Assignment 7

Priority Queues

Introduction

In this assignment you will be implementing the `PriorityQueue` interface using an array-based `Heap` data structure. A reference-based list implementation of a `PriorityQueue` is provided for you (`LinkedPriorityQueue.java` and `A7Node.java`), so you can run the tester and compare the run times of the two implementations.

Although there is a tester provided for this assignment, it does not include a comprehensive set of sets for each method. You should add your own tests for any test cases not considered.

NOTE: The automated grading of your assignment will include some different and additional tests to those found in the `A7Tester.java` file. For all assignments, you are expected to write additional tests until you are convinced each method has full test coverage.

Objectives

Upon finishing this assignment, you should be able to:

- Write an implementation of an array-based Heap
- Write code that uses an implementation of the Priority Queue ADT

Submission and Grading

Attach `HeapPriorityQueue.java` to the BrightSpace assignment page. Remember to click **submit** afterward. You should receive a notification that your assignment was successfully submitted.

If you chose not to complete some of the methods required, you **must** provide a stub for the incomplete method(s) in order for our tester to compile. If you submit files that do not compile with our tester, you will receive a zero grade for the assignment. It is your responsibility to ensure you follow the specification and submit the correct files. Additionally, your code must not be written to specifically pass the test cases in the tester, instead, it must work on all valid inputs. We may change the input values during grading and we will inspect your code for hard-coded solutions.
This video explains stubs.

Be sure you submit your assignment, not just save a draft. All late and incorrect submissions will be given a zero grade. A reminder that it is OK to talk about your assignment with your classmates, but not to share code electronically or visually (on a display screen or paper). Plagiarism detection software will be run on all submissions.

Instructions

The `LinkedPriorityQueue` implementation provided does not make use of the $O(logn)$ heap insertion and removal algorithms we discussed in lecture. Instead, when an item is inserted, the queue is searched from beginning to end until the correct position to insert the item is found.

The `HeapPriorityQueue` insertion implementation you write should improve on this implementation so that the insert runs in $O(logn)$ using the `bubbleUp` algorithm covered in lecture. Similarly, your `removeMin` should use the `bubbleDown` algorithm.

1. Download and unzip all of the files found in `a7-files.zip`.

2. The documentation for the methods you will implement are all found in `PriorityQueue.java`. I have also provided some comments in `HeapPriorityQueue.java` to help you with your implementation.

3. Compile and run the test program A7Tester.java with the LinkedPriorityQueue to understand the behaviour of the tester:
   ```
   javac A7Tester.java
   java A7Tester linked
   ```

   Note: the A7Tester is executed with the word "linked" as an argument. This argument allows us to run all of the tests against the LinkedPriorityQueue implementation provided for you. You will notice it is extremely slow with larger file input sizes (for the largest input size we will only test it with the heap implementation).

4. Compile and run `A7Tester.java` using the `HeapPriorityQueue` implementation:
   ```
   javac A7Tester.java
   java A7Tester
   ```
   (a) Uncomment one of the tests in the tester
   (b) Implement one of the methods in HeapPriorityQueue.java
   (c) Once all of the tests have passed compare the runtime against the LinkedList implementation (outlined in step 3)

HINT: The first big thing you need to think about is wheter you will store the `root` element in the heap an index 0 or index 1. I have provided a `rootIndex` field for you that should be set to 0 or 1 depending on what method you choose. This choice will affect how you access parent and child indexes, as well as how you print out the contents of the heap when debugging (you will notice an if-statement in the `toString` provided).

Another thing to think about is creating helper methods to help with the `bubbleUp` and `bubbleDown` methods. I made a number of helper methods in my implementation. The names of my helper methods are `isLeaf`, `minChild`, and `swap`. It might be a good idea to implement your own helper methods (possibly even more than I implemented for my solution).