CMPUT 175 - Lab 6: Queues

Submit by: Oct 26, 2020 *Demo in your lab starting: Oct 27, 2020*

Goal: Learn about bounded queues and circular queues

Exercise 1:

1. Download and save queues.py from eClass. This file contains the implementations for the Bounded Queue and the Circular Queue discussed in the lectures. It also contains the skeleton code for a number of tests to check various aspects of the Bounded Queue's implementation. Uncomment and run/write the code for each test, one at a time, to check that you get the same sample output as below.

   - Test 1: Complete the try statement. If the Bounded Queue's **dequeue()** method is correct, an exception should be raised when you attempt to dequeue from the empty queue, bq. Handle it by printing out the argument of the exception. Change the argument string in the Bounded Queue's dequeue() definition and rerun this test – is the new message displayed on the screen now?

   - Test 2: Test the Bounded Queue's **enqueue()** method by trying to add 'bob' to bq. When printing the contents of bq to the screen, are `print(bq)` and `print(str(bq))` the same? Does the method **isEmpty()** return the expected result?

   - Test 3: Run given code. When multiple items are added to the queue, does the queue store them in the correct order (test the **repr()** function)? Do the **isFull()** and **size()** methods give the expected results?

   - Test 4: Write a try statement to attempt to add an item to the full bq. If the Bounded Queue's **enqueue()** method is correct, it should raise an exception. Handle that exception in the main() by printing out the exception's argument.

   - Test 5: Run given code. Can an item be removed from a full bounded queue? Does the **dequeue()** method return the expected item, and are the contents of bq updated?

   - Test 6: Run given code. Test the **capacity()** method. How is capacity different from size?

   - Test 7: Try to access the private capacity attribute outside of the Bounded Queue class definition (i.e. in the main). What happens? Does the same thing happen if you try to access a non-private attribute outside of its class definition?

Sample Output:

```
My bounded queue is:
 Max=3
Is my bounded queue empty? True
---------------------------------
```

```
Try to dequeue an empty bounded queue...
Error: Queue is empty
---------------------------------
bob
bob
Is my bounded queue empty? False
---------------------------------
bob eva paul  Max=3
Is my bounded queue full? True
There are 3 elements in my bounded queue.
---------------------------------
Try to enqueue a full bounded queue...
Error: Queue is full
---------------------------------
eva paul  Max=3
bob was first in the bounded queue: eva paul
There are 2 elements in my bounded queue.
---------------------------------
Total capacity is: 3
```

Exercise 2:

In this task, you will compare the runtime of the dequeue() method in the Bounded Queue with the dequeue() method of the Circular Queue.

In the Bounded Queue, the dequeue method removes the item at the front of the queue, and then shifts the remaining items in the list to the left (i.e. to fill in the hole created by removing that first item). **For a Bounded Queue containing n items, what is the big-O time efficiency of the dequeue?**

In the Circular Queue, you just shift the head index when you dequeue an item – no shifting of the actual data elements is required. **For a Circular Queue containing n items, what is the big-O time efficiency of the dequeue?**

Considering your answers to the above two questions, **which queue's dequeue do you predict will run faster?**

1. Create a new file, lab6.py. In this file, import both the Bounded and Circular Queues from queues.py (downloaded in Exercise 1 above).

2. In a main function, create a Bounded Queue object and enqueue 50,000 items into it. For example, you can enqueue the integers 1 to 50,000.

3. In the same main function, create a Circular Queue object and enqueue the same 50,000 items into it.

4. Compute the time required to dequeue all items from the Bounded Queue, using the time module. The time module returns times in seconds. Hint on using the time module:

```
import time
start = time.time()
# The statement(s) that you want to test
end = time.time()
time_interval = end - start
```

5. Similarly, compute the time required to dequeue all items from the Circular Queue.

6. Print the dequeue() runtime for each queue to the screen, as shown in the sample output below.

Sample Output

```
For Bounded Queue, the total runtime of dequeuing 50,000 items is:
0.1325376033782959 seconds.
For Circular Queue, the total runtime of dequeuing 50,000 items is:
0.042598724365234375 seconds.
```

NOTE
We need to enqueue many items in the Circular and Bounded Queues in order to get large enough runtimes to measure for both queues. If the runtime is too short, we will just see zeros returned as the runtime for both. Moreover, keep in mind that the actual runtime will be different on different computers, and slightly different between runs. **How would you modify your code to find the average runtime of 5 trials each?**

7. Repeat steps 2-6, but for larger values of n. For example, n = 55,000 and then n= 60,000 and so on up to about n = 300,000. Copy your times for the different n values into a program like google sheets or excel to plot the time vs n for both the Bounded Queue and Circular Queue. **Which plot is linear and which is quadratic – why?**

   Do not run step 7 during your demo, but be prepared to show your plot to your TA during the demo.

