Agathe Benichou
CS 150: Data Structures and Algorithms
Professor Liew
Due: Saturday November 14th 2015

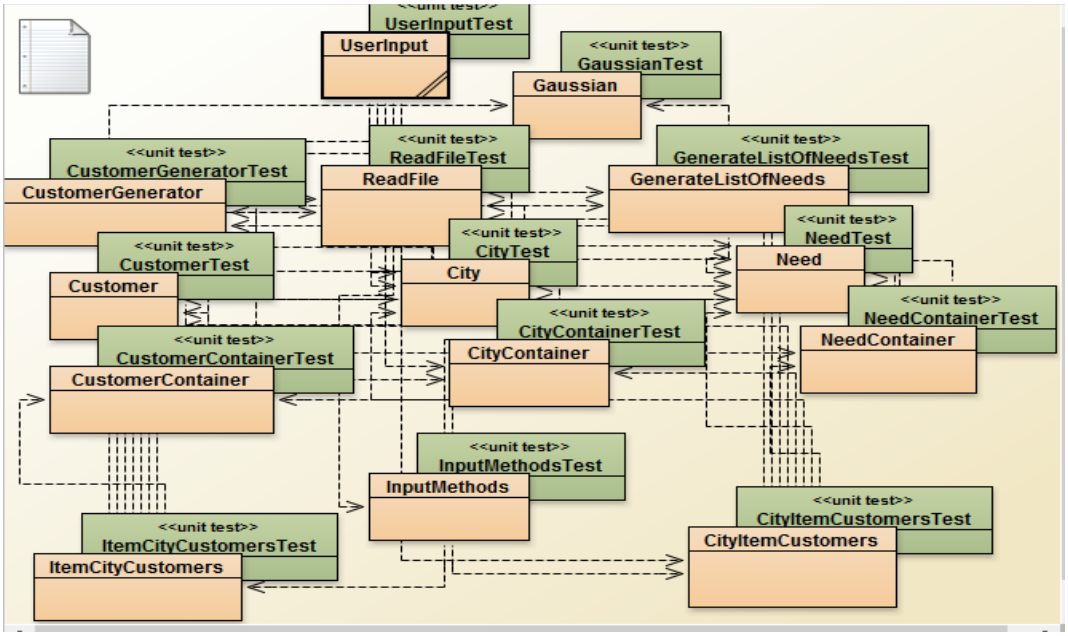
Project 2: Final Report

1. Introduction

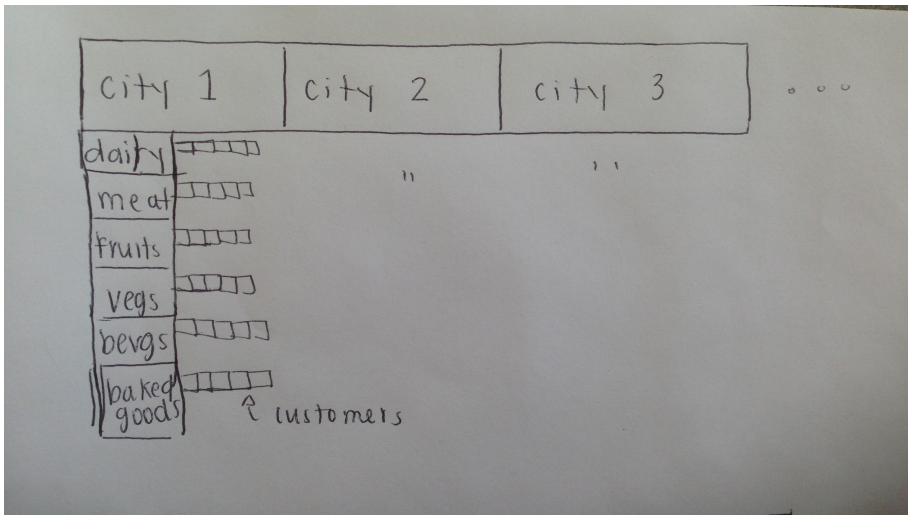
This goal of this project is to help identify and develop effective advertising outlets for the Easton Farmers Market and for vendors who are part of the market. In the previous project, we created a simulation of customers entering the Eastons Farmers Market with a list of needs from the market and going from stall to stall in the market getting all the needs they had on their list. In this project, the customers are still generated with their own list of needs however they are randomly stored in cities and sorted in data structures based on the city they are from and their needs. This makes it easy for the advertising outlets to look up information about cities and goods to give to the vendors. This way the vendors can know which of the cities the customers buy most of a certain item. This makes is efficient and effect to provide information to the vendors about the customers and their individual interests which will help the vendors decide which city they want to go to to sell their goods. The data structure I used in this project will be described in the Approach section of the report and analyzed in the Data and Analysis section of the report. However, I think that my data structure will have O(1) insertion and O(n) search/access time because of the insertion and search/access times of the data structures within it.

