Abiola Gabriel Olofin

Professor Ge Xia

CS 150

May 10, 2020

Project 3

Introduction:

In this project, we were asked to implement a program that creates a book index of any given book. The index will only list important words which are all the English words listed in the given book. The program will create an index using those important words and will put the line numbers where each word appears. The implementation of the program will be done using three different data structures which will help us answer the question, "What is the best data structure to use when creating an index". Before discussing the results of the implementations, I believe that the TreeMap implementation of the index will be the best implementation because TreeMap's sort each key by its natural order which means that whenever I make an entry into the TreeMap, it will sort itself into alphabetical order without any use of a sorting algorithm. This means that it will have a faster runtime when adding entries into the index and will print them out pretty quickly too since everything is in order like a regular book's index would as well.

Approach:

To implement program, we were asked to consider three approaches to use as the data structure of the index. Before implementing them, I had to create a interface because it would allow each implementation to have the same methods which makes running the program easier and it also allows me to compare the runtime of each knowing that they all use the same methods just

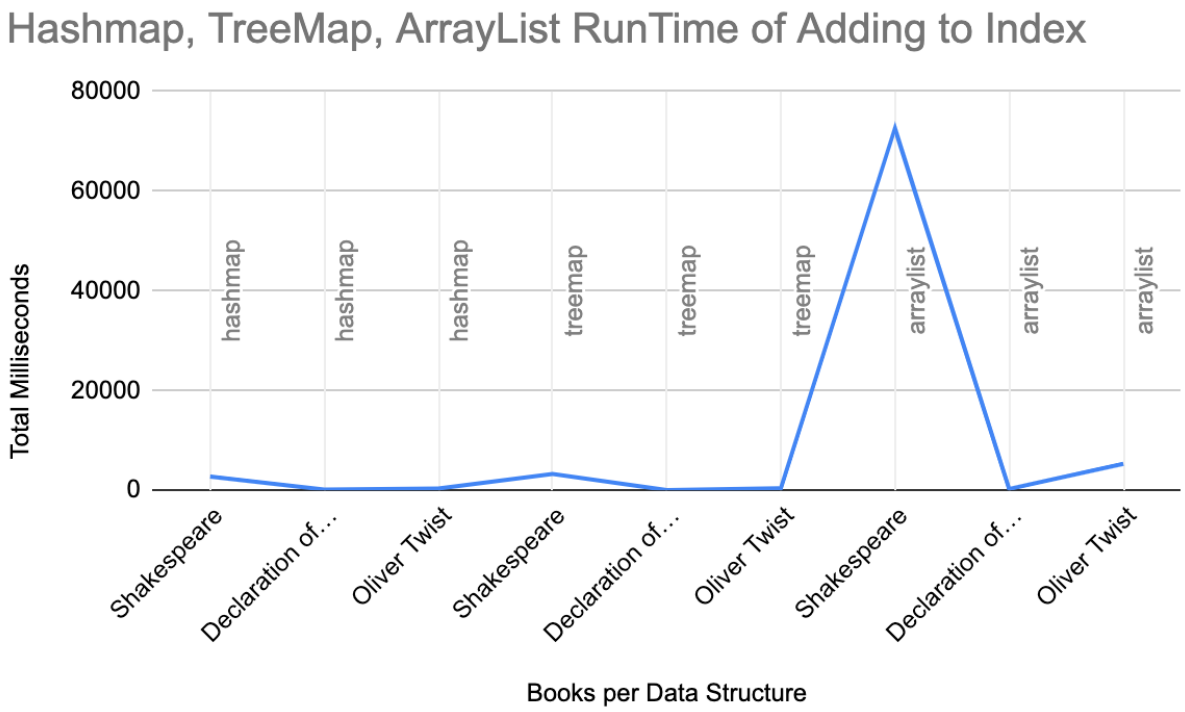
different implementations. The first was an ArrayList which we would sort as we added new Entry objects that contained the word and the line numbers where that word appears. Whenever a new word is found on a line, the arraylist will check to see if the entry of that word already exists. If it does, I will add the line number of that word to the list of line numbers that that word/entry has. If the word does not exist, it will create a new Entry object and add it to its sorted position in the arraylist. For the ArrayList, I had to make Entry an inner class which allowed me to store the "name" of the word as an instance of Entry and the line numbers associated with that word. Next, we were asked to use a TreeMap to implement the index. When a new word is found, if it does not already exist in the TreeMap, it would be added to the TreeMap; however, if the word exists, the line number found would be added to the list of line numbers of that word. Finally, we were asked to use a HashMap to implement the index. When a new word is found, if again it does not already exist in the HashMap, it would be added to the HashMap as new key; however, it it does, the line number would be added to the list of line number of that word and when printing out the index, it must be in alphabetical order.

