CSCI 204 – Introduction to Computer Science II

Lab 8 – Priority Queue ADT

1. Objectives

The objectives of this lab are to:

- Become familiar with priority queues.
- Exercise inheritance.
- Implement a priority queue using a traditional linked list.
- Implement a fast priority queue using an array of queues.
- Understand why the fast priority queue is more efficient in time than the traditional priority queue. Understand better the trade-off between time and space.

2. Introduction

A priority queue is a queue where items are associated with a priority value, and are dequeued based on priority: the item with highest priority is dequeued first in contrast to standard queue, where FIFO order is enforced. Priority is specified by integers where 0 is the highest priority (most favored) and higher numbers have lower priority (less favored).

For example, if we insert the following sequence of integers into a priority queue while using the value of the integer as the priority, 3, 7, 0, 1, 9, and 4, we would result in a priority queue with the values in the order of 0, 1, 3, 4, 7, 9 with the integer 0 being dequeued first.

This style of queue is used when some items are deemed more important than others and items have to wait in line. Phone calls to 911 for instance are more important than routine calls between users of telephone network, and should be given priority in the phone network if too many calls are coming through a phone service at a time. A business might decide that internet traffic associated with online meeting or credit card processing software is more important than web surfing when network demand is high. A printer might prioritize short documents first so fewer people wait around a high-traffic printer.

The class of priority queues as described above in which the smaller priority value is more favored is called MIN-Priority Queue. Depending on the task/application we can think of another implementation of the priority queue data structure, where task(*a*) will be processed before task(*b*) if the priority value of *a* is greater than that of *b*. This type of implementation is called MAX-Priority Queue. We will consider the MIN-Priority Queue only in this lab.

3. Getting Started

Begin by creating a `lab08` directory in your `csci204/labs` directory, and copy the following files from the directory ~csci204/2017-fall/student/labs/lab08

```
FastPriorityQ.py  PriorityQ.py  main.py
```

These files give you a starting point for your lab work.

4. Implement a Queue

Implement a standard linked list Queue ADT or re-use one from a previous project or lab. The file must be named Queue.py and the class must be named Queue. Your queue must support the methods of `enqueue()`, `dequeue()`, `is_empty()`, and `len()`. The `dequeue()` must return the item being removed.

5. Implement a Linked Priority Queue

Your priority queue will be a queue with a singly linked list. Because you have implemented a linked list based queue, you are asked to use object inheritance to complete the priority queue. That is, your priority queue should use directly the methods and attributes that are already in the linked list queue class and that are appropriate for the priority queue. For example, the front and back node in the linked list queue can be used directly without revision, the `__len__()` method, the `is_empty()` method, and the `dequeuer()` method can also be used directly. You will have to just implement the proper `enqueue()` method because the method in a priority queue is different from the one in a regular queue. In addition, the node object used in a priority queue differs a bit from the nodes in a regular queue. The node in a priority queue must have an attribute called *priority*, in addition to the fields in the node for a regular queue. You must implement a class called `PNode` for your priority queue, inheriting from the `Node` class implemented for the regular queue. The nodes from the class `PNode` will be the nodes used in the priority queue.

Complete the `PriorityQueue` class in the `PriorityQ.py` file. Examine the contents of `PriorityQ.py`. You are given a partially completed constructor to which you are to add information. You also need to revise the implementation so the `PriorityQueue` class inherits from the `Queue` class!

The `enqueue()` method takes two parameters. One is the priority of the item being added and the other is the item itself. The method needs to add the item to the queue so that it is:

- the last item with its priority
- after any items with a higher priority (lower number)
- before any items with a lower priority (higher number)

The last method you need to complete is the `__str__()` method. This method should return the name of the class. The application program may use the name for different purposes, so we suggest you give the class a name such as "*PriorityQueue*" without spaces in between words.

Write a method comment block using doc string for each of your three methods that states the purpose of the method, any preconditions, and what is returned (including values such as `None` in special cases).

As specified above, you must also implement a class called `PNode` for your priority queue that inherits from the `Node` class implemented for the regular queue.

Your next task is to test the priority queue class you just developed.

6. Use Different Python Classes for the Same Variable to Test Your Queue

Please open the file *main.py* and examine its content. Note that at the beginning of the file, there are two 'import' statements in the program as follows.

```
from PriorityQ import PriorityQueue as MyQueue
#from FastPriorityQ import FastPriorityQueue as MyQueue
```

The second import statement is commented out for the moment. Python allows the import phrase to specify the name of the class that the application program would prefer to use. In our example, we import the `PriorityQueue` from the file `PriorityQ.py` and the class `PriorityQueue` is renamed as `MyQueue`. The advantage of doing so, as you can probably guess, is that the applications can use the same class name in their program even though the implementation of the class is different. For example, throughout the *main.py* program, the class `MyQueue` is used as the class name, while different implementation of `MyQueue` can be tested separately. As it is, the `main.py` program uses the priority queue as the queue data structure. After completing your lab assignment, you can use the fast priority queue data structure for `MyQueue` by commenting out the first line of import and uncommenting the second line of import, without ever changing any other places in the program.