2. Approach

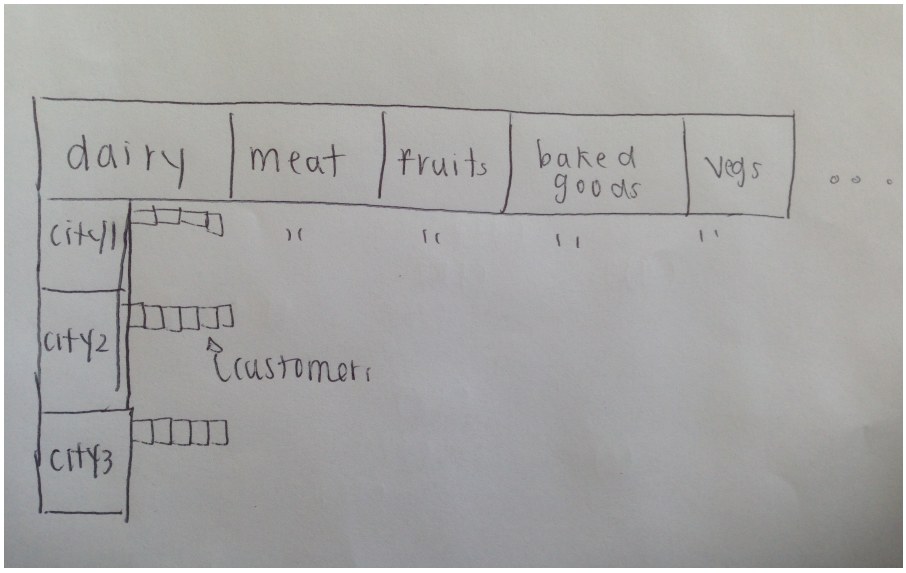
This program was built via input commands. The user who runs the program has the power of inputting certain commands. This program will read a test file of city names where each line is the name of the city with the “init” command. The text file is read using the Scanner class and the names of the cities created as City objects and added to a Linked List. The city text file is read in the ReadFile class and the cities are made into objects with the City class and stored into a container in the CityContainer class. This makes for O(1) insertion of the City objects and O(n) retrieval of the City objects.



This program can be run using the “run #” command, where # is the number of times the user wants to run the simulation. Each time the simulation is run, more customers are generated in the CustomerGeneration class, they are created into Customer objects with the Customer class and they are stored in the CustomerContainer class. The Customers are assigned a list of needs which is created in the GenerateListOfNeeds class. These list of needs is an Array List which stores Need objects created in the Need class and the entire list of needs is stored in the NeedContainer class. These classes, especially the various Container classes are made available for the classes which store the bigger data structure which store the information about Customers and their needs. This command will assign the Customers to cities and store them in the data structures according to their city and need. The two main data structures used in this program to store information are two forms of Hash Maps. They are both a Hash Map with Strings as their keys, another Hash Map as their values and this inner Hash Map has Strings as their keys and Linked List of Customers are their values. Hash Maps have an insertion, search, access and deletion time of O(1). These nested data structures makes for a total of O(1) insertion time and an O(n) search/access time. There are two of these data structures because of the different user input commands. The first data structure outer Hash Map stores keys in the form of Strings which represent the City names and values in the form of an inner Hash Map which stores keys in the form of Strings which represent the item name of the good and whose values is a Linked List of Customers in that city who have that item on their list. Below is a visual of this first data structure which will be referred to at the City-Item-Customer data structure.



The second data structure outer Hash Map stores keys in the form of Strings which represent the Items names and values in the form of an inner Hash Map which stores keys in the form of Strings which represent the Cities and whose values is a Linked List of Customers who have that item on their list and are from that city. Below is a visual of this first data structure which will be referred to at the Item-City-Customer data structure.



