


Theory: Queue in Python

 29 minutes

8 / 8 problems solved

Start practicing

1061 users solved this topic. Latest completion was 2 days ago.

§1. Queue

Queue is a linear data structure that implements the first-in-first-out (FIFO) principle, meaning that new elements can be added to the back of the queue, while removing an element is only possible from its head. Queue is really just like any real-life line in a supermarket: customers who come first are attended first, while the others are waiting for their turn.

Typically queue supports the following two basic operations:

- **enqueue**, that adds a new element to the back of the queue
- **dequeue**, that removes the first element of the queue.

Sometimes, operations **front** and **rear** are also supported. They return the first and the last element of the queue respectively without removing them.

In practice, queues are especially useful in threaded programming when information must be exchanged between multiple threads. Queues are at the heart of job schedulers, helping to schedule delayed tasks efficiently while preserving the order in which they must be performed.

In this topic, you will get familiar with a data structure called queue and how to implement it in Python.

§2. Deque

Before we learn how to implement queue in Python, we should get familiar with one more data structure that is closely related, namely **deque**. Deque stands for *'double-ended queue'* and allows for the elements to be added and removed from either side.

In Python, you can find deque implementation in the `collections` module. The essential methods are `append(element)` and `appendleft(element)`, which add a new element to the right or left side of the deque respectively, as well as `pop()` and `popleft()`, which remove the first element from the corresponding side of the deque.

Let's take a look at the example:

Current topic:

✓

[Queue in Python](#)

...

Topic depends on:

✓

[Deque](#)

...

✓

[Load module](#)

Stage 1

5★

...

✓

[Algorithms in Python](#)

...

```
1  from collections import deque
2
3  my_deque = deque()
4
5  my_deque.append('Middle')
6  my_deque.append('Right')
7
8  my_deque.appendleft('Left')
9
10 print(my_deque)
11
12
13 # deque(['Left', 'Middle', 'Right'])
14
15
16 print(my_deque.pop())
17
18
19 # Right
20
21
22 print(my_deque)
23
24
25 # deque(['Left', 'Middle'])
26
27
28 print(my_deque.popleft())
29
30
31
32 # Left
33
34
35 print(my_deque)
36
37
38 # deque(['Middle'])
```

Deque is implemented as a doubly linked list, meaning that its elements aren't stored next to each other in memory like the elements of the conventional list. By contrast, only smaller chunks of elements are stored together, and each such chunk stores the references to the previous chunk and to the next one. This architecture enables quicker append and pop operations from both ends of the deque.

Constant time for append and pop operations from both ends is achieved by storing pointers to the both ends of the doubly linked list.

§3. Using `collections.deque()` as a queue

Of course, deque can be used as a simple queue. You just need to ignore the fact that you can add elements to the head and remove them from the back.

Here is an example:

```
1 my_queue = deque()
2
3 my_queue.appendleft('First')
4 my_queue.appendleft('In')
5 my_queue.appendleft('First')
6 my_queue.appendleft('Out')
7
8 print(my_queue)
9
1
0 # Deque(['Out', 'First', 'In', 'First'])
1
1
1
2 print(my_queue.pop())
1
3
1
4 # First
1
5
1
6 print(my_queue.pop())
1
7
1
8 # In
1
9
2
0 print(my_queue.pop())
2
1
2
2 # First
2
3
2
4 print(my_queue.pop())
2
5
2
6 # Out
```

So, as you can see, we get the elements in the same order as we put them in the queue. Note that if you call `pop()` one more time, you will get an exception because there are no more elements in the queue to be popped:

```
1 my_queue.pop()
2
3 # IndexError: pop from an empty deque
```

To avoid this, you can always check if the queue is non-empty before trying to get the next element from it. This can be done with the help of the `len()` method:

```
1 len(my_queue)
2
3 # 0
4
5 my_queue.append('a')
6 len(my_queue)
7
8 # 1
```

§4. Conclusions

- Queue is a first-in-first-out data structure.
- Queue is at the heart of the job scheduling in parallel computing.
- Deque stands for a double-ended queue. You can add and retrieve elements to/from both sides of the deque.
- In Python, deque is implemented in the `collections` module.
- You can use deque as a queue.

[1 Queue in Python](#)

[§1. Queue](#)

[§2. Deque](#)

[§3. Using collections.deque\(\) as a queue](#)

[§4. Conclusions](#)

[Feedback & Comments](#)

87 users liked this theory. 0 didn't like it. What about you?



Start practicing

[Comments \(7\)](#)[Hints \(0\)](#)[Useful links \(0\)](#)[Show discussion](#)