Queue

Agenda

- Need of Queue
- Introduction to queue
- Queue: ADT
- Queue: Example
- Queue implementation
- Queue implementation using Array
- Circular Queue
- Circular Queue Implementation using Array
- Application: Sorting of Queue elements
- Application of Queue
- Summary

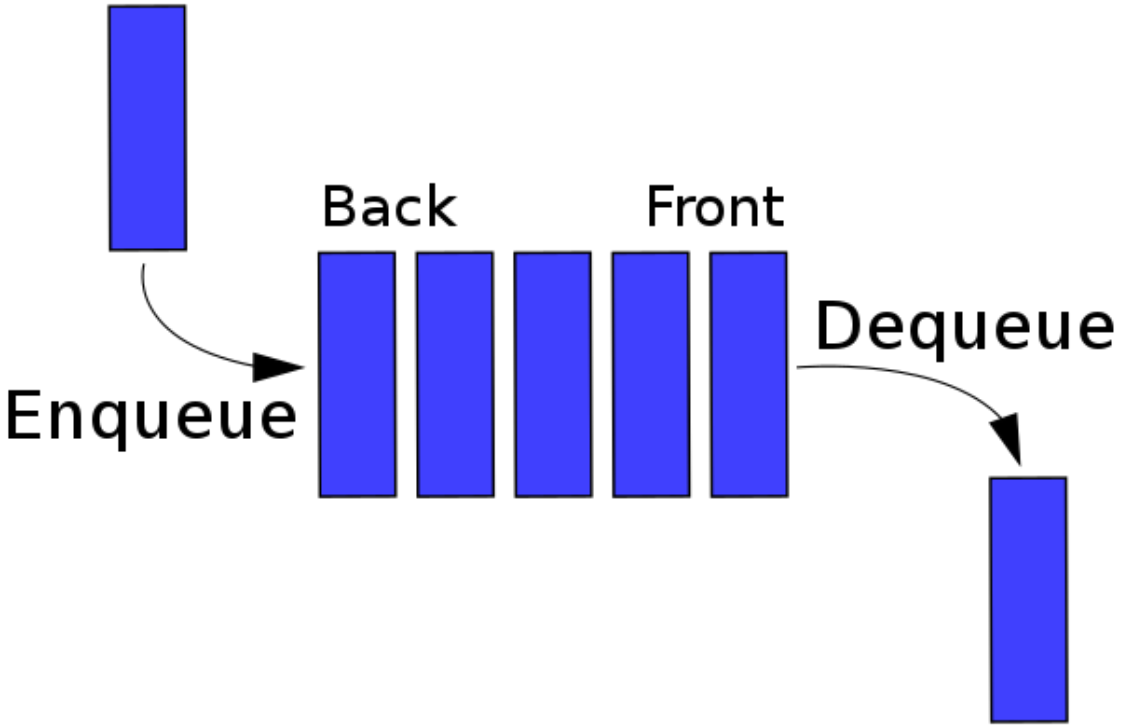
Need of Queue

- Consider an application that has certain requirements mentioned below:
  - Application is serving multiple users
  - Every user is generating request that requires processing
  - Only one request is executed at a time
  - Request that arrived first should be executed first
  - However;
    - Rate of execution might be slower than the rate of request arrival
    - Therefore you need to store the requests until they are processed
  - How to solve this problem?
    - Request should be stored in a way so that order of retrieval is similar to order of arrival
    - What Data Structure is appropriate?
    - Stack?

Introduction to queue

- A queue works on the principle of *first-in-first-out (FIFO)*. The object/data that is added at first will be deleted first.
- Data insertion and deletion in queue will happen from two different ends only. The variable that is used to maintain these positions are known as *front* and *rear/back.*
- Rear is used to perform insertion whereas Front is used to perform deletion.
- **Operations:**
  - **enque:** Insertion in queue is known as enque operation.
  - **deque:** Deletion in queue is known deque operation.

