Build a queue using an array

May 3, 2020

1 Implement a queue using an array

In this notebook, we'll look at one way to implement a queue by using an array. First, check out the walkthrough for an overview of the concepts, and then we'll take a look at the code.

Walkthrough

OK, so those are the characteristics of a queue, but how would we implement those characteristics using an array?

Walkthrough

What happens when we run out of space in the array? This is one of the trickier things we'll need to handle with our code.

Walkthrough

1.1 Functionality

Once implemented, our queue will need to have the following functionality: 1. `enqueue` - adds data to the back of the queue 2. `dequeue` - removes data from the front of the queue 3. `front` - returns the element at the front of the queue 4. `size` - returns the number of elements present in the queue 5. `is_empty` - returns `True` if there are no elements in the queue, and `False` otherwise 6. `_handle_full_capacity` - increases the capacity of the array, for cases in which the queue would otherwise overflow

Also, if the queue is empty, `dequeue` and `front` operations should return `None`.

1.2 1. Create the `queue` class and its `__init__` method

First, have a look at the walkthrough:

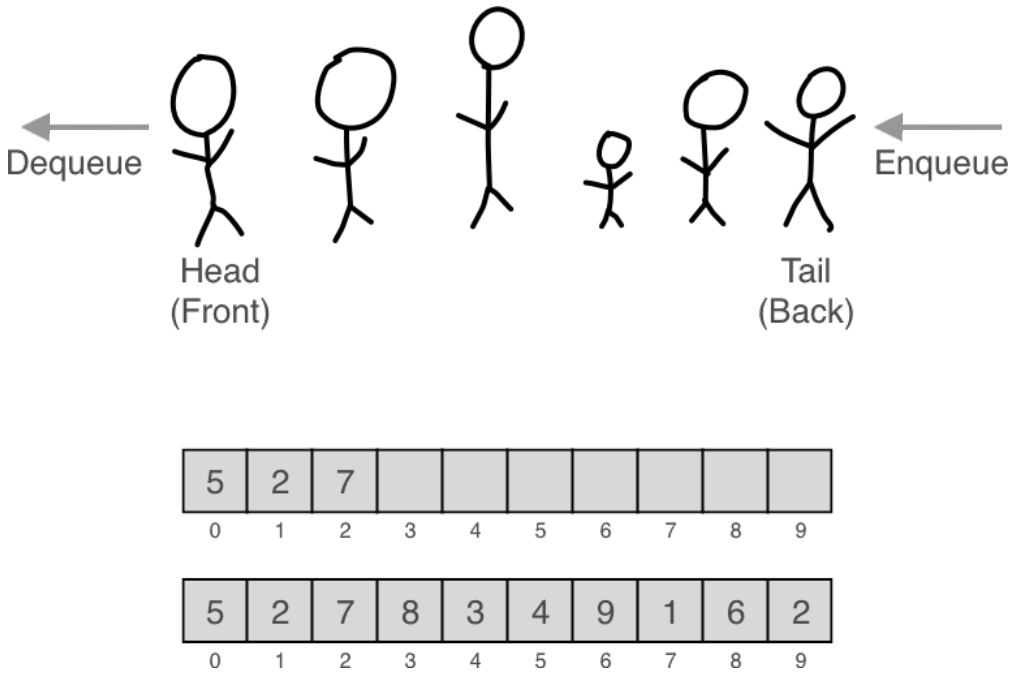
Walkthrough

```
In [ ]:
```

Now give it a try for yourself. In the cell below: * Define a class named `Queue` and add the `__init__` method * Initialize the `arr` attribute with an array containing 10 elements, like this: `[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]` * Initialize the `next_index` attribute * Initialize the `front_index` attribute * Initialize the `queue_size` attribute

```
In [ ]:
```

Hide Solution



People waiting in line.

```
In [ ]: class Queue:

            def __init__(self, initial_size=10):
                self.arr = [0 for _ in range(initial_size)]
                self.next_index = 0
                self.front_index = -1
                self.queue_size = 0
```

Let's check that the array is being initialized correctly. We can create a `Queue` object and access the `arr` attribute, and we should see our ten-element array:

```
In [5]: q = Queue()
        print(q.arr)
        print("Pass" if q.arr == [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] else "Fail")
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Pass
```

1.3 2. Add the `enqueue` method

Walkthrough

```
In [ ]:
```

In the cell below, add the code for the enqueue method.

