CSCI 2235
Programming Assignment 03 - Simulation

Assigned: October 08, 2019
Due: October 25, 2019 @ 23:00h

Purpose

- Explore the implementation and use of Queues, Deques, and Stacks
- Explore simulation as a means to answering "what-if" questions.
- Gain experience using the IEEE format for

Problem Statement

The TSA (Transportation Security Administration) has had trouble recently balancing customer service with operating costs. At many airports, they don't have enough security lines open making travelers angry, because they have to wait in line for so long. However, at other airports, they have too many lines open, which costs money (paying the personnel and managing extra equipment). The Director of the TSA has decided that the agency needs software that will tell each airport how many lines to keep open. Luckily for them, your are the Chief Data Scientist of Bloated Government Contracts, Inc. and are willing to produce this for only several million dollars.

You need to build a simulator for wait times in a line. The input for your program will be an integer representing the arrival rate (average number of people arriving at the security checkpoint per minute) as well as an integer representing the maximum number of lines that can be open (the user will pick that based on personnel and equipment availability). The arrival rate will be used to determine the number of people that arrive to the checkpoint each minute. The output for your program will be the average wait times (in minutes) of travelers at the checkpoint over a 12 hour period (720 minutes) for each potential number of lines. In other words, the user will determine what they expect the arrival rate to be that day (e.g. average of 18 people per minute) and the maximum number of lines they are able to open (e.g. 10) and will get output from your program that looks like this (you numbers should be close, but probably not exact). This will enable them to determine the best number of lines to open.

For a maximum arrival rate of 18 people/per minute and maximum number of lines at 10 and an initial seed of 1024 you should see the following values (each +/- 1):

```
Arrival rate: 18
Average wait time using 1 queue(s): 320
Average wait time using 2 queue(s): 280
Average wait time using 3 queue(s): 240
```

```
Average wait time using 4 queue(s): 200
Average wait time using 5 queue(s): 160
Average wait time using 6 queue(s): 120
Average wait time using 7 queue(s): 80
Average wait time using 8 queue(s): 41
Average wait time using 9 queue(s): 3
Average wait time using 10 queue(s): 1
```

The general flow of your simulation will follow this pattern: Every minute a random number of people arrive to the security checkpoint (call **getRandomNumPeople()** to get that random value). Each of those people (one at a time) join the line that is currently shortest. After each person joins a line (still in the same minute), two people (or fewer if there aren't that many people in the line) from the front of each line proceed through the checkpoint (leave) and have their wait time recorded. Do this each minute for 720 minutes and calculate the average wait time for people that got through. Don't worry about people that remain in the lines after 720 minutes are over. That is one iteration for the given number of lines. Since we need realistic values, run this for 50 iterations with the same parameters and average the average wait times. This is the value that you will report for the given number of lines. Do this for each possible number of lines (up to the value passed).

As an example, suppose I have some arrival rate and the maximum number of lines are two. I'll start with having one line for everyone. For each minute of the 720 minutes, I'll add **getRandomNumPeople()** to that line and remove two from the front (recording their wait time). After the 720 minutes are over, I'll determine the average wait time for people that got through over the past 720 minutes and put that value somewhere. I'll then do the same exact thing 49 more times. After that is complete, I'll average those 50 values to get the average wait time using one line. Then, I'll move to two lines. Each minute I'll add **getRandomNumPeople()** to the line(s) (person 1 picks the shortest, person 2 picks the shortest, …) and remove two from the front of each line. After the 720 minutes, I'll determine the average wait time for everyone that got through (regardless of the line they were in). I'll do this 49 more time, average the results, and report everything.

As part of such an experiment I would probably think of arrival rates that represent different times of day. This would help to explore the relationship between expected line lengths, time of day (arrival rate), and then line type. Since this is not a statistics class I will not be requiring you to execute the proper analyses to explore such relationships, rather a simple argument regarding this will be enough. Though, if you are aware of the proper analytical techniques do not shy away from using them.

Assignment

1. Implement a **LinkedQueue** implementing the **Queue** interface backed by a **SinglyLinkedList** or **DoublyLinkedList** of your own making. When a person enters the line this is via **offer**. When a person leaves this line it should be via **poll**.
2. Implement a **LinkedDeque** implementing the **Deque** interface backed by a **DoublyLinkedList** of your own making. When a person enters the line this should be either an **offerFirst** or **offerLast** (selected at random). When a person leaves this line it should be an **pollLast** or **pollFirst** depending on which has been waiting longer.
3. Implement a **LinkedStack** implementing the **Stack** interface backed by a **SinglyLinkedList** or **DoublyLi** of your own making. When a person enters the line it should be via **push** and when they leave the line it should be via **pop**.
4. Implement the **Simulation** Program initially using a **LinkedQueue** to represent the lines, and verify that it works.
5. Any methods/constructors defined should not be renamed or their headers changed (for testing purposes).

6. Build the classes/methods needed to execute the simulation described.
7. Setup the **Simulation** to be able to use the **LinkedQueue, LinkedDeque,** and **LinkedStack** to represent the lines.
8. Collect data regarding the different line types and number of lines. Complete a report of your results using the following as a guide:
    - Document format: IEEE Conference format
    - Minimum Length: 2 pages
    - Maximum Length: 5 pages
    - The report will need to have the following sections:
        - Introduction – A brief summary of the project and motivation for the study. This should include the problem statement in your own words and the main hypothesis you are investigating.
        - Background and Related Work (optional) – A synthesis of related work and related studies connecting this work to other work in the area.
        - Simulation Design – A description of the simulation, how it works, and what was to be studied.
        - Experimental Methods – The design of the experiment and what is to be compared. This should indicate how the simulation approach will be used to investigate the hypothesis you are studying.
        - Results – The actual results of the experiment. A chart/graph of the data would be useful here along with a table summarizing what was found. Do not simply put all the raw data in the report. You should also describe the results in paragraph form.
        - Analysis and Interpretation – Analysis of the results indicating the importance of the findings and their meaning in the context of the problem.
        - Conclusions – A summary of the paper, the key findings and their interpretation in the context of the problem, and paths towards future studies.
        - References (in IEEE Format)
    - Note: There are two templates (**template.docx** and **template.tex**) for the report, a Word template and a LaTeX template, in the **report** folder of the project. You are welcome to use either, but the end result should be stored as **report.pdf** in the **report** folder of the project.

Submission

Zip compress your project directory into a file named: [first_name]_[last_name].zip (Note: square brackets not included). Submit the file to Moodle by the deadline noted above.

Grading (50 points)

- Implementation of **LinkedQueue** – 5 points
- Implementation of **LinkedDeque** – 5 points
- Implementation of **LinkedStack** – 5 points
- Implementation of **Simulation** – 10 points
- Report – 25 points (breakdown as follows):
    - Format – 5 points
    - General Writing – 5 points
    - Results, Analysis, and Conclusions – 15 points

Hints

1. Implementing the Queue, Deque, and Stack should be done via Wrappers of LinkedLists.
2. I hope you have learned from the Sorting and Search experiments how to store data from experiments for later evaluation.