data Structures and Algorithms; Shi-Kuo Chang; World Scientifi c.
- Data Structures and Effi cient Algorithms, Burkhard Monien, Thomas Ottmann, Springer.
- Kruse Data Structure & Program Design, Prentice Hall of India, New Delhi
- Mark Allen Weles: Data Structure & Algorithm Analysis in C Second Adition. Addison-Wesley publishing
- RG Dromey, How to Solve it by Computer, Cambridge University Press.
- Shi-kuo Chang, Data Structures and Algorithms, World Scientifi c
- Sorenson and Tremblay: An Introduction to Data Structure with Algorithms.
- Thomas H. Cormen, Charles E, Leiserson & Ronald L. Rivest: Introduction to Algorithms.
- Prentice-Hall of India Pvt. Limited, New Delhi Timothy A. Budd, Classic Data Structures in C++, Addison Wesley