Data Structures

Elise Harrell

Question Answers

C-Level Project

* Which version of the list-based Priority Queue did you implement?
  + A PQ where the enqueue method always appends an item to the end of the list, and the dequeue method searches the list for the highest-priority item and removes it from wherever it is.
* What are the Big O times for the enqueue(), dequeue(), size(), str(), and is\_empty() functions for the PQ you implemented? Explain your answers.
  + Is\_empty: O(1)
    - no loop
  + Size: O(1)
    - no loop
  + Str: O(1)
    - no loop
  + Enqueue: O(1)
    - no loop
  + Dequeue: O(n)
    - Because of search function for loop

B-Level Project

* Which version of the Node-based Priority Queue did you implement? Why did you pick that one?
  + I picked the one that enqueues to the end of the list, and dequeues the highest priority node by searching the nodes. I picked this because I picked to dequeue the highest priority in the list based one, and I felt familiar with it, and like I could easily figure it out. Also for the sake of symmetry.
* What are the Big O times for the enqueue, dequeue, size, is\_empty and \_\_str\_\_ functions for the PQ you implemented? Explain your answers
  + Enqueue
    - O(1)
    - There is no loop
  + Dequeue
    - O(n)
    - There is a call to the search function that I made, which has a loop
  + Is\_empty
    - O(1)
    - There is no loop
  + \_\_str\_\_
    - O(n)
    - There is a loop
  + Size
    - O(1)
    - There is no loop

A-Level Project

* How did you decide how to count the amount of “work” each function is doing? What did you change in the code?
  + I created a variable called work, and added one to it every time I did something.
* Are the functions running in the expected Big O, more or less? Explain your answer and back it up with data for each function.
  + The functions are getting O(n) or more. For the list function I am getting O(n) because the is one loop and not much work happens in the loop. Only about 1 variable will change in that loop. Whereas for my node-based function, there is only one while loop but many variables are being counted and changed, so the work is much higher because the loop repeats for every item and each loop does 3-4 different things each time. For example, in the list based queue, when there was a list of 312522, the work was 312537. But in a node based queue of 310986, the work was 1244006.
* How does the “work” required to run the different operations change as the size of n changes? If you were to chart the times and work compared to the number of n, what would the graphs looks like? Were you surprised by any of the output? If so, what?
  + I was surprised that while both functions are only O(n), one grew so much more than the other. One increased much quicker but was still O(n) and the graphs were both linear. It seemed like the size of n was the same as the work in a line queue, but the work was 4 times bigger than the size of n and still grew linearly.