Introduction to queue



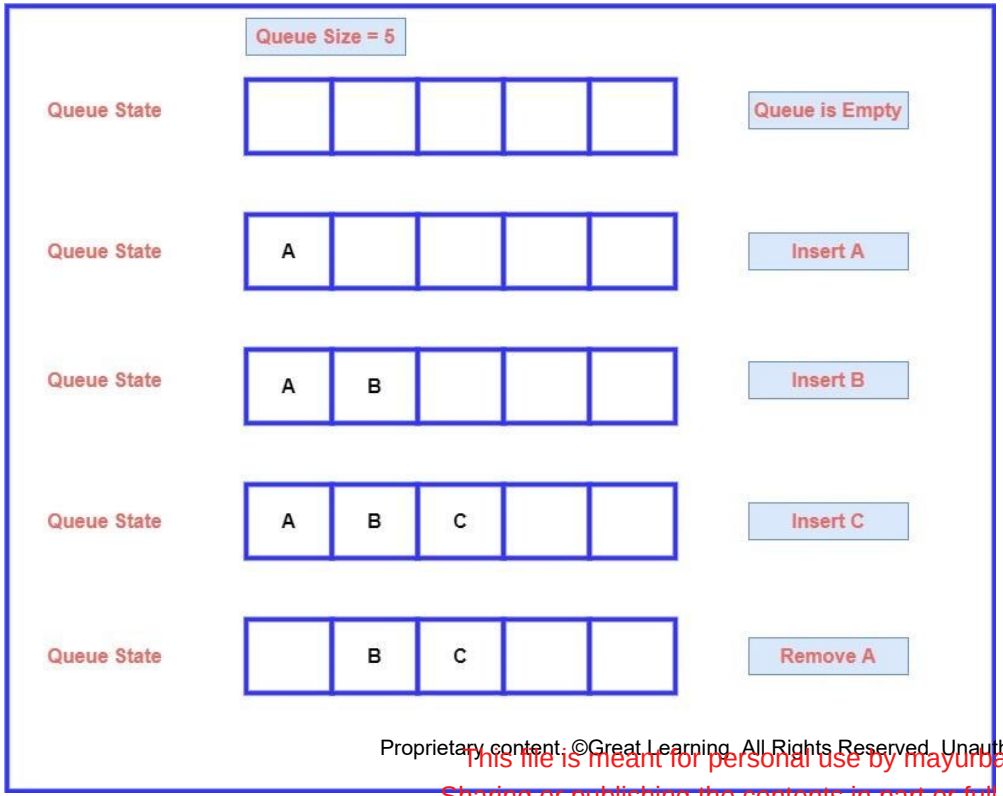
Queue ADT

- Generic operations on Queue data structure.
  - **new():** – Create new queue Q.
  - **enque():** - All the elements will be inserted using rear.
  - **deque():** - Element deletion from the queue using front.
  - **getfront():** – Returns the element that shall be deleted because of deque operation
  - **getrear():** Returns the element that was last inserted in the queue
- The following support methods could also be defined:
  - **size():** - Returning the number of elements in the queue.
  - **isEmpty():** - return true if there are no elements in the queue.

Queue example

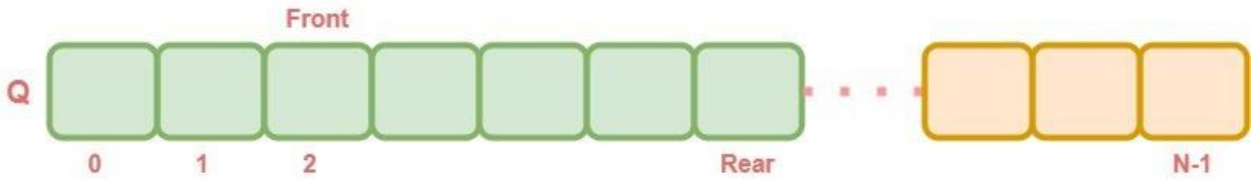
- Let’s have a look at one of the common example known as spooling:
  - Insert means; inserting a new job for printing.
  - Delete means; removing a job once the job is completed.
  - Lets assume that at maximum 5 requests can be stored at any time
  - Task A, B, C …. are supposed to arrive for execution.
  - Tasks that arrive first should be printed first and others will wait for there turn.

Queue example



Queue implementation using Array

- Implementation of queue using Array or python lists. Below are the operations:
  - Creating an array Q of size N. This means that we can not have more than N elements in array.
  - Two variables are declared named as Front and REAR and set to -1.
  - For enque operation Rear will be incremented
  - For deque operation Front will be incremented



Queue implementation using Array

```
# Custom Queue implementation in Python
class Queue:

  # Initialize queue
  def __init__(self, size):
    self.q = [None] * size
    self.capacity = size
    self.front = -1
    self.rear = -1

  def enque(self, value):
  # check for queue overflow
    if self.rear + 1 == self.capacity:
      print("OverFlow!! Queue is Full.")
      return

    print("Inserting element...", value)
    #setting front for the first inerstion
    if self.front == -1:
      self.front = 0
    # insertion other than first elelemnt
    self.rear = (self.rear + 1)
    self.q[self.rear] = value
```

```
# This function prints all te values available in
queue

    def show(self):
        if self.isempty():
            print("Queue is empty.")
        else:
            print("FRONT")
            i = self.front
            while i <= self.rear:
                print(self.q[i])
                i += 1
            print("REAR")
```

