Efficient Elevator Control
S19 CSCI332 Design and Analysis of Algorithms

Assigned: 2019-03-25
Reference: Sorting and Priority Queueing — Due Date: 2019-04-08 (midnight)

Summary:

Abstract: Create a project to efficiently control elevator movement of passengers in a multistory multi-elevator building using sorting and STL data structures.

Objectives:

1. Efficient Sorting Algorithms
2. Transform and Conquer Techniques
3. Priority Queues and Regular Queues
4. C++ Standard Template Library
5. Iterators and Iterator Algorithms
6. Algorithm Analysis

Grading: 45 pts - A $\geq$ 41.85; A- $\geq$ 40.50; B+ $\geq$ 39.15; B $\geq$ 37.35; B- $\geq$ 36; C+ $\geq$ 34.65; C $\geq$ 32.85; C- $\geq$ 31.75; D+ $\geq$ 30.15; D $\geq$ 28.35; D- $\geq$ 27

Outcomes: R1 (CAC-c,i,j,k; EAC-e,k,1); R2 (CAC-a,b,j; EAC-a,e,1); R6 (CAC-a,c,j; EAC-a,e,1) (see syllabus for description of course outcomes)

Description:

You have been tasked with creating a simple elevator control system for a multistory office building. The objective is to use priority queues, sorting, and iterators to create an efficient control system that minimizes high wait times for an elevator to service a request, and minimizes high travel times from the pickup floor to the destination floor. Conceptually, the lobby of every floor in the building has an elevator keypad with all of the floors located on the keypad, except for the current floor. When a passenger presses the button for their desired floor, the system assigns that passenger to a pick-up queue, indicating both the pick-up and destination floors. Every time any elevator stops on *any floor*, the pick-up queue is analyzed, with elements from the pick-up queue reassigned to a specific elevator's priority queue as follows:

1. If there is an elevator on the floor associated with the pick-up floor, then both the pick-up floor and destination floor are assigned to that elevator's priority queue, according to the floor order. Otherwise,
2. if the destination floor is on a higher floor than the pick-up floor,
   (a) then the pick-up floor and destination floor are assigned to an elevator's priority queue according to the floor order, whose current location is below the pick-up floor,
   (b) whose next stop (front of the queue) is above the pick-up floor and nearest to the destination floor, and
   (c) has no more than four (4) stops between the front of the priority queue and the destination floor location.
3. If the destination floor is on a lower floor than the pick-up floor,
   (a) then the pick-up floor and destination floor are assigned to an elevator's priority queue according to the floor order, whose current location is above the pick-up floor,
   (b) whose next stop (front of the queue) is below the pick-up floor and nearest to the destination floor, and
   (c) has no more than four (4) stops between the front of the priority queue and the destination floor location.

When an elevator stops on a given floor, it will display on its screen above the elevator, the list of destination floors (next stops) that include no more than four (4) stops away from the current (pick-up) floor. As elevators stops on the various floors in their priority queues, they will dequeue those floors.

Input File: The input file will consist of a sequence of arrivals on various floors with the destination being indicated. An example of the file will be as follows:

```
24 4
3 6 19 21
5 2 9 1
8 7 2
...
23 1 5 7
15 9 11 14
```

Where the first line is the number of floors in the building F followed by the number of elevators E that are in operation. The second line are the initial starting floors of each of the e elevators. and each line that follows is a set of passengers making requests for an elevator to take them to a given destination floor - elements $d_{(i,j)}$ starting with the second element on the line - from the pick-up floor p_i indicated by the first element of the line. So, the input file format is shown below:

$$
\begin{array}{lllll}
F & E & & & \\
e_1 & e_2 & \cdots & e_E & \\
p_1 & d_{(1,1)} & d_{(1,2)} & \cdots & d_{(1,m_1)} \\
p_2 & d_{(2,1)} & d_{(2,3)} & \cdots & d_{(2,m_2)} \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
p_n & d_{(n,1)} & d_{(n,2)} & \cdots & d_{(n,m_n)}
\end{array}
$$

The input file(s) may contain errors, therefore, you must check that the general format is correct, and that the requests from a given floor make sense.