The method should: * Take a value as input and assign this value to the next free slot in the array * Increment `queue_size` * Increment `next_index` (this is where you'll need to use the modulo

operator %) * If the front index is `-1` (because the queue was empty), it should set the front index to `0`

```
In [ ]: class Queue:

            def __init__(self, initial_size=10):
                self.arr = [0 for _ in range(initial_size)]
                self.next_index = 0
                self.front_index = -1
                self.queue_size = 0

            # TODO: Add the enqueue method
```

Hide Solution

```
In [ ]: class Queue:

            def __init__(self, initial_size=10):
                self.arr = [0 for _ in range(initial_size)]
                self.next_index = 0
                self.front_index = -1
                self.queue_size = 0

            def enqueue(self, value):
                # enqueue new element
                self.arr[self.next_index] = value
                self.queue_size += 1
                self.next_index = (self.next_index + 1) % len(self.arr)
                if self.front_index == -1:
                    self.front_index = 0
```

1.4 3. Add the `size`, `is_empty`, and `front` methods

Just like with stacks, we need methods to keep track of the size of the queue and whether it is empty. We can also add a `front` method that returns the value of the front element. * Add a `size` method that returns the current size of the queue * Add an `is_empty` method that returns `True` if the queue is empty and `False` otherwise * Add a `front` method that returns the value for the front element (whatever item is located at the `front_index` position). If the queue is empty, the `front` method should return None.

```
In [ ]: class Queue:

            def __init__(self, initial_size=10):
                self.arr = [0 for _ in range(initial_size)]
                self.next_index = 0
                self.front_index = -1
                self.queue_size = 0

            def enqueue(self, value):
```

```
            # enqueue new element
            self.arr[self.next_index] = value
            self.queue_size += 1
            self.next_index = (self.next_index + 1) % len(self.arr)
            if self.front_index == -1:
                self.front_index = 0

        # TODO: Add the size method

        # TODO: Add the is_empty method

        # TODO: Add the front method
```

Hide Solution

```
In [ ]: class Queue:

        def __init__(self, initial_size=10):
            self.arr = [0 for _ in range(initial_size)]
            self.next_index = 0
            self.front_index = -1
            self.queue_size = 0

        def enqueue(self, value):
            # enqueue new element
            self.arr[self.next_index] = value
            self.queue_size += 1
            self.next_index = (self.next_index + 1) % len(self.arr)
            if self.front_index == -1:
                self.front_index = 0

        def size(self):
            return self.queue_size

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

        def front(self):
            # check if queue is empty
            if self.is_empty():
                return None
            return self.arr[self.front_index]
```

1.5 4. Add the `dequeue` method

Walkthrough

```
In [ ]:
```

In the cell below, see if you can add the `deqeueue` method.

Here's what it should do: * If the queue is empty, reset the `front_index` and `next_index` and then simply return `None`. Otherwise... * Get the value from the front of the queue and store this in a local variable (to `return` later) * Shift the `head` over so that it refers to the next index * Update the `queue_size` attribute * Return the value that was dequeued

```
In [ ]: class Queue:

            def __init__(self, initial_size=10):
                self.arr = [0 for _ in range(initial_size)]
                self.next_index = 0
                self.front_index = -1
                self.queue_size = 0

            def enqueue(self, value):
                # enqueue new element
                self.arr[self.next_index] = value
                self.queue_size += 1
                self.next_index = (self.next_index + 1) % len(self.arr)
                if self.front_index == -1:
                    self.front_index = 0

            # TODO: Add the dequeue method

            def size(self):
                return self.queue_size

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

            def front(self):
                # check if queue is empty
                if self.is_empty():
                    return None
                return self.arr[self.front_index]
```

Hide Solution

```
In [ ]: class Queue:

            def __init__(self, initial_size=10):
                self.arr = [0 for _ in range(initial_size)]
                self.next_index = 0
                self.front_index = -1
                self.queue_size = 0

            def enqueue(self, value):
                # enqueue new element
```

```
        self.arr[self.next_index] = value
        self.queue_size += 1
        self.next_index = (self.next_index + 1) % len(self.arr)
        if self.front_index == -1:
            self.front_index = 0

    def dequeue(self):
        # check if queue is empty
        if self.is_empty():
            self.front_index = -1   # resetting pointers
            self.next_index = 0
            return None

        # dequeue front element
        value = self.arr[self.front_index]
        self.front_index = (self.front_index + 1) % len(self.arr)
        self.queue_size -= 1
        return value

    def size(self):
        return self.queue_size

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

    def front(self):
        # check if queue is empty
        if self.is_empty():
            return None
        return self.arr[self.front_index]
```

