

Data Structures and Algorithms

Dr. Farshid Mehrdoust
Kiarash Dadpour

University of Guilan

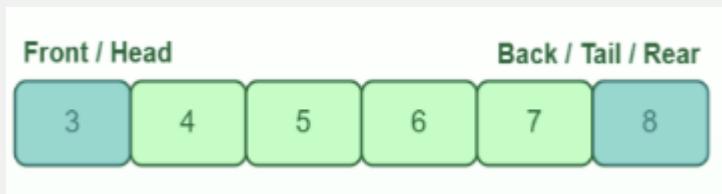
04

Queues



Queue

Queue is a linear data structure that follows the **FIFO (First In, First Out)** rule. The first element that enters the queue is the first one to be removed. In other words elements can be inserted at any time, but only the element that has been in the queue the longest can be next removed.



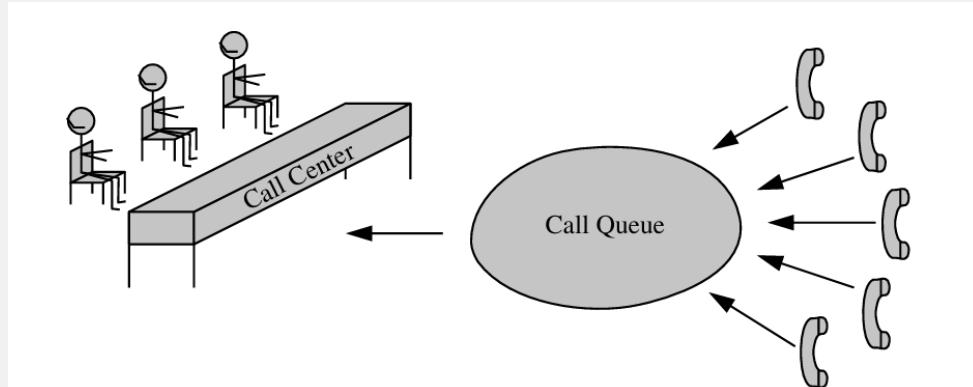
Examples

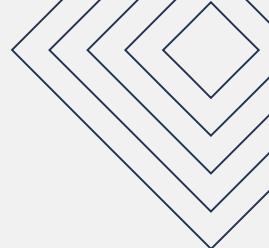
1. Customer Service Call Center

A queue is used to manage incoming customer calls. Calls are answered in the order they are received, following the *FIFO* principle. This ensures fairness—customers who have been waiting longer are served first.

2. Restaurant Wait-List

Restaurants often maintain a waiting list for guests. As people arrive, their names are added to the end of the list, and the next available table is given to the person who has been waiting the longest. This is another clear example of *FIFO* behavior.





Queue Abstract Data Type

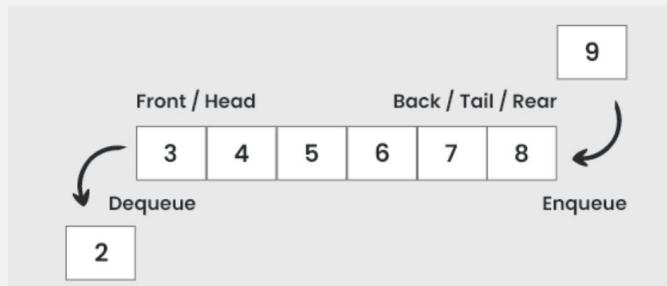
Q.enqueue(e): Add element e to the back of queue Q.

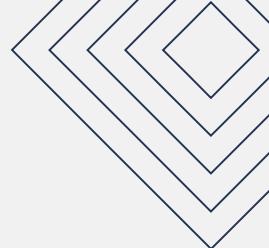
Q.dequeue(): Remove and return the first element from queue Q; an error occurs if the queue is empty.

Q.first(): Return a reference to the element at the front of queue Q, without removing it; an error occurs if the queue is empty.

Q.is empty(): Return True if queue Q does not contain any elements.

Q.len(): Return the number of elements in queue Q





Enqueue

Dequeue

Enqueue

```
def enqueue(self, item):
    if self.size == self.max_size:
        raise Exception("Queue overflow.")
    self.Q[(self.front_index + self.size) % self.max_size] = item
    self.size += 1
```

Dequeue

```
def dequeue(self):
    if self.is_empty():
        raise Exception("Queue is Empty!")
    answer = self.Q[self.front_index]
    self.Q[self.front_index] = None
    self.size -= 1
    self.front_index = (self.front_index + 1) % (self.max_size)
    return answer
```

Example

Operation	Return Value	$\text{first} \leftarrow Q \leftarrow \text{last}$
<code>Q.enqueue(5)</code>		
<code>Q.enqueue(3)</code>		
<code>len(Q)</code>		
<code>Q.dequeue()</code>		
<code>Q.is_empty()</code>		
<code>Q.dequeue()</code>		
<code>Q.is_empty()</code>		
<code>Q.dequeue()</code>		
<code>Q.enqueue(7)</code>		
<code>Q.enqueue(9)</code>		
<code>Q.first()</code>		
<code>Q.enqueue(4)</code>		
<code>len(Q)</code>		
<code>Q.dequeue()</code>		

Example

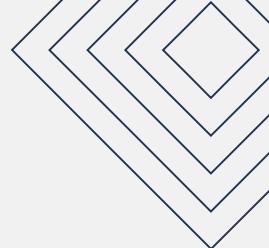
Operation	Return Value	$\text{first} \leftarrow Q \leftarrow \text{last}$
<code>Q.enqueue(5)</code>	—	[5]
<code>Q.enqueue(3)</code>	—	[5, 3]
<code>len(Q)</code>	2	[5, 3]
<code>Q.dequeue()</code>	5	[3]
<code>Q.is_empty()</code>	False	[3]
<code>Q.dequeue()</code>	3	[]
<code>Q.is_empty()</code>	True	[]
<code>Q.dequeue()</code>	“error”	[]
<code>Q.enqueue(7)</code>	—	[7]
<code>Q.enqueue(9)</code>	—	[7, 9]
<code>Q.first()</code>	7	[7, 9]
<code>Q.enqueue(4)</code>	—	[7, 9, 4]
<code>len(Q)</code>	3	[7, 9, 4]
<code>Q.dequeue()</code>	7	[9, 4]

Analyzing the Array-Based Queue Implementation

Operation	Running Time
<code>Q.enqueue(e)</code>	
<code>Q.dequeue()</code>	
<code>Q.first()</code>	
<code>Q.is_empty()</code>	
<code>len(Q)</code>	

Analyzing the Array-Based Queue Implementation

Operation	Running Time
<code>Q.enqueue(e)</code>	$O(1)^*$
<code>Q.dequeue()</code>	$O(1)^*$
<code>Q.first()</code>	$O(1)$
<code>Q.is_empty()</code>	$O(1)$
<code>len(Q)</code>	$O(1)$



References

- 1) Fundamentals of Python: Data Structures / Kenneth A. Lambert
- 2) Introduction to algorithms / Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.
- 3) Data Structures and Algorithms in Python / Michael T. Goodrich, Robert Tamassia, Michael H. Goldwasser