Evan Vu

4/20/20

Project 2

1. Introduction

The purpose of this project is to determine the most optimal number of cashiers for a coffee shop to have. The project provides an event-driven simulation to run through operation of a coffee shop from 6am to 9pm. Different data structures were utilized in this project, including ArrayList, ArrayDeque and PriorityQueue.

2. Approach

I. Design

The Coffee Shop is simulated in an event-driven fashion, which means it decides the next action based on a list of events, instead of based on the time passed. This idea is adapted from (Weiss). The Coffee Shop also has a number of cashiers and customers.

Everytime a new Arrival event is removed from the eventSet, a customer will also be added to either the wait line or a cashier line. This depends on if there is an empty spot on the cashier line. An empty spot is signified by a place-holding Customer object that has waitTime 0. This is unique since all Customers have waitTime more than 0. If there are now spots on either the cashier lines or the wait lines, the Customer will be turned away as an overflow. As the Customer is added to a cashier line, a departure event will be created for that Customer in the eventSet, ensuring that they will be served and removed.

II. Choice of Data Structure

The experimenter chose to store Event objects in a PriorityQueue (eventSet). The cashierLine is represented by an ArrayList of Customers. The waitLine is represented by an ArrayDeque of Customers.

III. Classes and Functions

The Customer object stores variables lineNum, waitTime and profit. The lineNum variable indicates which line in the cashierLine the Customer will be served at. The waitTime and profit variables are randomly generated according to given bounds.

The Cashiers class stores the cashierLine and the waitLine. The cashierLine by default will contain a number of place-holding Customers with waitTime 0. The waitLine by default will be empty. Every time a new Customer is to be added, the findEmpty method will be called to find if there is an empty cashier in the cashierLine. If there is a spot, it will return an int indicating the position of the line. If there is no spot in the cashierLine, it will return -1, which is stored in the Customer object as lineNum. The addToCashierLine and addToWaitLine methods add a new Customer to either the cashierLine or waitLine. The remove method has 2 different cases. First, if the removed Customer is currently in the cashierLine, it replaces it with the top Customer in the waitLine and then returns the new Customer in the cashierLine. Second, if the removed Customer is removed while the waitLine is empty, it returns null.

The CoffeeShopSim class contains addArrival, arrival, departure and runSim methods. The method addArrival will set up the eventSet with only arrival events from an input file. On arrival, the arrival method will be called. However, if there is no spot left (ensured by the variable spots), the arrival will be turned away as overflow. After each arrival to the cashierLine, a departure event will be created for the Customer. However, if they arrive in the waitLine, no departure will be created yet. This is due to complicated calculation of the waitTime of all Customers being served before it. In the departure method, Customers in the cashierLine are replaced by Customers from the waitLine, and then their new departure events can be created. The time of departure in this situation can be calculated as the sum of the time of departure of the old Customer and the new Customer's waitTime.

<u>3. Methods</u>

The experimenters ran all of these tests on a Chromebook that runs Debian Linux. The relevant specifications of the machine are: Intel Celeron 1.6GHz, 4GB RAM. The compiler used to run these tests is BlueJ 4.1.4, using Java SE8. Each selection of cashier number was run in 6 trials, with the final data being averaged out throughout the five. The experimenter ran different cashier numbers from 1 to 10. Data analysis was created using Excel.

<u>4. Data and Analysis</u>

Cashier Number	1	2	3	4	5	6	7	8	9	10
Trial 1	610	322	109	7	0	0	0	0	0	0
	952	1776	2323	2441	2105	1889	1586	1291	1014	697
Trial 2	604	332	118	12	0	0		0	0	0

	992	1754	2365	2353	2200	1918	1621	1223	986	680
Trial 3	616	321	116	13	0	0		0	0	0
	951	1788	2358	2397	2165	1851	1562	1347	980	703
Trial 4	603	335	103	21	0	0		0	0	0
	970	1735	2394	2398	2150	1849	1557	1215	1007	676
Trial 5	605	316	113	17	0	0	0	0	0	0
	997	1837	2304	2473	2212	1873	1608	1251	957	638
Trial 6	611	325	103	6	0	0	0	0	0	0
	890	1773	2344	2437	2183	1858	1550	1291	969	690
Average Overflow	608	325	110	12	0	0	0	0	0	0
Average Profit	959	1777	2348	2416	2169	1873	1580	1269	985.5	680

