Assignment 5: Priority Queue YEAH

Ashwin Siripurapu

May 21, 2013

- This is the first assignment where you are given rather broad freedom to make your own design choices regarding the implementation of your classes, provided that you implement the interfaces we provide using the algorithms specified in the assignment handout.

  Consequently, you should be thinking about *design choices* constantly. Whenever you come to a point where you could sensibly implement a particular feature in two or more different ways, stop and think about the pros and cons of each way before proceeding.

  For instance, we require that you implement a `size` method for each of your priority queues. There are a couple reasonable ways to do this for the linked list priority queues: you could maintain a member variable that stores how many elements there are (as in the code for the `Vector` class that we created in lecture), or you could loop through the linked list (also as in lecture) and count up how many elements there are.

  What are the pros and cons of each way? Think in terms of big–O running time and space efficiency.

  Which way (if either) is, on balance, better than the other for our purposes?

  Think about your design choices like this, and make sure you can defend your answers.

- For the `VectorPQueue`, you should not be manually managing memory. All memory management should be abstracted away to the underlying `Vector` that stores the elements of your priority queue.

- For the linked list priority queues, be very, very cautious about two things:

    1. *Memory management.* Don't forget to free memory that you no longer need, and don't use memory that you have already freed.
    2. *Edge cases.* Think about all the special cases that could affect the behavior of your code. What if the linked list is empty to begin with? What if you are inserting or deleting from the head element or the last element of the list? What if you are removing the only remaining element in a list? The code you write should be robust enough to deal with all these special cases.

- For the `HeapPQueue`, make sure you really understand how the algorithm works before you dive into the code. Take a look at the picures and convince yourself that if you start out with a valid binary heap and then perform an insertion or a deletion using the bubble–up or bubble–down procedures, you will subsequently end up with a valid binary heap.

Also, you should *not* be implementing the binary heap data structure the way we did the binary search tree we did in class, with a `struct node` that has a `left` and a `right` pointer. Look over the handout and see how to represent a binary heap using a raw array, which you will manage and grow as necessary. By the way, if you're curious, think about how you might decide to shrink the array if you have a bunch of unused allocated space ;)