When the simulation is run, the same city file is read and the same customers are generated with the same list of needs. But the customers are stored in both data structures. I decided to make two data structures instead of one because of the three most important user input commands. Some commands would take too long in one data structure so another data structure was needed to make sure that the search time remained O(n). The City-Item-Customer Hash Map is created and the methods to store information in that Hash Map is in the CityItemCustomer class and the Item-City-Customer Hash Map is creates and the methods to store information in that Hash Map is in the ItemCityCustomer class. The three most important user input commands are the ones that retrieve information, these are the inputs that the advertising outlets use to retrieve information for the vendors.

The first of the important retrieval commands is the “list” command which takes in the name of a city and the name of an item returns all the Customers in that city who have that item on their list of needs. This input command could have been used in either of the data structures but since the name of the city is the first input to the “list” command, I decided it should use the City-Item-Customer command and in the CityItemCustomer class. This command takes in the name of the city and searches for that inner Hash Map whose key is the name of the city, then it takes the name of the item and searches for the Linked List values whose key is the name of the item. It returns that Linked List of Customers. This “list” command should be done in O(n) time since search and access of this data structure is O(n).

The second of the important retrieval commands is the “findgoods” command which takes the name of a city and returns all the goods from most purchased to least purchased in that city. This command is clearly better for the City-Item-Customer Hash Map and in the CityItemCustomer class because it searches through the outer Hash Map for the inner Hash Map with the name of the city. It then records the sizes of all the Linked Lists of Customers for every item. With an item and the amount of Customers who have that item on their list, the program creates a Tree Map which stores the amount of customers as the keys and the item name as the value. This Tree Map sorts itself and it returns the items in order of most purchased in that city to items least purchased in that city. The “findgoods” command should be done in O(n) time since the search and access of this data structure is O(n).

The final of the important retrieval commands is the “findcities” command which takes the name of ani

“item and returns all the cities from which city had customers who most purchased that item to which city had customers who had least purchased that item. This command is clearly better for the Item-City-Customer Hash Map and in the ItemCityCustomer class because it takes the name of the item and searches through the outer Hash Map for the inner Hash Map with the name of that item. It then records the sizes of all the Linked Lists of Customers for every city. With a city and the amount of Customers who have that item on their list, the program creates a Tree Map which stores the amount of customers as keys and the city name as the value. This Tree Map sorts itself and it returns the cities in order of which city has purchased the most of that item to which city has purchased the least of that item. The “findcities” command should be done in O(n) time since the search and access of this data structure is O(n).

The “add” input command takes in the name of a new city and adds that city to the City Container in the CityContainer class so that the next time the program is run, that city is taken into account and stored in both Hash Maps and customers are assigned to that city. The “clear” input command clears out all the information in all the data structures. The “exit” input command exits out of the input command terminal and terminates the user input. All of these commands are called in the UserInput class which takes in the users input and passes it to the InputMethods class which calls the methods in the CityItemCustomer class and the ItemCityCustomer class so that the InputMethods class is kept as clean as possible and all implementation is pushed down.

3. Methods

To test my program and the various data containers, I created many tests which test the insertion and search/access of the containers as needed. Not all containers are used to insert and search, some are only used for insertion. I did not test the containers and classes that are from my other project such as the GenerateListOfNeeds class and the CustomerGenerator class.

