Abiola Gabriel Olofin

CS 150-02

Professor Falconi

04/26/2020

Lab 11 Report

Introduction

In this lab, we were asked to build our own hash table and analyze the performance of the insertion and find operations under different conditions. The hash table would be an array and each index would hold a linked list of a student object. In other words, we are creating our hash table data structure and based on a given file holding student's LNumber's and names, how fast it takes different table sizes to input the data and search the data for specific elements. Then, we will create a graph that compares the different hash table sizes and the different times of completion. Using these comparisons, we would discover the optimal range for our hash tables size and how long we expect the insertion and searching to take.

Approach

To complete this lab, the first step was to create the Student class. In this class, the constructor took in 3 Strings as parameters to populate the variables of the student class. Those variables are firstName, lastName, and studentID. The rest of the class were getter and setter methods that either change or showed the values of a Student object.

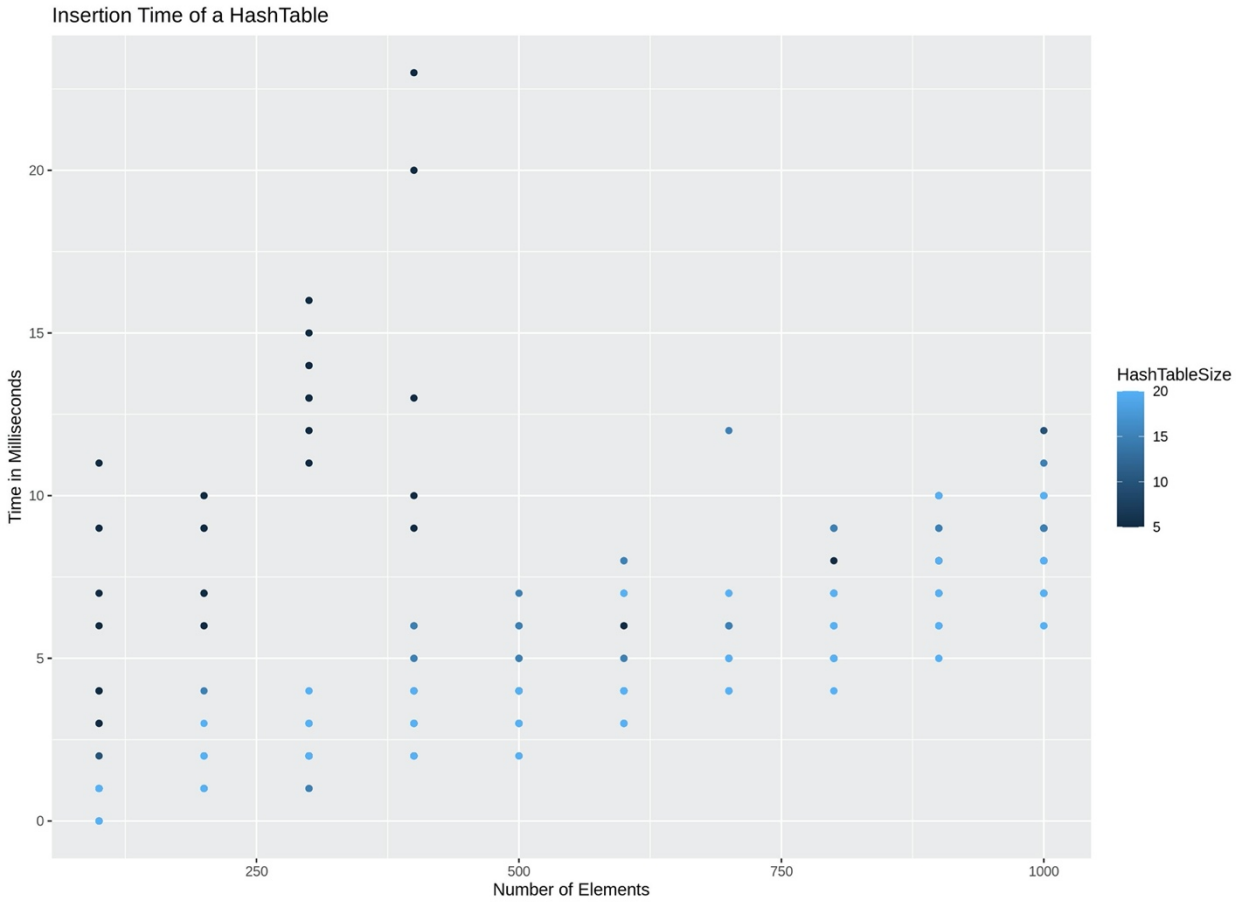
The next step was to create MyHashTable class. This is the class that will contain my implementation of a hashtable. The class first creates a variable of type array of

