COMP 210 – Data Structures and Analysis (Sec 1&2)

Assignment #5 – Priority Queues & Heaps

Issue Date: October 24th, 2024.

Due Date: Monday, November 4th, 2024, 11:55PM

Marks: 5

This assignment is based on an implementation of Priority Queues using Heaps. For this assignment, you are tasked with implementing the Maximum Binary Heap within a practical application scenario. Imagine a local hospital struggling to manage its emergency room efficiently. They've enlisted your assistance because their current system, designed to prioritize the sickest patients, is lagging. As a result, patients in dire need often endure prolonged wait times.

Recall that a Queue operates on a FIFO (First In, First Out) principle, capturing a waiting line perfectly. If the hospital solely aimed to serve patients based on their arrival order, a FIFO Queue would suffice. However, such an approach is suboptimal. A patient with a life-threatening condition should be prioritized over another with a minor ailment, regardless of their arrival order. In our system, each patient in the queue will have an associated priority level. Thus, when retrieving the next patient from the Priority Queue, the one with the highest priority is chosen.

As specified in Part B, you will need to complete 4 Tasks. In Task 1, we will see an inefficient way to implement such a Priority Queue. In Task 2, you will make use of your understanding of binary heaps to devise a more efficient Priority Queue. In Task 3 you will convert an existing list of patients into a heap, and finally in Task 4, you will compare the results of Task 1 & Task 2.

As always, don't stress if something isn't working — come to the office hours and/or post on Piazza! If you feel entirely stuck and unable to resolve your issue, please email Qiwei Zhao: qiwei@cs.unc.edu and our TA/UTAs team will work together to help you!

A. Complete the Prerequisites

Complete the following requirements as done in previous assignments.

1. Prepare the Java Develop Environment

This step should already have been completed in the previous assignments. You should have already installed the required software such as Java, and IntelliJ.

2. Set up your java project environment.

To set up the starter code for Assignment #5, follow these steps:

I. Open IntelliJ IDEA.
II. Navigate to your previously created **COMP210** Project. Within this project, right click on the **src** directory and choose "New Package". Name this new package "**assn05**".
III. Locate the **six** starter code java files that came along with this homework package and copy all the Java files and paste them into the "assn05" package you created in the COMP210 Project.
IV. Ensure that the JDK is set up as it was in previous assignments.

With these steps completed, you are all set to proceed with this assignment. Feel free to incorporate any helper methods or classes you deem beneficial. There are likely multiple ways to successfully complete this assignment.

B. Tasks to be completed

Task 1. SimpleEmergencyRoom

`**Prioritized.java**` introduces an *interface* that defines *elements* (e.g. patients) suitable for a priority queue. Each element that implements the Prioritized interface will possess a *value of type V* and a *priority of type P*. As in the previous assignment, the notation `**P extends Comparable<P>**` asserts that elements of type P must be comparable, enabling the identification of elements based on priority. The interface comprises of three methods: (i) getValue(): to retrieve the value, (ii) getPriority(): to retrieve the priority of an element, and (iii) *compareTo(Prioritized<V, P> other)*: that takes another element 'other' of type 'Prioritized' and returns the result of the comparison.

`**Patient.java**` implements the *Prioritized* interface and symbolizes an element that is a patient entering the ER. It offers two constructors: the first accepts both a *value* (indicating e.g. the order of arrival) and a *priority* (indicating the severity of the case), while the second only requires a value, and assigns a random priority between 0 and 999,999. A higher priority value implies a more severe injury, warranting earlier treatment. For instance, a patient with a priority of 3 is in a worse condition than one with a priority of 2 and hence should be attended to first.

Although you've encountered Java Generics in prior assignments, the header for the Patient class might appear intricate. The `Patient` class implements the Prioritized interface, allowing the value to be of any type V while designating the priority as an integer which must be *comparable*.

`**SimpleEmergencyRoom.java**` depicts the current (inefficient) methodology that the hospital database employs to oversee the ER. At present, patients are cataloged in an *ArrayList* according to their arrival sequence (Note that *ArrayList* implements the *List* interface.) While most methods are already completed, you will need to implement the ***dequeue*** method as instructed below.

TODO - Task 1:

- Fill in the dequeue method to find the patient with the highest priority using a *for* loop. Return that patient and remove them from the list.
- Write some tests in Main.java to convince yourself that your implementation is working.

Task 2A. Improved System – *Maximum Binary Heap Emergency Room*

The issue with the implementation in task 1 is that it takes O(n) time to run the *dequeue ()* method. However, you recently learned about a cool data structure called a Binary Heap and figured it would apply great to this situation! Since the patients with highest priority are the ones, we want to help first, you will be creating a Maximum Binary Heap. This will allow us to bring the runtime of the *dequeue ()* method from O(n) down to O(log n).

BinaryHeap.java is the interface that outlines all the methods you will need to implement in **MaxBinHeapER.java**. MaxBinHeapER has an overloaded constructor. For now, you can ignore the second one (the one that takes in an array), we will get back to it in task 3.

TODO - Task 2A:

- **Complete the methods in MaxBinHeapER.java according to the specifications in BinaryHeap.java.**
- The enqueue method is overloaded in MaxBinHeapER but the functionality behind the two are the same. The only difference is that in **enqueue(V value, P priority)** you create a Patient using the constructor that takes in both a value and priority, and in **enqueue(V value)** you create the Patient using only the value and the priority is randomly set.
- For **dequeue**() and **getMax**(), you should return null if there are no patients in the queue.
- Write some tests in **Main.java** to convince yourself that your implementation is working.