Obtaining Project Files:

1. Log into your account on the `gitlab.cs.mtech.edu` department server and fork this project `EfficientElevator` into your own account.
2. Go to your cloned project in your own account on the GitLab department server and select [settings] and then [members] for the project and add your instructor (and any teaching assistants for the course) as a `Developer` member.
3. (optionally - only need to perform this step once) If you are going to use the `ssh` protocol to obtain your project files from the GitLab department server, you need to make sure the `ssh` key from the machine on which you will be working with the project are copied to the list of valid keys in your account.
4. copy either the `ssh` or `http` url paths to your clipboard
5. Log into the `lumen.mtech.edu` department linux server with your department credentials. If this is the first time you have logged into the server, your username will be the first part of your campus email account and your default password will be your student id; make sure to change your password the first time you login using the `passwd` linux command.
6. Create a projects folder for the course using the command `mkdir -p ~/CSCI332/Projects`, and then change into this directory using the command `cd ~/CSCI332/Projects`.
7. Clone the project to your course project folder using the command `git clone <url>`, where `<url>` is the project url you copies to your clipboard. This will create a new directory for the project.

8. Your should use the command `cd` to change into the new project folder you just cloned.
9. Now proceed to the project activities in the next section.

Project Activities:

The following tasks need to be performed in order to complete the project:

1. Design a UML for passenger ADT - include in your project folder that will be sent to GitLab.
2. Design a UML for elevator ADT - include in your project folder that will be sent to GitLab.
3. You should provide a single UML diagram showing both ADTs
4. Implement the passenger ADT
5. Implement the elevator ADT
6. Make use of the C++ STL for the queue ADT, priority-queue ADT and their associated iterators.
7. You will need to read in and parse the input file, using the initial information to configure the number of elevator objects and establish their starting floors. Then, you will need to place in a regular queue the elevator requests from subsequent lines of the file.
8. Your main routine will the cycle through the request queue and the elevator priority queues and process each (set of) request(s) for the elevator objects according to the algorithm described above.
9. As each request is dequeued and processed, your program should output:
   - the request,
   - the elevator it is assigned to,
   - the location of each elevator (by floor) at the time of the request,
   - how many people will enter the elevator and how many will leave the elevator at the given floor, and
   - the distance of the requester's destination floor from the head of the assigned elevator's priority queue (in terms of elements).
10. All iteration through the queues must be done with C++ iterators and you are encouraged to use as many of the C++ STL built-in algorithms that make use of iterators in the implementation of your solution.
11. The program terminates when the request queue is empty and all elevator priority queues are empty.
12. You can ignore travel time of the elevator for this simulation; allowing the elevator to move to each destination floor in unit time.
13. Provide an algorithm analysis of:
    - the wait-time for a passenger requesting any specific floor in the building.
    - the amount of travel-time for a passenger to their destination floor.
    - include your analysis in a file in your project folder.

Figure 1: Programming Project Grading Rubric

Attribute (pts)	Exceptional (1)	Acceptable (0.8)	Amateur (0.7)	Unsatisfactory (0.6)
Specification (10)	The program works and meets all of the specifications.	The program works and produces correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results, but does not display them correctly.	The program produces incorrect results.
Readability (10)	The code is exceptionally well organized and very easy to follow.	The code is fairly easy to read.	The code is readable only by someone who knows what it is supposed to be doing.	The code is poorly organized and very difficult to read.
Reusability (10)	The code could be reused as a whole or each routine could be reused.	Most of the code could be reused in other programs.	Some parts of the code could be reused in other programs.	The code is not organized for reusability.
Documentation (10)	The documentation is well written and clearly explains what the code is accomplishing and how.	The documentation consists of embeded comments and some simple header documentation that is somewhat useful in understanding the code.	The documentation is simply comments embedded in the code with some simple header comments separating routines.	The documentation is simply comments embedded in the code and does not help the reader understand the code.
Efficiency (5)	The code is extremely efficient without sacrificing readability and understanding.	The code is fairly efficient without sacrificing readability and understanding.	The code is brute force and unnecessarily long.	The code is huge and appears to be patched together.
Delivery (total)	The program was delivered on-time.	The program was delivered within a week of the due date.	The program was delivered within 2-weeks of the due date.	The code was more than 2-weeks overdue.

The *delivery* attribute weights will be applied to the total score from the other attributes. That is, if a project scored 36 points total for the sum of *specification*, *readability*, *reusability*, *documentation* and *efficiency* attributes, but was turned in within 2-weeks of the due date, the project score would be $36 \cdot 0.7 = 25.2$.

Project Grading:

The project must compile without errors (ideally without warnings) and should not fault upon execution. All errors should be caught if thrown and handled in a rational manner. Grading will follow the *project grading rubric* shown in figure 1.

Collaboration Opportunities:

You may optionally work with one (1) other fellow classmate to develop a single solution. You ***must*** indicate in your code (via comments) the contributions of each classmate in order for credit to be awarded. Failure to indicate the contributions of all collaborative work will likely result in credit not be awarded fairly - there will be no modification of award credit without source code justification of contribution - PERIOD.