The “init” command stores its information in the City Container which is a Linked List of City Objects in the City Container class. This graph should be a constant graph since insertion to a Linked List takes O(1) time. To test this container, I ran this “init” command 10 times and recorded the time taken to store the City Objects into the City Container. The “add” command takes in the name of a new city, creates a City object and inserts that City object to the City Container in the City Container class. To test this container, I ran this “add” command 10 times and recorded the time taken to store 10 different new City objects to the City Container. This graph should be a constant graph since insertion to a Linked List takes O(1) time. The “run #” command stores the information from the Need Container in the NeedContainer class, the City Container in the CityContainer class and the Customer Container in the CustomerContainer class into two different but same Hash Maps that were described in the previous section. The “run #” command is only for insertion into these nested data structures which is a time of O(1). To test insertion into these data structures, I ran the simulation 10 times, each time increasing the number of runs and recorded the times taken to insert information. This graph should be a constant graph. To test the search/access of these data structures, I can test the “list”, “findcities” and “findgoods” commands which retrieves information from these data structures based on inputs such as city and item names. The retrieval time of my nested data structures is O(n) because the search/access time of a Linked List is O(n). I can test this by running my simulation 10 times while increasing the number of runs each time so that there will be more information to retrieve each time for all of the commands. I systematically generated my data by running my simulation using the “run # ” command. I started with # = 1. At run = 1, with 238 customers inside the data strucutre, I timed how long it took the three most important command to return desired information to me. For each command for run = 1, I put in 5 different parameters (this means different cities, different items and different city – item combos) and acquired 5 times for each command. I took the average of those times and graphed it. Each of the three important command got their own graph which shows how long it took to search up and return information in my data structures. I increased the # until it reached 10 runs so the number of customers stored in the container was 2582 customers.

4. Data and Analysis

The Needs Container is created during the first run of the program by creating a new Need object for each of the needs (fruits, vegetables, meat, dairy, baked goods, beverages). The Customer Container is filled based on the Customer Simulation. The Needs Container is an Array List and the Customer Container is a Linked List. These lists were tested in my previous project report and are not tested in this one. However, since they are both Lists: they have O(1) insertion time and O(n) access/search time.

The City Container is stored by reading the cities text file using the “init” user input command, taking those city names to create City objects and then inserting those City objects to a Linked List which is the City Container. The classes with the larger data structures call this City Container to fill themselves during each run. I ran the init command 10 times and timed how long it took to be filled and each time it took O(1) time. The insertion time of a Linked List is O(1) and the graph is a constant graph of 1.

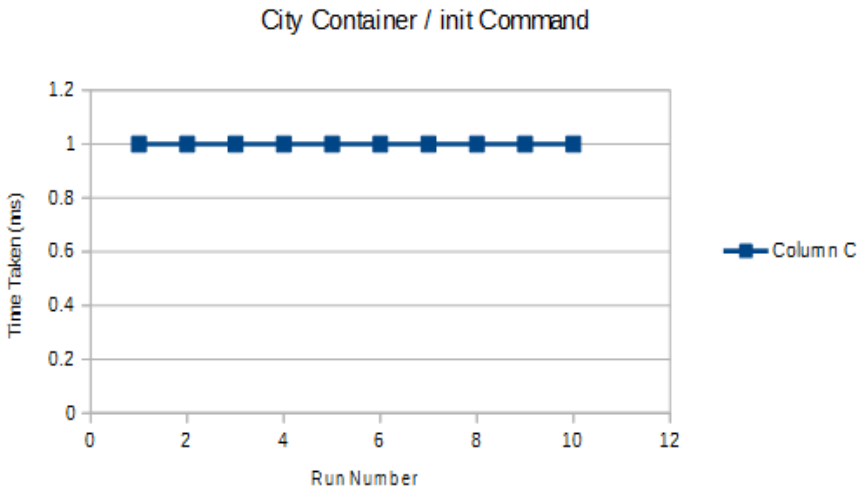
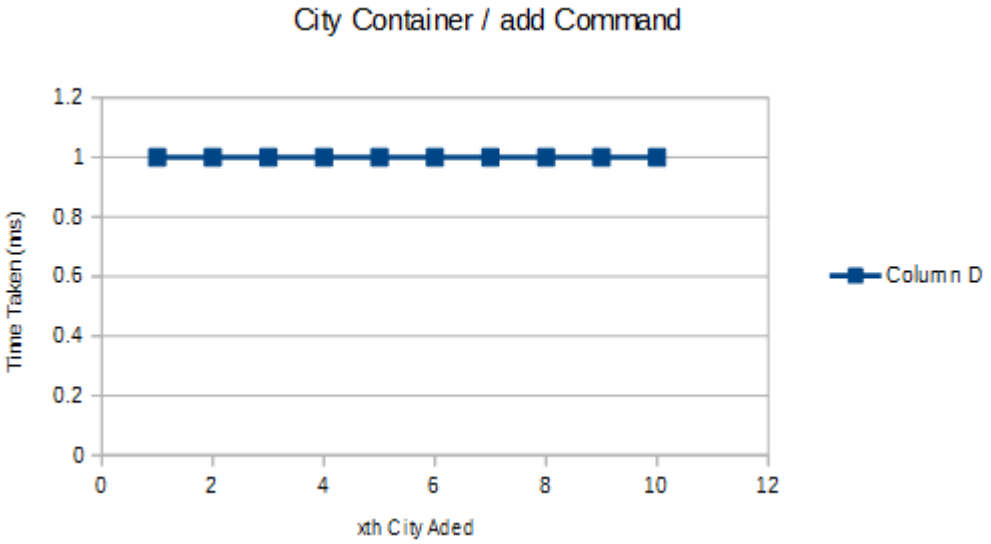


