4.2.2 Queues

Queue

Definition

A queue is a data structure that follows the FIFO principle (First In, First Out).

Operations

Enqueue

- Check if the queue is full
- If not, insert an item at the rear of the queue
- If it is full, raise an error message

Dequeue

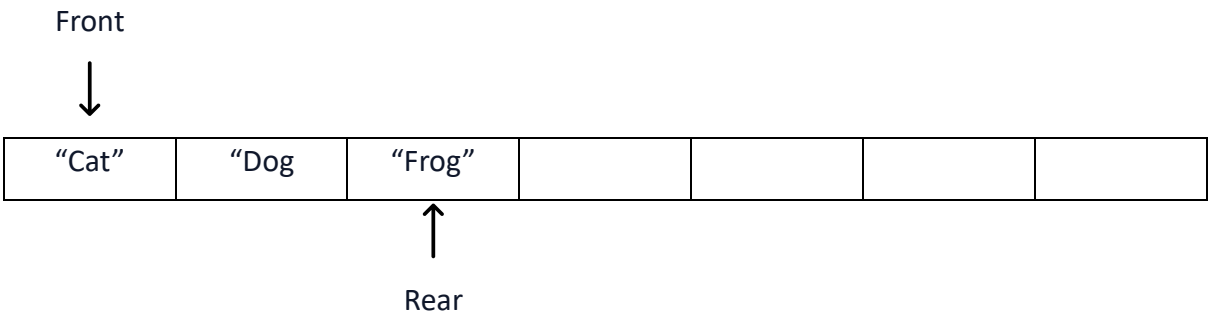
- Check if the queue is empty
- If not, return the item from the front of the queue
- If it is empty, raise an error message.

Implementation

One way of implementing a queue is:

- Using a 1D array.
- Using 2 pointers: front and rear.

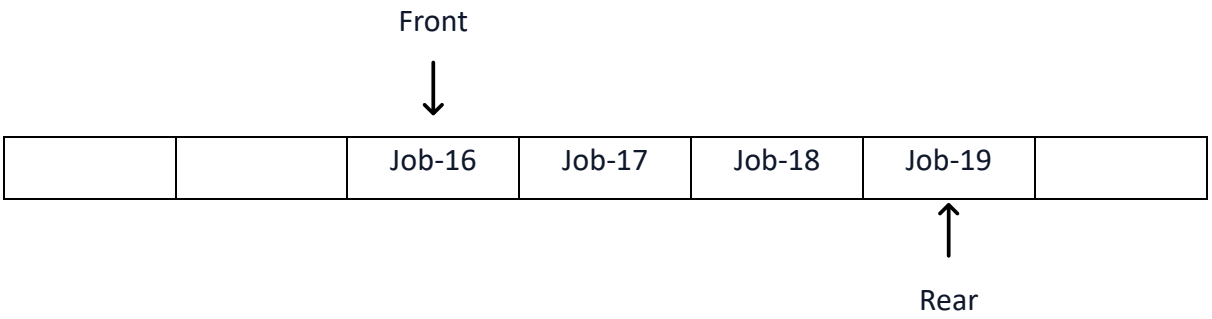
The **front pointer** stores the position of the next item to be removed.
The **rear pointer** stores the position of the most recently added item.



Linear Queue

In a linear queue, the queue is treated as full when the rear pointer reaches the last index of the array, even if earlier positions have become empty after dequeuing.

Any free space created at the beginning of the array after an item is dequeued is either wasted, or time is taken to shift the items along.



Example Code

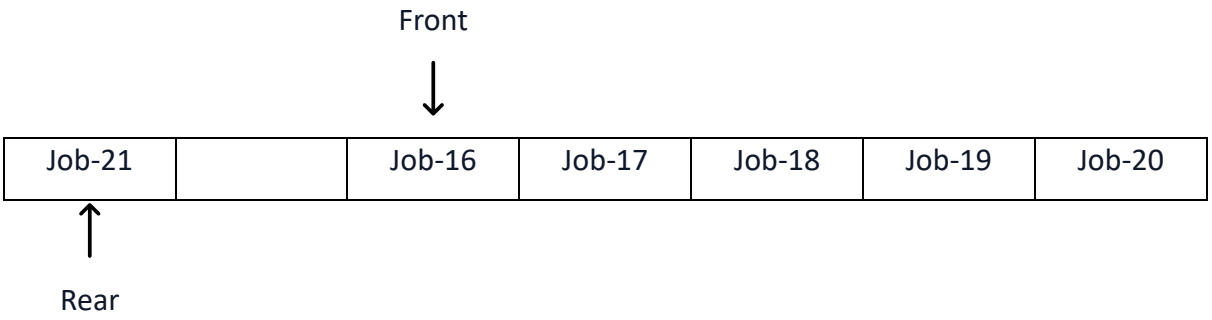
```
Enqueue(item)
      if Rear == array.length – 1:
            print(“Queue is full”)
      else:
            Rear = Rear + 1
            array[Rear] = item


Dequeue()
      if front == Rear + 1:
            print(“Queue is empty”)
            return -1
      else:
            item = array[Front]
            Front = Front + 1
            return item
```

Circular Queue

A circular queue improves on a linear queue by having pointers loop back around to the front of the array after they reach the end. This allows space at the beginning of the queue to be reused without having to shift the items along.

In a circular queue, an extra count variable or one unused array position is often used so that the algorithm can distinguish between a full queue and an empty queue.



Example Code

```
Enqueue(item)
     if item_count == array.length:
            print("Queue is full")
      else:
            Rear = (Rear + 1) MOD array.length
            array[Rear] = item
            item_count = item_count + 1


Dequeue()
      if item_count == 0:
            print("Queue is empty")
            return -1
      else:
            item = array[Front]
            Front = (Front + 1) MOD array.length
            item_count = item_count - 1
            return item
```

Priority Queue

In a priority queue, a new item is inserted according to its priority rather than simply being added to the rear. Items are placed so that higher-priority items are nearer the front of the queue, while items with the same priority remain in the order they were added.

While enqueuing is more complex, dequeuing works in a similar manner to the other queues.

Front

Job-12	Job - 14	Job-13	Job-16	Job-17	Job-15	
Priority A	Priority A	Priority B	Priority B	Priority B	Priority C	

Rear

Enqueuing an item:

Starting with the item at the rear of the queue each item is moved back one place in the array.

This is done until the start of the queue is reached or an item with the same OR higher priority than the item to add is found.

The new item is added in the position before that item.