Challenge Exercise: (Optional extra practice)

Write a program to simulate two waiting queues at a grocery store. One queue line is for normal customers, and the other line is for VIP customers. Normal customers can only wait in the Normal customers' queue, and VIP customers can only wait in the VIP customers' queue.

Your program should start by asking the user if they want to add a customer to the queue, serve the next customer in the queue, or exit.

If the user chooses to add a customer, the program asks the user to enter the name of the customer, followed by whether or not the customer is a VIP. If the customer is a VIP, then the customer's name is added to the VIP queue; otherwise, the customer's name is added to the Normal queue. After EACH add operation, the program prints the customer's name and the queue (Normal or VIP) that the customer has been added to. The program also prints both queues after EACH add operation.

After the add operation is completed, the user is asked to enter Add, Serve, or Exit again. Your program will continue to prompt the user repeatedly with these options and perform the task associated with these operations until the user enters Exit. Here is a partial sample output of the program that shows what is displayed after the add operation:

```
Add, Serve, or Exit: Add
Enter the name of the person to add: Fred
Is the customer VIP? True
Add Fred to the VIP line.
Normal customers queue: ]]
VIP customers queue: ]Fred]

Add, Serve, or Exit: Add
Enter the name of the person to add: Bob
Is the customer VIP? False
Add Bob to the normal line.
Normal customers queue: ]Bob]
VIP customers queue: ]Fred]
```

If the user chooses to serve a customer, the program will check the VIP Queue first. If the VIP Queue is <u>not</u> empty, the customer at the front of the VIP queue will be served first. If the VIP Queue is empty, then the customer at the front of the Normal queue is served. After the serve operation, the program will print the name of the customer who has been served and the content of each queue. Here is a partial output from the program that shows what is displayed after the serve operation:

```
Add, Serve, or Exit: Serve
Fred has been served.
```

```
Normal customers queue: ]Bob]
VIP customers queue: ]]
```

If the user wants to add a customer to a queue (Normal or VIP) that is full, the program should print one of the following error messages:

```
Error: Normal customers queue is full
```

or

```
Error: VIP customers queue is full
```

After such error messages are displayed, the program should continue to ask the user if they wish to Add, Serve, or Exit.

If both Normal and VIP Queues are empty, and if the user requests a serve from the queue, then the program should print the following error message:

```
Error: Queues are empty
```

After the error message is displayed, the program should continue to ask the user if they wish to Add, Serve, or Exit.

If the user enters the word Exit, the program should end by printing the word Quitting.

Please note the following:

- You may restrict the capacity of each queue to 3 to simplify testing.
- You should try your code with a Bounded Queue first, and then (once your program is working) swap it out for the Circular Queue. Did you need to change your program at all?

Sample Output:

```
Add, Serve, or Exit: Add
Enter the name of the person to add: Fred
Is the customer VIP? True
Add Fred to VIP line.
Normal customers queue: ]]
VIP customers queue: ]Fred]

Add, Serve, or Exit: Add
Enter the name of the person to add: Barney
Is the customer VIP? False
Add Barney to the normal line.
Normal customers queue: ]Barney]
VIP customers queue: ]Fred]

Add, Serve, or Exit: Add
Enter the name of the person to add: Wilma
Is the customer VIP? False
Add Wilma to the normal line.
Normal customers queue: ]Barney,Wilma]
VIP customers queue: ]Fred]

Add, Serve, or Exit: Add
```

```
Enter the name of the person to add: Pebbles
Is the customer VIP? False
Add Pebbles to the normal line.
Normal customers queue: ]Barney,Wilma,Pebbles]
VIP customers queue: ]Fred]

Add, Serve, or Exit: Add
Enter the name of the person to add: Flint
Is the customer VIP? False
Error: Normal customers queue is full
Normal customers queue: ]Barney,Wilma,Pebbles]
VIP customers queue: ]Fred]

Add, Serve, or Exit: Serve
Fred has been served.
Normal customers queue: ]Barney,Wilma,Pebbles]
VIP customers queue: ]]

Add, Serve, or Exit: Serve
Barney has been served.
Normal customers queue: ]Wilma,Pebbles]
VIP customers queue: ]]

Add, Serve, or Exit: Serve
Wilma has been served.
Normal customers queue: ]Pebbles]
VIP customers queue: ]]

Add, Serve, or Exit: Serve
Pebbles has been served.
Normal customers queue: ]]
VIP customers queue: ]]

Add, Serve, or Exit: Serve
Error: Queues are empty
Normal customers queue: ]]
VIP customers queue: ]]

Add, Serve, or Exit: Exit
Quitting
```