Figure 1: The O(1) insertion time of the City Container via init command

The City Container can gain cities if the user inputs the command “add” followed by the name of the new city. The new city will be created into a City object and inserted to the City Container. On the next run, this City is added to the larger data structures and Customers are assigned to that city. I ran the add command 10 times and each time I added a new city to the container. I timed how long it took to add that city to the container. The insertion time to a Linked List is O(1) and the graph is a constant graph of 1.



The “run #” command stores the information from the Need Container in the NeedContainer class, the City Container in the CityContainer class and the Customer Container in the CustomerContainer class into two different but same Hash Maps that were described in the previous section. The “run #” command is only for insertion into these nested data structures which is a time of O(1). To graph this, I ran the run command 5 times for each run. So I did the “run 1” command 5 times and took the average of those 5 times and graphed that number on the graph below. Each run number increased the number of customers in the data structures. I did this for each run up to run 10 with 2582 customers. The time graphed below is the average time of that run #. The graph is supposed to be constant time but it is a little skewed.

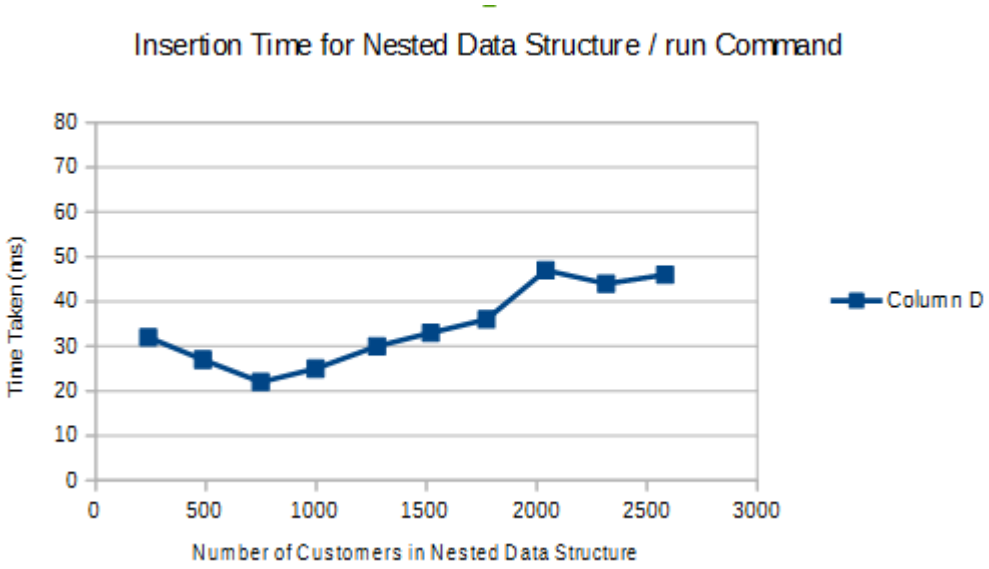


Figure 2: The O(1) insertion time for the Nested Data Structure via run command

The “list” command is one of the commands that I can test the search/access of my data structures with. This command takes a city name and an item name and searches through the first outer Hash Map with the city name for the inner Hash Map, searches through the inner Hash Map with the item name for the Customers in that City who had that item on their list. The search/access time of my data structure is O(n) because the search/access time of Hash Maps are O(1) and the search/access time of the Linked List is O(n). I ran the run command 10 times and for each run, I tested the list command with 5 different city and item names. I took the average of the times it took for those 5 city and item names and plotted that. I did that with every run and with every run, more customers were added to the data structure which means there was more information to return on each run.. The graph is a rough sketch of O(n).

