Queue (Implementation)

Lets look at the basic functionality and implementation of queues in Python!

We'll cover the following

Implementation of Queues

Queues are implemented in many ways. They can be represented by using lists, Linked Lists, or even Stacks. But most commonly lists are used as the easiest way to implement Queues. As discussed in the previous lesson, a typical Queue must contain the following standard methods:

- `enqueue(element)`
- `dequeue()`
- `is_empty()`
- `front()`
- `back()`

We will take a look at these functions individually, but, before that, let's construct a class of Queue and create an object. The class will consist of the list that holds all the elements in the queue and the relevant functions. The code given below shows how to construct a Queue class.

Queue.py

```python
class MyQueue:
    def __init__(self):
        self.queue_list = []


queue = MyQueue()

```

Adding Helper Functions

Now, before adding the `enqueue(element)` and `dequeue()` functions into this class, we need to implement some helper functions to keep the code simple and understandable. Here's the list of the helper functions that we will implement in the code below:

- `is_empty()`

- `front()`
- `back()`
- `size()`

Queue.py

```python
class MyQueue:
    def __init__(self):
        self.queue_list = []

    def is_empty(self):
        return len(self.queue_list) == 0

    def front(self):
        if self.is_empty():
            return None
        return self.queue_list[0]

    def back(self):
        if self.is_empty():
            return None
        return self.queue_list[-1]

    def size(self):
        return len(self.queue_list)


queue = MyQueue()
print("is_empty(): " + str(queue.is_empty()))
print("front(): " + str(queue.front()))
print("back(): " + str(queue.back()))
print("size(): " + str(queue.size()))
```

If your output returns `True` for `is_empty()` and `None` for `front()`, then this means that these helper functions are working correctly. We consider the last element of the list to be the back (which means we will enqueue elements here!) and the first element (the zeroth element) to be the front so we will dequeue elements from here. You can also do the opposite - this is just how we are implementing queues in this course. Can you think of use cases where picking a particular front or a back would optimize the time complexity?

Now, examine the following extended code with the `enqueue(element)` and `dequeue()` functions added to the `MyQueue` class. We have created an object of the `MyQueue` class and will try to add and remove some elements from this queue by using these two functions. Let’s try!

Queue.py

```python
class MyQueue:
    def __init__(self):
        self.queue_list = []

    def is_empty(self):
        return self.size() == 0

    def front(self):
        if self.is_empty():
            return None
        return self.queue_list[0]

    def back(self):
        if self.is_empty():
            return None
        return self.queue_list[-1]

    def size(self):
        return len(self.queue_list)

    def enqueue(self, value):
        self.queue_list.append(value)

    def dequeue(self):
        if self.is_empty():
            return None
        front = self.front()
        self.queue_list.remove(self.front())
        return front


```

If you look at the output of the code, you can see that the elements are enqueued in the back and dequeued from the front. This means that our queue works perfectly. Congratulations, you have now successfully implemented Queue using Lists!

Complexities of Queue Operations

Let’s look at the time complexity of each queue operation.

Operation	**Time Complexity**
is_empty	O(1)
front	O(1)
back	O(1)
size	O(1)
enqueue	O(1)
dequeue	O(n)

Note: Here we have implemented queue using python list. However, if the queue is implemented using a Linked list, the time complexity can be optimized to $O(1)$.

In the last few chapters, we discussed some fundamental data structures and roughly went through their implementations.

Now lets try some challenges that use your knowledge of stacks and queues. After that, we will take a look at some advanced data structures which are derived using the basic data structures that we have studied so far.

Back

What is a Queue?

Next

Challenge 1: Generate Binary Number...

✓ Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/queue-implementation__introduction-to-stackqueues__data-structures-for-coding-interviews-in-python)