Queue implementation using Array

```
def dequeue(self):
    if self.isempty():
        return -1
    else:
        x = self.q[self.front]
        self.front = self.front + 1
        return x

def isempty(self):
    if (self.front == -1):
        return True
    return False
```

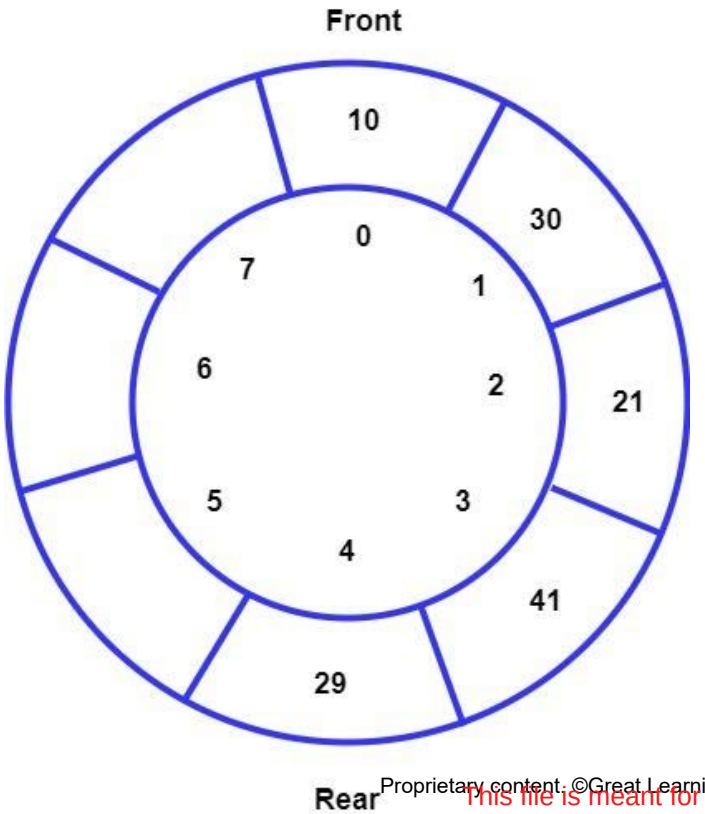
```
def getfront(self):
    if self.isempty():
        print("Queue is empty.")
    else:
        print("Front element of queue is :", end=" ")
        print(self.q[self.front])

def getrear(self):
    if self.isempty():
        print("Queue is empty.")
    else:
        print("Rear element of queue is :", end=" ")
        print(self.q[self.rear])
```

Circular Queue

- Limitations of general queue
  - Memory inefficient
    - Once the element is deleted and front is pointing to next elements
    - The space available can not be consumed and may result in inefficient memory consumption
    - How to resolve this?

Circular Queue



Circular Queue implementation using Array

```
# Circular Queue implementation in Python
class CircularQueue:

  # Initialize queue
  def __init__(self, size):
    self.q = [None] * size
    self.capacity = size
    self.front = -1
    self.rear = -1

  def enque(self, value):
  # check for queue overflow
    if ((self.rear + 1 % self.capacity) == self.front):
      print("OverFlow!! Circular Queue is Full.")
      return

    print("Inserting element...", value)
    elif self.front == -1:
      self.front = 0
      self.rear = 0
      self.q[self.rear] = value
    elif:
      self.rear = (self.rear + 1) %self.capacity
      self.q[self.rear] = value
```

```
def show(self):
  if self.isempty():
    print("Queue is empty.")
  print("FRONT")
  elif (self.rear > self.front):
    i = self.front
    while i <= self.rear:
      print(self.q[i])
      i += 1
  else:
    i = self.front
    while(i<self.size):
      print(self.q[i])
      i += 1
    i = 0
    while(i<self.rear):
      print(self.q[i])
      i += 1

  print("REAR")
```