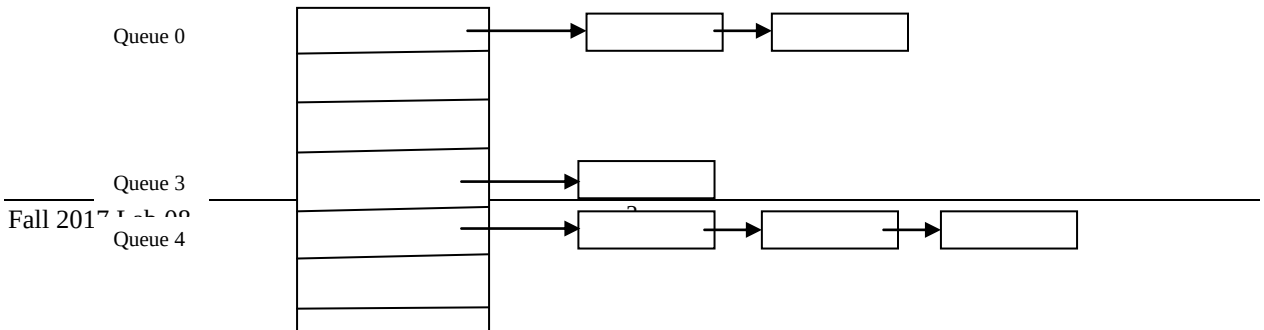
For now, keep the import statements as they are. Test your priority queue implementation by running the `main.py` program. The program saves the output of the program to a file named `PriorityQueue.txt`. Examine the content of this output file after executing the program `main.py`. Your output should be:

```
Fix broken sink
Order cleaning supplies
Shampoo carpets
Replace light bulb
Pet the dog
Take a nap
Empty trash
Water plants
Clean coffee maker
Remove pencil sharpener shavings
```

7. Implement a Fast Priority Queue

A `FastPriorityQueue` implements a priority queue as an array (Python's built-in list) of queues. Each item of the array is a linked list that contains elements with the same priority.

First item of the array is a linked list that contains the priority 0 (the highest priority) items, the second item has a list of the priority 1 items, so on and so forth. If a list is empty, there are no items with that priority. Items with priority 0 have the highest priority. The `FastPriorityQueue` constructor requires a parameter that specifies the lowest allowed priority (highest number). The constructor needs this information so that it can create and initialize an appropriately sized array. Look at the constructor code and make sure you understand what it is doing.



You need to complete four methods for this class to work.

The `enqueue()` method has two parameters. One is the priority of the item being added and the other is the item itself. The method needs to add the item to the end of the list specified by the priority. As a precondition, you can assume the priority parameter will be between 0 and the minimum priority, inclusive.

The second method you need to complete is the `dequeue()` method. This method will search through the lists sequentially until a non-empty list is found. It will remove the first item from the list and return it. If all of the lists are empty, the method returns a `None`.

The third method you need to complete is the `__len__()` method. This method should do exactly what you think it should do, return the number of elements in the queue. Note now that the elements of the queue are probably distributed across different parts of the array.

The last method you need to complete is the `__str__()` method. This method should return the name of the class. The application program may use the name for different purposes, so we suggest you give the class a name such as "*FastPriorityQueue*" without spaces in between words.

Write a method comment block using doc string for each of your three methods that states the purpose of the method, any preconditions, and what is returned, including values such as `None` in special cases.

Now test your priority queue implementation by running the `main.py` program. This time, please comment out the first import statement and uncomment the second import statement as follows.

```
#from PriorityQ import PriorityQueue as MyQueue
from FastPriorityQ import FastPriorityQueue as MyQueue
```

This program will save the output to a file named `FastPriorityQ.txt`. Examine the content of this output file after executing the program `main.py`. Your output should be (the same as before):

```
Fix broken sink
Order cleaning supplies
Shampoo carpets
Replace light bulb
Pet the dog
Take a nap
Empty trash
Water plants
Clean coffee maker
Remove pencil sharpener shavings
```

8. Compare the Priority Queues for Efficiency

Analyze your enqueue and dequeue methods for both the traditional priority queue and fast priority queue. Figure out the best case run time and the worst case run time and put them in a comment at the top of these two methods in the two classes. Remember that access to Python's built-in list (array) is O(1).

- Is the fast priority queue faster for both enqueue and dequeue?
- If you knew most items have the same priority, which implementation of a priority queue should you choose?
- What if you knew the enqueue and dequeue operations would be interleaved, that is, the enqueue and dequeue operations are separated evenly, and the items to be added to the queue are evenly distributed among different priorities?
- With the same interleave condition as above, what if all the enqueue operations would happen before any dequeue operations?

Explain your reasoning. Write your answer in full sentences. Save your writing in a *readme.doc* file. About half a page should be sufficient for your answers.

9. Confirm the Efficiency Analysis with Some Experiments

While algorithm analysis is very reliable, it helps if we actually see some experiment results. The `time_enqueue()` method times how long it takes to enqueue 10,000 items. You will now write more tests to find answers to each of the questions you just answered with reasoning.

- Copy the `time_enqueue()` method in the `main.py` file and make a `time_dequeue()` method. Enqueue all 10,000 items and then dequeue all 10,000 items. Time the dequeue portion of the method. Test your method on both Priority Queues.

- Copy the `time_enqueue()` method and make a `time_few_priorities()` method. Set the priorities to only have 3 possibilities 0, 1, and 2, instead of the default 16, 0..15. Time the enqueue portion of the method (the same as the `time_enqueue()` method). Test your method on both Priority Queues.

- Copy the `time_enqueue()` method and make a `time_interleaved_enq_deq()` method. In this method you will make it so enqueue dequeuer are interleaved. We can assume that the random number generators can produce evenly distributed integer values for the given range. Test your method on both Priority Queues.

- Copy the `time_interleaved_enq_deq()` method and make a `time_enqueue_before_dequeue()` method. Revise the method such that all enqueue operations take place before dequeue operations. But we want to have the items evenly distributed across different priority levels. Test your method on both Priority Queues.

While the exact number of seconds to complete a task may vary a little bit, the trend should be very clear that if the items in the queues are randomly (evenly) distributed, the fast priority queue should perform better than the traditional linked list.

10.Submitting Your Results

Clear all unnecessary files, make a zip file out of all the submission files, then upload the zip file to Moodle and double check your submission to make sure your files are all uploaded successfully.