CMPUT 175 - Lab 6: Queues

Goal: Learn about bounded queues and circular queues, and their efficiencies.

Exercise 1: Complete the Bounded and Circular Queue Class Definitions, and Test

1. Download and save queues.py from eClass. This file contains partial implementations for the Bounded Queue and the Circular Queue, as discussed in the lectures.

   - `enqueue(item)` – Complete this method for both types of queues. You may refer directly to the lecture slides on eClass.

   - `dequeue()` – Complete this method for both types of queues. You may refer directly to the lecture slides on eClass.

2. Test your Bounded Queue and Circular Queue classes (especially the methods you have just written). To get you started, queues.py 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
```

****IMPORTANT NOTE****

In Exercises 2 and 3, you are provided with a large amount of code that has been written by someone else. You do not have to understand all of the implementation details of this code. However, you are responsible for understanding the general purpose of this code and how to use the functions we have provided. **Prepare for your lab ahead of time by reviewing the provided code (pay attention to docstrings and other comments)**, and be ready to ask your TA questions about any parts you do not understand.

Exercise 2: Compare the Time Efficiency of Bounded and Circular Queues

In this task, you will compare the runtimes of the dequeue() methods in the Bounded Queue and 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. Download and save lab6_efficiency.py and terminalplot.py. (Save in the same folder as your queue.py from Exercise 1.) In this file, both your Bounded and Circular Queues have been imported from queues.py. This file also contains 4 function definitions: 3 have already been completed for you (`enqueue_experiment`, `average_dequeue_experiment`, and `main`). Read through the comments of those 3 functions to understand what they are doing.

2. Complete the function `dequeue_experiment(queue)` so that it removes the first item in the queue, and continues to do so until that queue is empty. Note that the queue is passed in as an input to the function – so you do NOT have to create it or fill it with data in this function. Use a function from either the *time* or *timeit* modules to measure how long it takes to dequeue all of the items in the queue, and return that time measurement.

   Hint: there is an example of how to use `time.time()` at the bottom of lab6_efficiency.py. The time module returns times in seconds.

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

3. Run lab6_efficiency. (This may take some time to run.) This will run a number of experiments, measuring the time it takes to dequeue all items from Bounded and Circular Queues of increasing capacities. These times will be printed in a table for you to view. The data should also be plotted for you, with time on the y-axis, and n on the x-axis (where n is the number of dequeues made). If a plot is not displayed (i.e. if you see a message that says "Not able to print graph. Continuing…"), plot the data from the table using a spreadsheet program. You can run more experiments with larger queue capacities to get more values for your graph, but keep in mind that doing so will increase the runtime of the overall program. **Which has a more efficient dequeue method: the Bounded Queue, or the Circular Queue?**

<u>Exercise 3: Using High and Low Priority Queues:</u>

<u>Background information:</u>
Computing devices like laptops and smartphones may have multiple computing cores, which allow us to run multiple programs at once. But what happens when the number of programs that we want to run is more than the number of cores in our device? For example, what if we want to run 8 programs on a device with only 2 CPU cores? **Job scheduling** helps the device to run the most important programs first.

Imagine that each program we want to run submits a job request to the operating system before it is actually run. Those jobs are stored in either a high-priority queue or a low-priority queue, depending on how important the program is. For example, processes that are fundamental to how the device operates (e.g. displaying things to the screen, dealing with system input and output) have a higher priority than user-installed applications (e.g. web browsers, music playing app, calculator app).

At any given time, jobs waiting in the high-priority queue will be executed first, in the order that they were requested in. If there are no high-priority jobs waiting to be executed, then the jobs in the low-priority queue can be executed.

<u>Problem:</u>

1. Download lab6_priority.py from eClass, and save in the same folder as your other lab files. This file contains a Job class definition, as well as two functions that have already been completed for you (`get_job()`, and `process_complete()`). Read the comments for this code to understand what it does.

2. Add code to the `main()` function in this file so that every time a new job is created (i.e. every time get_job() is called), that job object is enqueued to the appropriate queue: `high_priority_queue` or `low_priority_queue`.

   Hint: use the Job's high_priority() method to check if a job is high-priority (True) or low-priority (False). Notice that get_job() may also return no job (None) which will <u>NOT</u> go into either queue, so that must be checked for as well.

3. Complete the `main()` function so that whenever a process has finished (indicated when process_complete() returns True), a new process is started by dequeuing a job from the appropriate queue. i.e. If there is at least one job in the high-priority queue, it should be dequeued and assigned to the `current_job` variable. However, if the high-priority queue is empty and there is at least one job in the low-priority queue, then it should be dequeued and assigned to the `current_job` variable. If a job has been successfully dequeued from either queue, the `process_running` variable should be set to True.

<u>Sample Output (your output may differ since the jobs are generated randomly)</u>:

```
######## RUN : 1 ########

Job [USER] Music generated


[PROCESSOR] Busy
Jobs waiting in High Priority Queue :0
Jobs waiting in Low Priority Queue :0


######## RUN : 2 ########

Job [OS] Display generated


[PROCESSOR] Busy
Jobs waiting in High Priority Queue :1
Jobs waiting in Low Priority Queue :0

######## RUN : 3 ########

Job [USER] Browser generated

JOB COMPLETED
ID              : 142
Process Name    : [USER] Music
Priority        : LOW

[PROCESSOR] Busy
Jobs waiting in High Priority Queue :0
Jobs waiting in Low Priority Queue :1

######## RUN : 4 ########

Job [USER] Browser generated


[PROCESSOR] Busy
Jobs waiting in High Priority Queue :0
Jobs waiting in Low Priority Queue :2

######## RUN : 5 ########

Job [OS] File Read generated

JOB COMPLETED
ID              : 329
```

```
Process Name    : [OS] Display
Priority        : HIGH

[PROCESSOR] Busy
Jobs waiting in High Priority Queue :0
Jobs waiting in Low Priority Queue :2

######## RUN : 6 ########

Job [USER] Calculator generated
JOB COMPLETED
ID              : 167
Process Name    : [OS] File Read
Priority        : HIGH

[PROCESSOR] Busy
Jobs waiting in High Priority Queue :0
Jobs waiting in Low Priority Queue :2

######## RUN : 7 ########

Job [USER] Music generated


[PROCESSOR] Busy
Jobs waiting in High Priority Queue :0
Jobs waiting in Low Priority Queue :3

######## RUN : 8 ########

Job [OS] File Read generated

JOB COMPLETED
ID              : 486
Process Name    : [USER] Browser
Priority        : LOW

[PROCESSOR] Busy
Jobs waiting in High Priority Queue :0
Jobs waiting in Low Priority Queue :3

######## RUN : 9 ########

Job [USER] Browser generated


[PROCESSOR] Busy
Jobs waiting in High Priority Queue :0
```

```
Jobs waiting in Low Priority Queue :4

######## RUN : 10 ########

Job [USER] Calculator generated


[PROCESSOR] Busy
Jobs waiting in High Priority Queue :0
Jobs waiting in Low Priority Queue :5
```