Circular Queue implementation using Array

```
def dequeue(self):
    if self.isempty():
        return -1
    elif (self.front == self.rear):

        x = self.q[self.front]
        self.front = -1
        self.rear = -1
        return x
    else:
        x = self.q[self.front]
        self.front = (self.front+1) % self.size
        return x

def isempty(self):
    if (self.front == -1):
        return True
    return False
```

```
def getfront(self):
    if self.isempty():
        print("Queue is empty.")
    else:
        print("Front element of queue is :", end=" ")
        print(self.q[self.front])

def getrear(self):
    if self.isempty():
        print("Queue is empty.")
    else:
        print("Rear element of queue is :", end=" ")
        print(self.q[self.rear])
```

Queue implementation using Array

- Let’s do the analysis of operations on Array implementation of stack:
  - **enque():** constant time; insertion always using rear.
  - **deque():** constant time; deletion always using front.
  - **getfront():** constant time; returning front element
  - **getrear():** constant time; returning rear element

Queue applications

- Printer Spooling
- E-mail services
- Schedulers in CPU
- Buffer feature in Keyboards

Queue: Covid Infection example

- Let's assume a scenario where data of covid infected patients are stored in the following way in an array:
  - Assume there is a 2D-array with name : A[n][n]
  - if person i is infected by person j;
    - In this case A[i][j] in the array will show value 1
    - Else A[i][j] will contain 0

- Problem statement
    - Find all the patients infected from a particular person directly or indirectly through intermediate persons.

Queue: Covid Infection example

	0	1	2	3	4	5	6	7	8	9	10
0	0	1	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	1	1	0	0	0	0	0
3	0	0	0	0	0	0	0	1	1	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0

Use of Queue

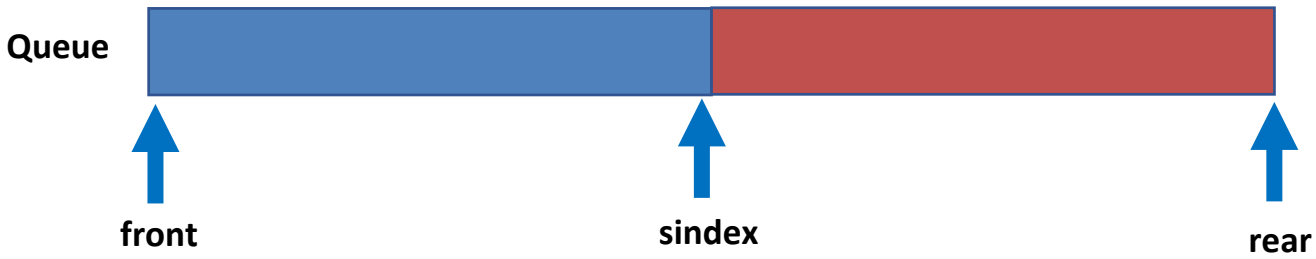
```
def find_infected_people(u, data,n):
        Infected_people=Array[n]
        #Create a queue
        q=Queue()
        #add u in the Queue
        q.enque(u);
        while(!q.empty()):
        e=q.deque()
        Infected_poeple[e]=1
        for i in range(n):
            If data[e][i] ==1 and Infected_people[i]==0:
                q.enque(i)
Return Infected_people
```

Example: Sorting Queue Elements

- Given a queue with its elements in no particular order, sort queue elements without using any extra space.
    - Input: 2, 6, 3, 9, 10, 7 ; front element= 2, rear element= 7
    - Output: 2, 3, 6, 7, 9, 10 ; front element = 2, rear element = 10

Example: Sorting Queue Elements

- **Algorithm**



Property: Elements after sindex are sorted.

Example: Sorting Queue Elements

- **Algorithm**
  1. sindex = rear index of the queue
  2. mindex = find the index of the minimum element from front to sindex.
  3. minimum = Store the value of mindex element
  4. Perform Enque and Deque for each of the elements in the Queue except for the mindex, for which perform only Deque.
  5. Enque the minimum element.
  6. Perform steps 2 to 5 until all the elements get processed, i.e., front = sindex+1

Example: Sorting Queue Elements

18	9	33	6	11	7	8	10
18	9	33	11	7	8	10	6
18	9	33	11	8	10	6	7
18	9	33	11	10	6	7	8
18	33	11	10	6	7	8	9
18	33	11	6	7	8	9	10
18	33	6	7	8	9	10	11
33	6	7	8	9	10	11	18
6	7	8	9	10	11	18	33

Summary

- Understood basics of Queue Data structure
- Applications of queue
- Implementation of simple and circular queue
- Examples
  - Covid infection example
  - Sorting of queue elements