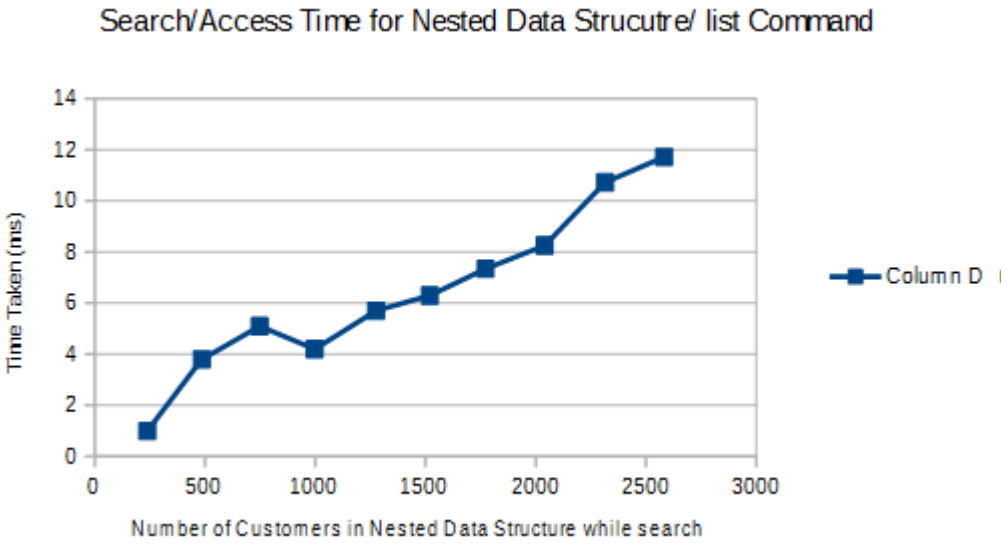


Figure 3: The O(n) search/access time for the Nested Data Structure via list command

I did the same procedure with the “findgoods” command and this is the graph I obtained. Again, it is a linear graph but with some difficulty in the middle.

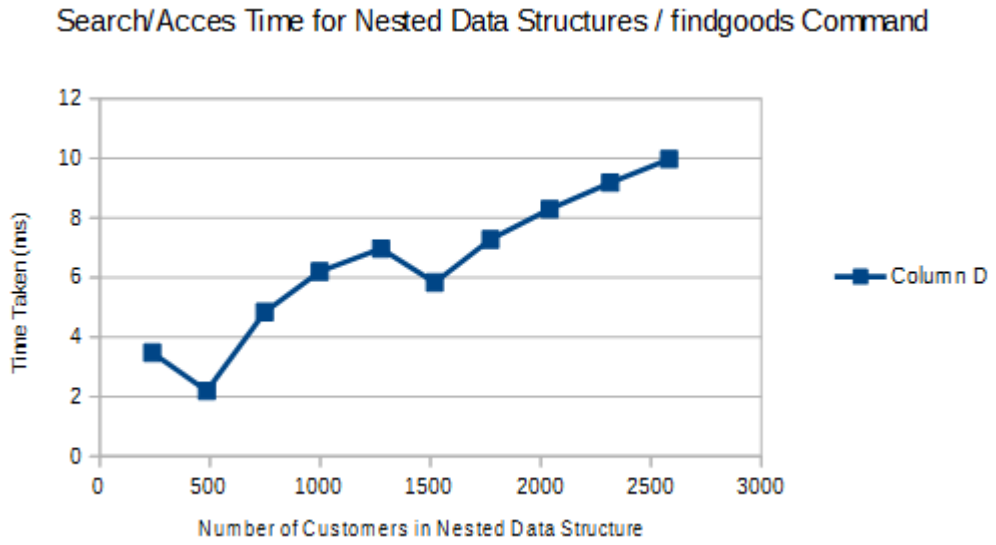


Figure 4: The O(n) search/access time for Nested Data structure via findgoods command

I did the same procedure with the “findcities” command and this is the graph I obtained. This linear graph was amount the smoothest linear graph.