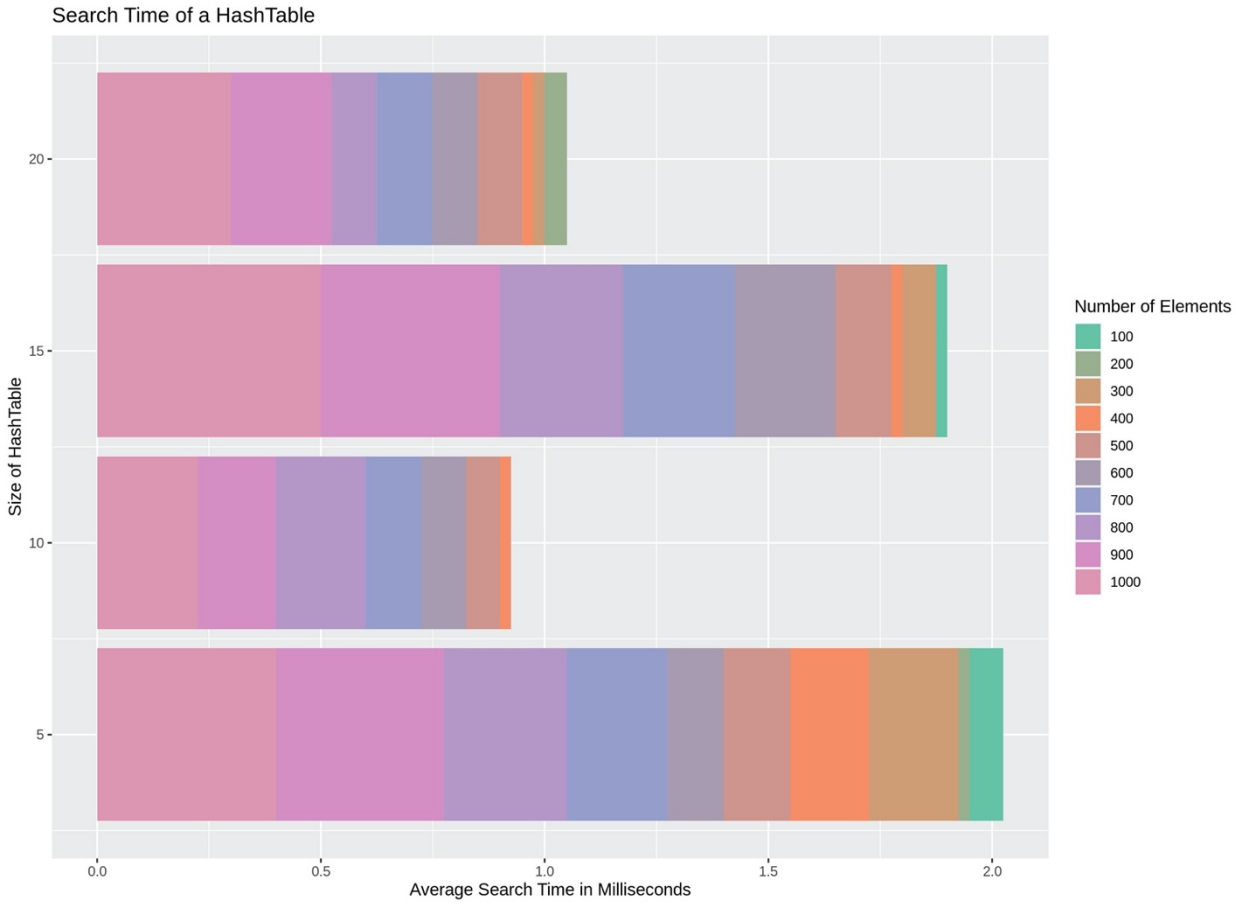
LinkedList<Students>. In the constructor, the parameter which is an integer, is used to determine the size of the array that is the hashtable. In the methods, we had to create an insert method that would insert a student into a bucket(index) in the hashtable based on their hashcode which gives the “address” of the bucket that they’re going to. The method does allow for the same object to be listed twice. The method also returns true or false whether the student was successfully inserted or not. We also had to create a method when given an ID returns the student with that ID, and another method that given an ID returns and removes the student with that ID.

Finally, the experiment controller runs the entire simulation from grabbing the values from a CSV to computing the time it takes to insert and find the values and producing the CSV file for the graph.

Data

After running the entire program to get 400 different outputs based on different hash sizes and number of elements inputed and searches, I created a scatterplot and stacked bar graph that showed my findings.


Insertion Time of a HashTable
Time in Milliseconds
0
5
10
15
20
250
500
750
1000
Number of Elements
HashTableSize
20
15
10
5



Based on my data, the optimal size for a hashtable insertion and search is having a hashtable with 20 indexes.

Conclusion

In conclusion, this lab taught us to understand how a hashtable works from the inside and taught us the optimal size for a hashtable if needed.