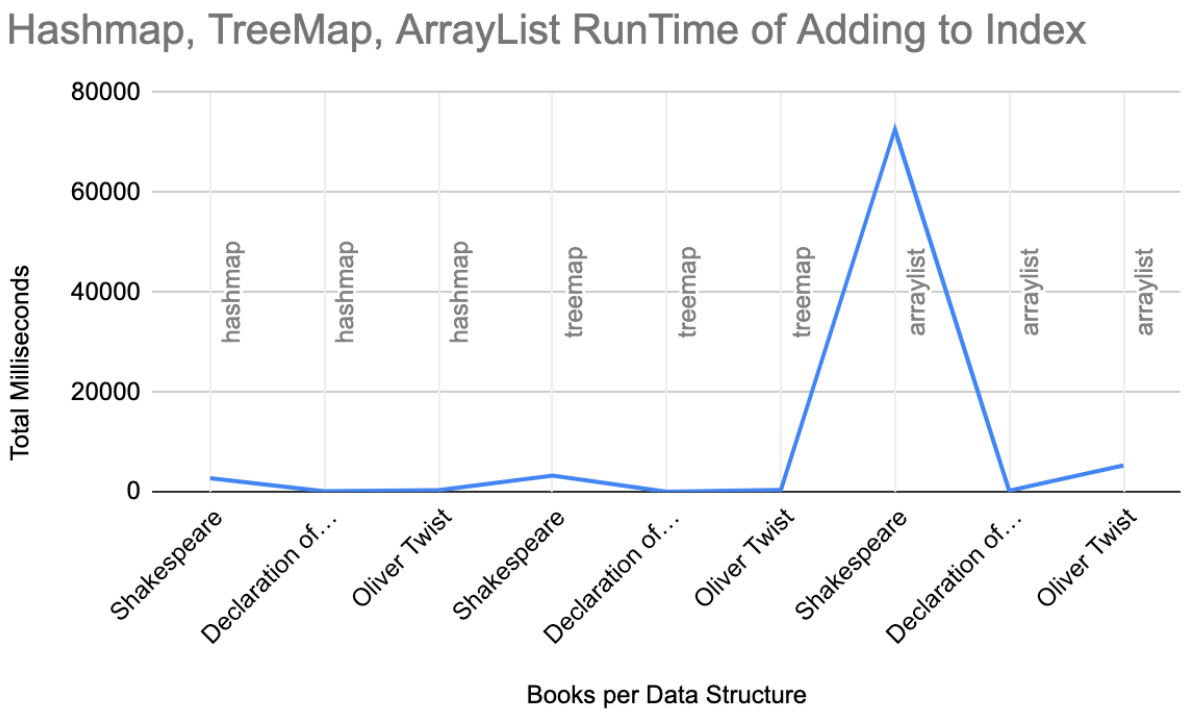
Methods:

To gather my data, I ran each implementation of the index 5 times in order to get an average runtime of both adding to the index and printing out the index. I also changed the length of the passages used. For example, I used 3 different text files of different lengths to figure out which index implementation works the fastest despite the size of the book being read. This will allow me to compare each implementation and whether or not size plays a factor in runtime and to figure out how consistent each implementation is and whether there's a certain limit to how many words an implementation can add to an index or print out from an index. I did not really

use parameters except for allowing someone using the implementation to change the name of the text file they want to have read and analyzed.

Data and Analysis:





According to my data, it appears that the TreeMap is the optimal data structure to use because despite the size of the book, the TreeMap does not take very long to add elements to the index and also does not take very long to print out the index into a text document in the correct order that an index should be in. This is due to the effectiveness of the TreeMap. A TreeMap like a previously stated, sort elements added into their natural order/position in a collection. This means that the user does not have to worry about taking extra steps to try and sort the elements in the TreeMap which saves time and lowers the runtime. Also, when printing the elements, since they are already in order and the values are saved under that element/key, it will also be very easy and fairly quick to print out the results and create the index in a text file.

Conclusion:

In conclusion, my original hypothesis was proven correct that a TreeMap is the best implementation of an index due to its ability to naturally sort elements. This ability allows it to take less time to both create the index and display the index making it the optimal data structure to use to solve this problem.