CSCI 2270 – Data Structures and Algorithms
Instructor: Hoenigman/Zagrodzki/Zietz
Final project
Due Sunday, May 6, before 5pm

Priority Queue Implementation and Analysis

Your local town is hosting this year's Very Very Pregnant People Pageant, which gathers hundreds of women who are very, very pregnant. Unexpectedly during the pageant, every single pregnant woman goes into labor. You live in a small town with only one hospital, so every pregnant person from the pageant heads to the hospital at the same time. How will the doctors decide the order in which the women will be seen? As the hospital's lone programmer, you volunteer to quickly code up a priority queue that will process the pregnant people properly.

As with most things in programming, there are many ways to solve a problem. To make things interesting, you decide to implement this priority queue using three different implementations and compare their respective performances in order to determine the best data structure for the job.

You need to evaluate the performance of each implementation in terms of runtime for building the queue, inserting new elements, and removing elements. Your evaluation and the results need to be described in a 2 - 3 page paper, single-spaced, 12pt. Times New Roman font.

Assignment data

There is a file on Moodle called *patientData2270.csv* that includes a list of patients, time to delivery, and estimated treatment time. The patients with shorter time to delivery are higher priority than those with a longer time to delivery. The first row in the file is the header row, which shows what every column in the file includes. The second row in the file is the first row of data that you need to add to the queue:

Aaliyah, 152, 62

The fields in the header row show:
Name – first name of the patient
Priority – estimated time until baby born in minutes, between 30 and 300.
Treatment – estimated time that patient needs to spend with doctor in minutes, between 30 and 90.

Assignment requirements

Build priority queue using heap, linked list, and STL

You need to implement a priority queue using three approaches – a binary heap, a linked list, and the C++ priority queue offered in the Standard Template Library.

Each of your implementations should have its own .hpp and .cpp files, similar to the other data structures we have implemented this semester. For the STL, you don't need to build a separate class, you can just include the priority queue header in your main driver file.

Read priority queue data from a .csv file
The data that you need to store in your priority queue is in the patientData_2270.csv file available on moodle. The name of that file needs to be handled as a command line argument to your program. There are 880 rows in the file. You can use an array that is large enough to store the entire data set, you don't need to implement array doubling.

Process priority queue data
After reading the data from the .csv file, you'll need to build your priority queues using each of your implementations. This means your implementations will need all the required methods to enqueue and dequeue items to and from a priority queue. Your priority queue should prioritize on the time until the baby is born (minimum first). In the case of equal times until the baby is born, prioritize next on the treatment time the patient requires (again minimum first). For example, a patient with an estimated delivery time of 30 minutes and treatment time of 10 minutes will have a higher priority than a patient with an estimated delivery time of 30 minutes and a treatment time of 20 minutes.

Runtime analysis

The different operations on your priority queue should have different runtime performance. For example, to remove an element from the heap should be constant, which means that a priority queue with 10 elements should be able to have all of its elements removed in approximately the same runtime as a priority queue with 500 elements. The different implementations – array and linked list – should also have different performances. We expect to see that adding an element to the array heap is faster than adding it the same element to the linked list, generally speaking. The STL may or may not be faster than either of your implementations just because it's a built-in library.

To analyze the runtime performance, you need to perform the build and remove operations at least 500 times for each implementation and calculate the mean and standard deviations of the runtimes in milliseconds. Your analysis needs to include running the data on files of different sizes so that you can get a picture of how the runtime changes with the size of the input data. There are 880 rows of data. You could divide up your files into sets of 100 rows and run tests with 100, 200, 300, and so on.

Note: if you're running 8 tests 500 times each, it will take some time. Don't wait until the last minute to start these tests.

User input

Your program can ask for user input if you would like, but for this assignment, it's not required. You will be doing interview grading and can explain your implementation to the TA during the interview. If you are interested in having a menu, it could include items such as

1. Build LL priority queue
2. Build heap priority queue
3. Dequeue LL
4. Dequeue heap
5. Dequeue all LL
6. Dequeue all heap

Code separation into multiple files

Your code needs to be separated into multiple files, such as priorityQueueLL.hpp, priorityQueueHeap.hpp, priorityQueueLL.cpp, priorityQueueHeap.cpp, and FinalProject.cpp, similar to other assignments in this class. If you're working in sublime, you won't be able to build all of these files in your IDE. You will need to build on the command line or write a make file. To build from the command line, use something along the lines of:

g++ -std=c++11 priorityQueue.cpp priorityQueueHeap.cpp FinalProject.cpp –o FinalProject

then run your code with
./FinalProject

Written requirements

Your project needs to include a 2 - 3 page paper that describes your project and the results of your analysis of the three implementations. Your paper needs to include the following sections:

Purpose - What is the purpose of this project? What are you evaluating?

Procedure - What data structures are you using for your implementations? This section should include a description of how each implementation works. Also include what metric you will be using to measure the runtime differences between the implementations.

Data – Describe the data set used for this project. This should include a description of the data, including what field is the priority and how ties are broken.

Results – Describe how each implementation performed. This section needs to have at least one graph showing the performance differences. You also need to include the mean and standard deviations of the runtimes for each implementation.

Work is individual

We're not working in teams on this project. It is assumed that everyone will write their own code and produce their own write up.

What to submit

Submit your code and report as FinalProject.zip to the Final Project Submit link on Moodle.