Figure 1: Recorded overflow and profit in different trials and cashier numbers

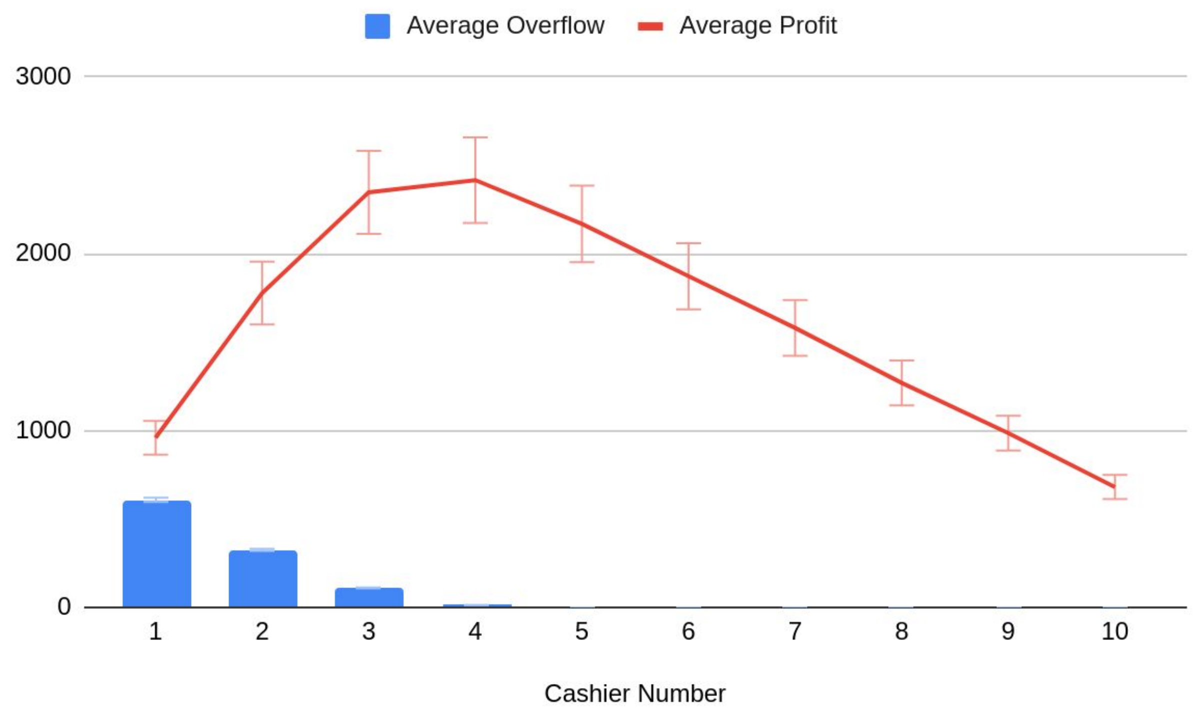

Figure 2: Plot of overflow and profit versus number of cashiers

After six different trials with each cashier number from 1 to 10, the experimenter plotted the average overflow and profit accordingly to each number of cashiers. The most overflows observed is in choosing to have 1 cashier. The most profit observed is in choosing to have 4 cashiers. Overall, there is evidence to support that the best choice of number of cashiers is around 3 and 4 cashiers.

5. Conclusion

After building an event-driven simulation of a coffee shop, the experimenter obtained a program that would take in different numbers of cashiers and produce average total overflow and average total profit. It can be concluded that, to minimize the number of overflows and optimize profit, the number of cashiers to be chosen should be 4 or 3.

6. References

Oracle. “ArrayDeque (Java Platform SE 8).” *Java Documentation*, Oracle, 2020, docs.oracle.com/javase/8/docs/api/java/util/ArrayDeque.html. Accessed 21 Apr. 2020.

Oracle. “ArrayList (Java Platform SE 8).” *Java Documentation*, Oracle, 11 Sept. 2019, docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html.

Oracle. “PriorityQueue (Java Platform SE 8).” *Java Documentation*, Oracle, 2020, docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html.

Weiss, Mark Allen. *Data Structures and Problem Solving Using Java*. Wesley, 2011.

Xia, Ge. *CS150 Lectures*. Lafayette College, 2020.