1.6 5. Add the `_handle_queue_capacity_full` method

Walkthrough

```
In [ ]:
```

First, define the `_handle_queue_capacity_full` method: * Define an `old_arr` variable and assign the the current (full) array so that we have a copy of it * Create a new (larger) array and assign it to `arr`. * Iterate over the values in the old array and copy them to the new array. Remember that you'll need two `for` loops for this.

Then, in the `enqueue` method: * Add a conditional to check if the queue is full; if it is, call `_handle_queue_capacity_full`

```
In [ ]: class Queue:

    def __init__(self, initial_size=10):
        self.arr = [0 for _ in range(initial_size)]
        self.next_index = 0
```

```
        self.front_index = -1
        self.queue_size = 0

    def enqueue(self, value):
        # TODO: Check if the queue is full; if it is, call the _handle_queue_capacity_fu

        # enqueue new element
        self.arr[self.next_index] = value
        self.queue_size += 1
        self.next_index = (self.next_index + 1) % len(self.arr)
        if self.front_index == -1:
            self.front_index = 0

    def dequeue(self):
        # check if queue is empty
        if self.is_empty():
            self.front_index = -1   # resetting pointers
            self.next_index = 0
            return None

        # dequeue front element
        value = self.arr[self.front_index]
        self.front_index = (self.front_index + 1) % len(self.arr)
        self.queue_size -= 1
        return value

    def size(self):
        return self.queue_size

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

    def front(self):
        # check if queue is empty
        if self.is_empty():
            return None
        return self.arr[self.front_index]

    # TODO: Add the _handle_queue_capacity_full method
```

Hide Solution

```
In [ ]: class Queue:

    def __init__(self, initial_size=10):
        self.arr = [0 for _ in range(initial_size)]
        self.next_index = 0
        self.front_index = -1
```

```
        self.queue_size = 0

    def enqueue(self, value):
        # if queue is already full --> increase capacity
        if self.queue_size == len(self.arr):
            self._handle_queue_capacity_full()

        # enqueue new element
        self.arr[self.next_index] = value
        self.queue_size += 1
        self.next_index = (self.next_index + 1) % len(self.arr)
        if self.front_index == -1:
            self.front_index = 0

    def dequeue(self):
        # check if queue is empty
        if self.is_empty():
            self.front_index = -1   # resetting pointers
            self.next_index = 0
            return None

        # dequeue front element
        value = self.arr[self.front_index]
        self.front_index = (self.front_index + 1) % len(self.arr)
        self.queue_size -= 1
        return value

    def size(self):
        return self.queue_size

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

    def front(self):
        # check if queue is empty
        if self.is_empty():
            return None
        return self.arr[self.front_index]

    def _handle_queue_capacity_full(self):
        old_arr = self.arr
        self.arr = [0 for _ in range(2 * len(old_arr))]

        index = 0

        # copy all elements from front of queue (front-index) until end
        for i in range(self.front_index, len(old_arr)):
            self.arr[index] = old_arr[i]
```

```
                index += 1

            # case: when front-index is ahead of next index
            for i in range(0, self.front_index):
                self.arr[index] = old_arr[i]
                index += 1

            # reset pointers
            self.front_index = 0
            self.next_index = index
```

1.6.1 Test your queue

In [3]:
```
# Setup
q = Queue()
q.enqueue(1)
q.enqueue(2)
q.enqueue(3)

# Test size
print ("Pass" if (q.size() == 3) else "Fail")

# Test dequeue
print ("Pass" if (q.dequeue() == 1) else "Fail")

# Test enqueue
q.enqueue(4)
print ("Pass" if (q.dequeue() == 2) else "Fail")
print ("Pass" if (q.dequeue() == 3) else "Fail")
print ("Pass" if (q.dequeue() == 4) else "Fail")
q.enqueue(5)
print ("Pass" if (q.size() == 1) else "Fail")
```

```
Pass
Pass
Pass
Pass
Pass
Pass
```