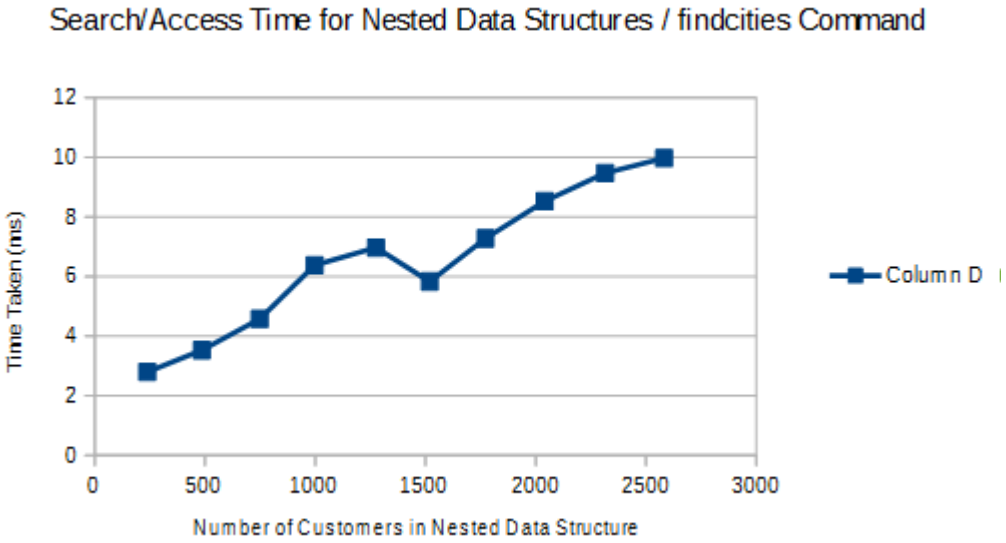


Figure 5: The O(n) search/access time for the Nested Data Structure via findcities command

5. Conclusion

The smaller List containers all have an insertion, search and access time of O(1) which was shown in Figure 1. The larger Hash Map/Hash Map/ List containers also have an insertion time of O(1) as shown in Figure 2 but it was more rocky and the graphs were not as smooth as the insertion time graphs for the smaller List containers. The larger Hash Map/ Hash Map/ List containers have a search/access time of O(n) as shown in Figures 3, 4 and 5 but this linear graph was not very smooth and it was harder to see the linear aspect. I think this is because of minor flaws in my program where maybe I am passing information through more methods than I need to. I am pushing information down from class to class to push my implementation down. The larger containers are the last to receive the input command compared to the smaller containers which are first to receive the input command and the information from the smaller container and that is why the graphs of the larger containers are not smooth and perfectly linear or perfectly constant. The performance varies for each command as the size of the database grows. Each time the size of the database grows by adding a run, the performance takes longer. This is because there are most customers and more information for the commands to access and it takes longer for the commands to gather and sort that information. I think that my choice of data structure (Hash Map – Hash Map – Linked List) was a very good design because the time complexities for Hash Maps are O(1) for everything and the worst time complexity for Linked List is O(n). I think that separating into two different Hash Maps where the keys are switched off was good because it made my findcities command equally as fast as my findgoods command. The only tradeoff I made by creating two different Hash Map data structures was that a lot of space was taken up to store all that data but it kept the time fast. This did not affect the performance of the commands.

6. References

"Duplicate Key in TreeMap." Java. N.p., n.d. Web. 14 Nov. 2015.

"HashMap (Java Platform SE 7)." HashMap (Java Platform SE 7). N.p., n.d. Web. 05 Nov. 2015.

"Java.util.TreeMap.descendingMap() Method Example." Www.tutorialspoint.com. N.p., n.d. Web. 05 Nov. 2015.

"Know Thy Complexities!" Big-O Algorithm Complexity Cheat Sheet. N.p., n.d. Web. 14 Nov. 2015.

"LinkedList (Java Platform SE 7)." LinkedList (Java Platform SE 7). N.p., n.d. Web. 05 Nov. 2015.

"Scanner (Java Platform SE 7)." Scanner (Java Platform SE 7). N.p., n.d. Web. 31 Oct. 2015.

"String (Java Platform SE 7)." String (Java Platform SE 7). N.p., n.d. Web. 31 Oct. 2015.

"TreeMap (Java Platform SE 7)." TreeMap (Java Platform SE 7). N.p., n.d. Web. 05 Nov. 2015.