Task 2B: Implementing Update Priority in MaxBinHeapER

In the actual emergency room, the priority of patients is constantly changing. Sometimes, worsening symptoms require an elevation in priority, while at other times, symptom improvements after initial treatment necessitates a reduction in priority. In this task, you will enhance the MaxBinHeapER class by adding the ability to update the priority of an existing patient. This feature is crucial for reflecting changes in patient conditions and ensuring that the queue remains ordered according to the most urgent needs.

TODO - Task 2B:

- Implement an **updatePriority(V value, P newPriority)** method: According to the specifications in **BinaryHeap.java**, this method searches for a patient in the heap using their value, updates their priority to **newPriority**, and then restructures the heap to maintain the max heap property.
    - If the patient is found and their priority is updated, **ensure** the heap's integrity by **performing necessary adjustments**.
    - If the patient is not in the heap, the method should handle this gracefully, and not modify the heap in any way.
    - If **newPriority** < 0 and the patient is found, remove the patient from the heap. This reflects the patient's improved condition, indicating they no longer require emergency attention.
    - **Note: However, if the original priority was already < 0**, and the value is ***not*** matching, then such patients should not be removed from the heap. This ensures that the heap only modifies patients who are actively involved in the priority update process. Patients with an initial priority < 0 were originally marked as non-critical, indicating that they require lower-priority monitoring.
- Testing**:** Write tests in **Main.java** to verify your implementation works as expected. You may want to write tests that search for a patient who is in the heap, and one who is not to ensure your method works correctly.
- Ensure your implementation does not compromise the efficiency of the heap operations, maintaining the O(log n) complexity for updating priorities.

Task 3. Hospital Transfers

Sometimes, another hospital down the street is overflowing with patients and transfers them to your ER. These new patients are represented by an array **Patient[]** in the database. You need to add functionality to your heap to turn this array of patients into a Maximum Binary Heap.

TODO - Task 3:

- Fill in the *second constructor* in **MaxBinHeapER.java** to build a heap when given an initial array of Prioritized objects. (A naive approach would run in O(n*log n) time; however, if you have all the items in advance a better algorithm, as discussed in class, is called **Build-Heap** can be used to create a heap in O(n) time. You may optionally implement the Build-Heap algorithm).
- Write some tests in Main.java to convince yourself that your implementation is working.

- An example of how you can test this is shown in Main.java under the Part 3 comments.

Task 4. Bringing it Home

Now that we have implemented a Priority Queue in both an inefficient and efficient manner, let's try to compare the two!

In Main.java, look at the *compareRuntimes()* function. You will need to implement this function and then call it from your main method to test its behavior. The function will return an array of doubles containing four values.

TODO - Task 4:

1. First, you will create an emergency room using your implementation from **Task 1** and then add in 100,000 patients. You will record the time needed to deque all 100,000 patients. You can use the built-in *System.nanoTime()* which returns the current time represented in nanoseconds; if you call this once before dequeuing and then once after, you can subtract the results to get the total time. Store the number of nanoseconds required to dequeue all 100,000 patients in the **first index** (index 0) of the array. Then, in the **second index** of the array, calculate and store the average time it took to dequeue a single patient in the example.
2. You will now repeat the same exercise but using your new **MaxBinHeapER** Implementation. The **third index** in the return array should contain the time needed to dequeue all 100,000 patients and the **fourth index** should return the average time needed to dequeue a patient using the faster Priority Queue implementation.
3. Call this function from the main method in Main.java and verify that the results are what you would expect.

C. Submit to Grade Scope for Grading

It's time to submit your work on Gradescope for evaluation!

Before the deadline, you can submit your assignment multiple times without any penalties. We encourage this approach, especially since the assignments are autograded, offering nearly instant feedback. We aim for you to leverage these resubmissions to achieve full autograding credit.

Creating a Zip Archive in IntelliJ: Let’s return to IntelliJ to prepare your zip archive for submission:

Mac Users

Along the bottom of your window, you should see an option to open a terminal integrated into IntelliJ. Type the following command (all on a single line, including the ‘.’):

zip -r assn05_submission.zip assn05 . (Your path should be set to the src file)

If zipping from COMP210 workspace:
zip -r assn05_submission.zip src/assn05

In the file explorer pane, look for the zip file named "assn05_submission.zip". If you right click on this file "Open in -> Finder" on Mac, the zip file's location on your computer will open. Upload this file to Gradescope to submit your work for this assignment.

Windows Users

Please navigate to your course workspace in the File Explorer window. Then right click on the src folder in your exercises directory and compress the directory into a zip folder. You can name it "assn05_submission.zip" Please pay attention, the level of the folder needs to be the same as the description. Compression software on Windows may introduce additional folder structure, and the file naming must be the same.

Before uploading the zip folder to Gradescope, please delete any files that showed up in the src/ folder that were not actually part of assn05. Specifically, in this assignment, the files that need to be kept are ended with the .java. Then:

1. Log in to Gradescope.
2. Choose the assignment titled "**Assignment #5 (Binary Heap)**".
3. You'll find an option to upload a zip file.

Autograding will require a short while to process. For this exercise, there isn't a "human graded" component. Consequently, you should aim to achieve the full score of 100 points. If you encounter any reported issues, you should try to resolve these and then continue to resubmit until all issues are cleared.