

Implementing min-heaps

Preamble

The result of this assignment is used in A7, which is an implementation of Dijkstra's algorithm to find a shortest path in a graph. Google maps and other map apps use the shortest path algorithm to find the best route from one place to another.

A6 involves implementing a min-heap with the added functionality of being able to change a priority. It is an example of a common phenomenon: changing a data structure when functionality is added to keep it all efficient.

Keep track of how much time you spent on A6; we will ask for it upon submission.

Last semester, a similar assignment took a mean and median of 3.89 and 4.0 hours. We believe that this assignment will take less time because of how we have structured it. You have to write only about 55 lines of code, in 8 small methods. We suggest getting A6 done *well* before the due date. The last weeks of the course can be stressful.

So that A6 doesn't take *too* long, we give you a complete JUnit testing program. If your code passes its test cases, your code is correct. *You may still lose many points for not adhering to the execution-time bounds given in the specifications of the methods.* For example, if an operation should be done in expected constant time and your code searches an array in linear fashion, that will cost points.

Collaboration policy and academic integrity

You may do this assignment with one other person. Both members of the group should get on the CMS and do what is required to form a group well before the assignment due date. Both must do something to form the group: one proposes, the other accepts.

People in a group must *work together*. It is against the rules for one person to do some programming on this assignment without the other person sitting nearby and helping. Take turns "driving" — using the keyboard and mouse.

With the exception of your CMS-registered group partner, you may not look at anyone else's code, from this semester or earlier ones, in any form, or show your code to anyone else (except the course staff), in any form.

Getting help

If you don't know where to start, if you don't understand testing, if you are lost, etc., please SEE SOMEONE IMMEDIATELY — an instructor, a TA, a consultant. Do not wait. A little in-person help can do wonders. See the course homepage for contact information.

The release code

The zip file `release.zip` contains an Eclipse project with two `.java` files:

1. File `Heap.java`. The methods you must write are marked with `//TODO` comments, which tell you the order in which method bodies should be written and tested and give information on testing. Complete other stubbed-in methods *only* if you use them.
2. File `HeapTester.java`, which has methods to completely test the methods you write.

These files go in package `heap` — not in the default package. As with assignment A5, you can start the project by (1) opening a project from the file system or (2) the usual way — starting a new project and copying/pasting. See the Piazza A6 FAQs note for details.

What to do for this assignment

Your job is to implement the methods in class `Heap` that are marked “//TODO”. These are: `add`, `ensureSpace`, `swap`, `bubbleUp`, `peek`, `poll`, `bubbleDown`, and `changePriority`.

Hints, guidelines, suggestions, requirements

1. You must implement the methods marked with “//TODO” *so that they have specified time bounds*.
2. One method is stubbed in that does not have a “//TODO” note. You do not have to write it, and if you do, you can change its specification. We will not test it by itself. We placed it there because *we* found it useful.
3. You can add other methods if you want. Points may be deducted if methods you add do not have good javadoc specifications.
4. Points will be deducted if you violate the text given in the bodies of the “//TODO” methods.
5. Class `HeapTester` does all necessary testing. If running `HeapTester` does not show an error, your class `Heap` should be correct.
6. We have declared *fields in `Heap` and written a class invariant for `Heap`*. Study the class invariant. Do not declare any other fields. Point 6 discusses the reason for the `HashMap`.
7. **Using an array for the heap.** Generally, one uses an `ArrayList` for the heap. Instead, we use an array, for two reasons: (1) you get to see how `ArrayList` can change the size of its backing array when the backing array is too small. For this purpose, you will write procedure `ensureSpace`. (2) The use of array notation will make the code a lot easier to read.
8. Special problem. Class `Heap` would be easy to write, using just array `c` for the heap, except for one issue. A call `changePriority(v, p)` changes the priority of value `v` to `p`. This requires finding value `v` in array `c`, and without any other data structure to help, this could cost time linear in the size of the heap.

To overcome this problem, we add a field `map`, of type `HashMap<V, Integer>`, which maps a value in the heap to its index in array `c`.

Of course, when value `c[i]` in the heap is moved to a different position, the `HashMap` entry for that value must be changed accordingly.

Using this technique, the expected time for updating a priority should be $O(1)$ for finding the corresponding element in the `HashMap` and then $O(\log n)$ for updating the heap. The corresponding worst-case times are $O(n)$ and $O(\log n)$.

What to submit

1. Remove all your `println` statements from class `Heap`; 5 points will be deducted if your code outputs anything.
2. In the comment at the top, put the hours `hh` and minutes `mm` that you spent on this assignment. Be careful changing `hh` and `mm`; don't change anything else. Put your name(s), netid(s), and write a few lines about what you thought about this assignment.
3. Submit (only) file `Heap.java` on the CMS.